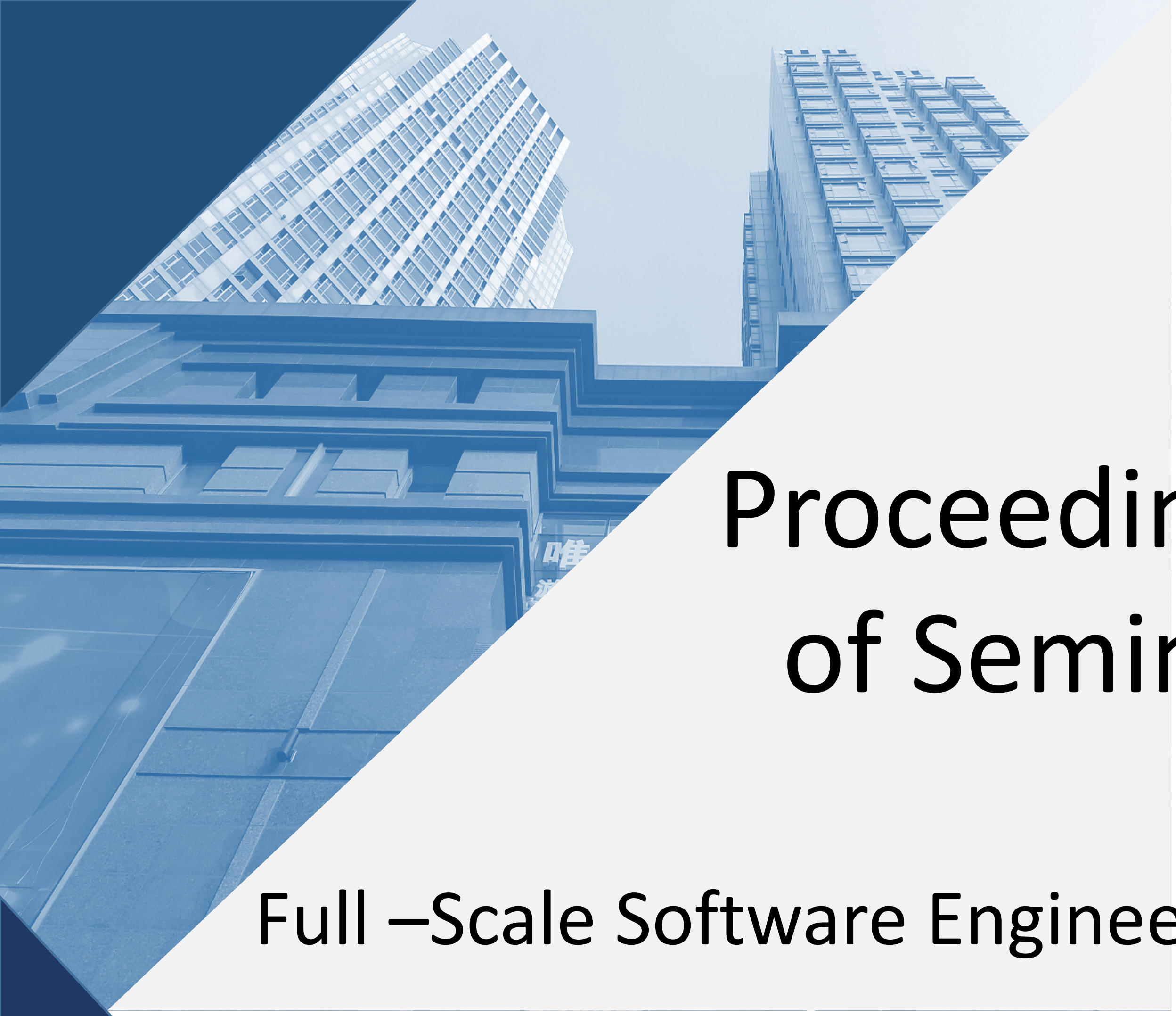


Proceedings of Seminar

Full –Scale Software Engineering

2019

Editors: Horst Lichter
Konrad Fögen
Christian Plewnia
Simon Hacks

Software Construction

RWTH AACHEN UNIVERSITY

A Systematic Literature Review on Functional Robustness Testing

Felix Klenner
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
felix.klenner@rwth-aachen.de

Tim Bolender
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
tim.bolender@rwth-aachen.de

ABSTRACT

With the increasing application of software in critical infrastructure, its functional robustness gains more and more focus. While a variety of techniques for functional testing are available, their benefits and disadvantages in robustness testing remain unclear. Currently, there is a lack of recent reviews and comparisons of robustness testing techniques. We therefore systematically review and analyze recent publications in the field. Using five different databases, we searched with common keywords as well as for well known functional techniques to get a broad understanding of available techniques. We see that fuzz testing and state transition testing received the most attention in the last 8 years. The latter promises a higher detection rate at the cost of higher efforts in the first place.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management—*productivity, programming teams, software configuration management*

Keywords

Robustness Testing, Functional Testing, Software Robustness, Literature Review

1. INTRODUCTION

As it can be found in nearly every area of life, software is nowadays almost omnipresent. This especially holds for critical infrastructure. Combined with an increasing complexity, the reliable provision of service becomes challenging. The question of functional robustness gains more and more relevance.

For regular functional testing, a wide range of black-box techniques is available. Unfortunately, existing literature reviews [38, 17] cover robustness testing only in a general sense. They therefore lack tangible measurements to evaluate the robustness of a program and improve it during development. Thus, we decided to investigate the available functional robustness testing techniques. To ensure the completeness of our search, we do so in a structured literature review. Based on previous reviews, we focus on contributions after 2012 (cf. [38]).

The paper is structured as follows: in Section 2, we introduce robustness in a more comprehensive way and present findings of other literature reviews in the field. This is followed by the explanation of our research goals an how we conducted our review in Section 3. Our findings are presented in Section 4. Their discussion and a comparison of the found techniques can be found in Section 5. Section 6 concludes this paper.

2. BACKGROUND & RELATED WORK

Robustness is the measurement to describe how well a system behaves under non-standard conditions [32]. The IEEE standard [21] defines robustness as

> the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

An alternative taxonomy defines it as a specialized attribute of dependability [2]. Dependability is the degree to which one can trust the promised services of a system. Robustness describes in this context a system reaction to a certain class of errors.

The goal of robustness testing is to evaluate how robust the system delivers its service. This is achieved by selecting a faultload which contains exceptional input or stressful conditions [32] leading to an error. Since it does not require any knowledge about the internal working, robustness testing is a black-box testing variant.

The failures unveiled are normally classified according to the five point scale CRASH [28] which was established in the course of the Ballista project: *Catastrophic* (whole system crashes), *Restart* (process hangs, requiring restart), *Abort* (process terminates abnormally, e.g. segmentation fault), *Silent* (no error code although one occurred), and *Hindering* (incorrect error code returned). These failure types can then be used to prioritize more critical errors and allow the categorization of errors found by automated testing methods.

During our research, we found two reviews on software testability which contained papers on robustness testing. Hassan et al. [17] mainly reviewed papers on the observability and controllability of testing. While these topics deal with the requirements of robustness testing, they do not fo-

Initial SLR	Type	Found	Filtered
[17]	Forward	34	9
	Backward	214	23
[38]	Forward	7	3
	Backward	35	5
Total		**290**	**40**

Table 1: Contribution found based on initial SLRs

Search Engine	Found	Limited	Filtered
Springer	2824	556	20
IEEE	215	155	12
ACM	208178	414	18
ScienceDirect	116137	600	13
Wiley Online	31059	600	7
Total	**358413**	**2325**	**70**

Table 2: Search engine search results (including duplicates)

Search Engine	Found	Limited	Filtered
testing technique *OR* practices	57480	500	14
fuzz	657	270	14
combinatorial	7248	329	10
boundary value	112298	406	12
equivalence class	65556	404	8
state transition	115174	416	12
Total	**358413**	**2325**	**70**

Table 3: Search engine results by search term (including duplicates). All search terms also included "software robustness".

cus solely on finding robustness testing methods. Shahrokni et al. [38] categorized their results into categories including requirements, analysis, design & architecture and verification & validation. They found and categorized a total of 144 papers. Many of the papers are on fault prevention, fault measurement and other analysis of software. Of these categories, the subcategories fault injection and automated robustness testing contain papers on robustness testing methods. Since this review is from 2012, many new contributions are not included. Both reviews do not compare the testing techniques based on parameters describing their usability. As a result, both reviews do not cover the robustness testing techniques in-depth. We include mainly new papers and focus only on blackbox testing methods, to provide an overview over the currently available technologies and techniques used in robustness testing. Furthermore, we decided on research questions aimed at finding the best technique depending on specific requirements, to help to understand which techniques are most suitable for which applications.

3. METHODOLOGY

We decided to perform a literature search with a structure similar to other reviews [17, 38] in order to present our results as structured as possible. Thus, we first decided on critical research questions, then on sources and finally on specific search terms.

3.1 Research Questions

Our research goal is to create an overview over robustness testing techniques and to classify the techniques. Furthermore, we want to find use cases and examples for different techniques, to give a general idea of when a technique could be used. We concretize this in three research questions:

RQ1 Techniques: Which concrete techniques are available for robustness testing?

RQ2 Applicability: In which context are the techniques applicable?

RQ3 Effectivity: How effective are the techniques in finding robustness issues?

Thereby, we want to achieve a broad overview of all available techniques to aid in finding a suitable robustness testing technique depending on requirements and software environment.

3.2 Sources & Search Criteria

To find the most relevant papers we conducted a forward/backward search on the initial starting papers [17, 38]. These papers were chosen because of their general topic and a high likelihood to be cited in papers on robustness testing techniques of all types. A forward search is conducted by looking up all papers citing the starting paper using a search engine and a backward search is done by looking up the sources cited in a paper. Our search consisted of a forward search as the first step, and a backward search starting from the results of the first stage as the second step. This gave us a higher number of more recent results than two forward searches, because the results after the first step were mostly too recent, to have been cited in other work. The amount of results of this search is summarized in Table 1.

Additionally, we conducted a keyword search with five search engines (Springer, IEEE, ACM, ScienceDirect Wiley Online). We selected these search engines based on their popularity, number of results and accuracy of the results. We only allowed results newer than 2012 in this search to focus on recent advancements that are not yet summarized by the previously mentioned reviews [17, 38]. When a search resulted in too many results, the results were limited to 100 and sorted by the 'most relevant' option to get the results containing the most keywords. We always marked all keywords as necessarily contained in the result and used an appropriate function to substitute the OR statement. Further, we narrowed down the results by selecting computer science and matching areas where applicable. As keywords we used "software robustness testing techniques OR practices" as a general search term and "software robustness" combined with *fuzz*, *combinatorial*, *boundary value*, *equivalence class* and *state transition* as terms based on well-known black-box testing techniques.

Table 4 provides an overview over the amount of papers we found in each area. In this table and in Table 1 we use "Found" to refer to the amount of results the search had and "Limited" to refer to the amount of results after limiting the results in every search engine to a maximum of 100. Then we filtered all results first based on title and in unclear cases the abstract. All papers that were relevant based on this technique were collected in a table with the search term, title, link and notes. Table 3 shows the amount of results we found for each search term. Finally, we removed all duplicates, which were found using different search engines or different terms.

This resulted in a total of 110 filtered papers. In the next step, both authors read the abstract of the papers and classified them. The papers were marked as "relevant", "not relevant", and "unclear" depending on whether they investigate a robustness testing technique. We then discarded the papers, which got marked as not relevant by both and kept the papers we both marked as relevant. The papers one of us marked as unclear were kept or removed by looking into and discussing the contents. In total we had 28 relevant papers after this step. This includes reviews, chapters in books and general papers. All papers were then sorted into categories based on their content. This was necessary since the categories often only reflected the search term and not the actual topic. In the next step, we sorted the papers by focus: whether they contain new and/or improved solutions to problems and applications or cover testing methods applied to specific applications. Table 4 shows the final number of results for each category divided into techniques and applications.

3.3 Threats to Validity

We addressed many concerns of the threats to validity by using the literature review by Hassan et al. [17] and Shahrokni et al. [38] as guidelines. We thereby also followed the recommendations for structured literature reviews [25].

We searched multiple databases to gain a publisher independent overview. To consider all contributions in the field, we used multiple search queries. A general key word search was used to find all latest techniques researches without association to known approaches. However, to ensure that improvements of concrete techniques were included, we also searched the combination with well known techniques.

To conduct the literature review within an respectable time period we limited our results. We admit that this decision carries the risk of missing important contributions. We argue that we only limited results when the number of findings exceeded 100. In addition, particularly the search engines with constantly high results such as ACM and ScienceDirect (cf. Table 2) are affected. Based on sampling we can assume that we also received partial hits as results, i.e., papers matching only "robustness" or "software". We therefore rely on the relevance ordering provided by the search engine to consider primarily those contributions corresponding to the largest possible usage of entered key words. Furthermore, we conducted a forward and backward search based on the initial literature reviews to increase the chance of finding relevant results. Finally, we note that the sighting of more than 350000 results without any limitation is utopistic.

We adopted a multi-stage filtering process to select the relevant contributions from the results. In all stages (the title-based filtering step, the abstract-based filtering, and detailed study for categorization) checks were performed. Every author conducted the task independently from the other and compared their results in the aftermath. In the case of deviation, the selected case was jointly studied and decided. We therefore think to have limited the risk of individual bias to influence our results to a minimum.

4. RESULTS AND ANALYSIS

Our reviews covers 2325 results from five search engines as well as 290 based on our original starting literature reviews (see Section 2) and yielded 28 relevant papers. After studying them in detail, we were left with four concrete techniques which were covered: *fuzz testing*, *combinatorial testing*, *boundary value testing*, and *state transition testing.* For 10 contributions we were not able to decide on a technique since they cover multiple and/or investigated robustness in a more general sense. Furthermore, we decided for each contribution whether the main focus is the extension of the body of knowledge or the investigation of its application. A full breakdown can be found in Table 4.

Topic	Technique	Application
Fuzz	5	4
Boundary Values	1	1
Combinatorial	1	1
State Transition	5	1
Miscellaneous	2	6
Total	**15**	**13**

Table 4: Unique results aggregated by topic and focus

A slightly different classification of robustness testing techniques was introduced by Micskei et al. [32]. They categorized techniques based on the main milestones of the evolution of testing technique beginning from the early 1990s to the early 2010s. They root the origins of robustness testing to *physical fault injection.* To verify especially safety-critical systems, fault injection can be used to simulate faults created by interacting components or underlying layers. However, according to Micskei et al., *randomized input* can be considered as the first actual robustness testing technique in the mid 1990s. In the late 1990s, the idea of *invalid input* became popular, where values outside of the allowed input domain were tested. Those independent techniques were then combined in *type specific testing.* This involves the combination of valid and invalid for different parameters. With the rise of object oriented programming, this approach was extended to *testing object oriented systems.* Independent of this, robustness testing and *code mutation* were investigated in the early 2000s. Based on valid code, invalid and thereby robustness testing sequences can be generated by, e.g., omitting calls or replacing parameter values. The latest trend by 2012 is noted to be *model-based robustness testing* with the rise of model-driven software development.

On first sight, our results do not fit this taxonomy. However, during research, we noticed that the sharp separation by technique evolution is not reflected in all publications. All found papers on the topic of *randomized input* referred to it as *fuzz testing.* Although defined slightly different by Micskei et al. [32], we consider it as synonymous. We noticed that for the techniques *invalid input*, *type specific testing* and *testing object oriented systems*, there is no possibility to distinguish nowadays. Object oriented development is de facto standard, and testing for invalid values based on the type is common practise [23]. Common for all techniques, however, is the goal to find values which violate the input constraint, i.e., by being outside a "boundary". Furthermore, valid and invalid values are tested in combination. Therefore, we consider *boundary value testing* and *combinatorial testing* as a mix of this taxonomy. This perspective is supported by our findings (cf. Section 4.2 and Section 4.3). The same applies to the two robustness trends *code mutation* and *model-based* robustness testing. Micskei et al. [32] define the first as technique for state-based systems and separate it from models with formal or semi-formal specifications. In our findings, both are mixed for *state transition testing*. All contributions dealing with mutations of valid code consider their research area as model-based. Additionally, state machine based testing is commonly considered as model-based [40] (cf. Section 4.4).

Although we included equivalence class testing in our search keyword, we were not able to determine a contribution which covered this testing technique explicitly for robustness testing. We root this mainly to the nature of equivalence testing, its overlapping with boundary value testing and the fact that only two papers covered the latter at all. Notably, *fuzz testing* (9 contributions) and *state transition testing* (6 contributions) receive the most attention by the research community since 2012. While for fuzz testing, half focuses on the application, the overwhelming majority for state transition testing is investigating the improvement of the technique in general.

In the following, we give a short introduction for each technique and present the insights from the respective contributions as shown in Table 4.

4.1 Fuzz Testing

Fuzzy robustness testing techniques randomly generate input data to test the target software. Many of the following techniques improve the way the data is generated or how it is inserted into the target software. Despite seeming like a bad alternative to other techniques due to the randomness, it is a popular technique and has a high number of errors found compared to other techniques [26]. The target of many techniques is to cover as much of the functionality of the software as possible. Fuzzing can be especially useful, because no model specific to the application has to be created. Challenges include the grouping of errors without creating duplicates or omitting errors [26].

Figure 1 shows the typical workflow of fuzz testing techniques. First test data is generated using a generator that uses a fuzzy algorithm. Then the generated inputs are applied to the target system, which has outputs in the form of a returned value or a crash. This output is then analyzed to determine if it is expected or an error. After this, some implementations use the gathered data to generate new test cases and repeat the process.

This is used, e.g., in the JCrasher robustness testing tool [10], which automatically searches bugs in Java applications. It tries to raise runtime exceptions, by calling functions with random values after examining their input range. It also utilizes heuristics, to determine if an exception is a bug or occurred because of invalid preconditions created by JCrasher. It takes advantage of the public methods in Java to use them as an interface to the software.

A different approach to generate test data is to start with known valid inputs and use a mutation modifier every time a test is executed [35]. The mutation modifier replaces a valid input with one just outside the valid bounds. By controlling the probability of this modifier it allows to control the degree of the atypical input.

A technique that does not directly use random inputs to generate the input data is described by Kargen et al. [24]. Instead of randomly generating the direct inputs for a software it uses programs that generate valid inputs. These generation programs are then modified with random mutations to create new slightly different generators for input data. The researchers implemented the technique in their tool called MutaGen and discovered 8 bugs in Linux tools using it. They found increases of one order of magnitude in code coverage. There are also examples of fuzzing software for mobile devices like Chizpurfle [20] for Android. The researchers developed Chizpurfle on the Samsung Galaxy S6 Edge with Android 7 and used it to find bugs in the customizations made by the manufacturer.

One of the advantages of fuzzing is the low workload compared to other techniques. The fuzzer SulleyEX [43] expands the open source fuzzer Sulley. It uses a state machine, to describe the states of a network protocol and includes a custom generation algorithm. It aims at increasing automation and reducing the workload. There are also fuzzy test methods combined with combinatorial testing to improve coverage and reduce the amount of shallow paths followed like SimFuzz [42]. SimFuzz uses a traditional Blackbox fuzzing algorithm for the initial data acquisition, which can test for deep semantics. Then it uses combinatorial generation on these test cases, to increase the amount of inputs exploring deep program paths.

There is also an approach [9] that uses recorded human interactions with the software. It records human input streams and then modifies the recordings and plays them back to cause errors. This can include complex input chains as opposed to most other fuzzing techniques. It is also possible to generate test cases at runtime [37] instead of before the execution, to improve the efficiency. This method can use previous results to improve the newly generated test cases easily, because the test cases are generated at runtime. Fuzzing can also be used to test deep learning systems. The fuzzer DLFuzz [16] changes the input to maximize the prediction difference. This is an important use case as deep learning rises in popularity and testing their reliability is hard.

A test of the trustworthiness of the results of fuzzing techniques was also performed [26]. It highlights the effects of improper evaluation of testing methods.

4.2 Boundary Value Testing

When applying boundary value testing, the functionality of a program or function is tested based on its input domain [23]. Test cases are generated by selecting specific values from the domain of each input variable and combining them. The selection and combination depends on the aimed extent of testing.

For the normal boundary value testing, extreme values and nominal are chosen, all inside the valid domain. Test cases are then generated by combining the non-nominal values of one variable with the nominal values of the others. To increase the depth of testing, every value of each variable can be combined with each of other variables. This approach is considered as worst-case boundary value testing and generates an exponential number of test cases.

However, for evaluating robustness, invalid values are of interest. Their usage is considered as robust boundary value testing. This involves the selection of values very close to the boundary of the input domain. Combining them with the valid values of other variables yields a set of invalid test cases. Thereby, a correct behavior to invalid input can be tested.

Our literature review with focus on the years after 2012

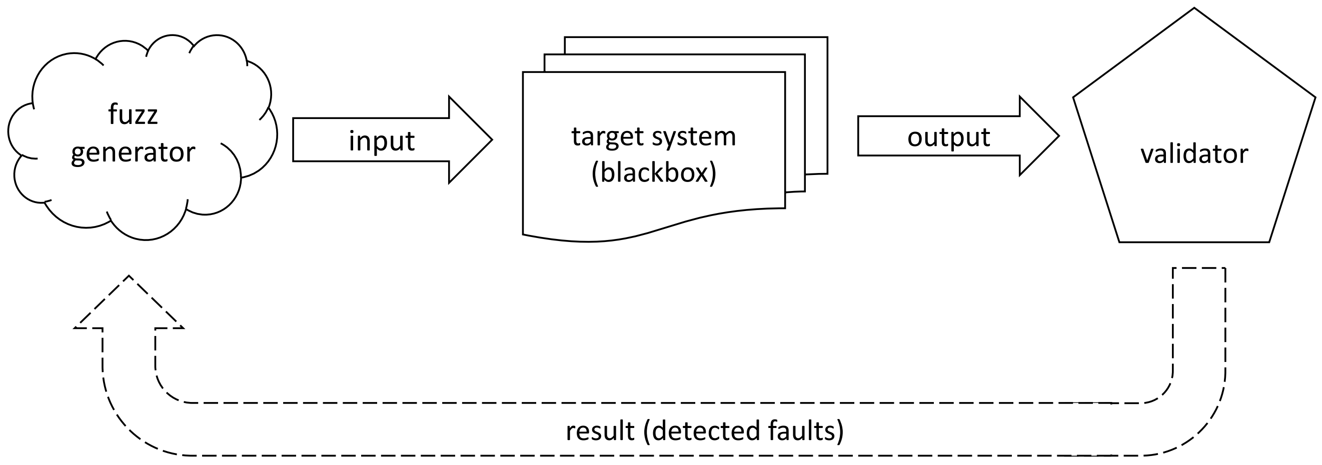


Figure 1: The typical workflow of fuzz testing techniques

yields only two contributions dealing with this topic. Furthermore, only one of them aims at broadening the body of knowledge in general while the other focus on the application of robust boundary value testing.

Crucial for the generation of test values is a comprehensive knowledge about the boundaries of the input domain. A very recent publication describes a method to determine the boundary values automatically [31]. For this purpose, test cases over a wide range of valid inputs are generated. These test cases are mutated to find invalid test cases close to the valid ones. The presented method is evaluated by applying it to three modules from the standard library of the Julia programming language.

In [11], a generic tool for the generation of test cases for boundary value testing including robustness testing is presented. However, this tool does not focus explicitly on robustness testing nor can it be considered as more than a small helper tool.

4.3 Combinatorial Testing

Combinatorial testing is a testing technique that finds combinations of inputs to achieve a broad coverage with low computational requirements [5]. In general, systems have many inputs and it is impossible to test all combinations of these inputs. To reduce the number of combinations there are multiple algorithms to generate covering arrays in combinatorial testing, which is the main challenge [29]. Covering arrays contain tests with all t-way combinations of input parameters. Higher values of t indicate a better coverage. Because multiple combinations containing each parameter are tested it can find failures, which are triggered by interactions between parameters [33].

This can be done by first partitioning the values of input parameters. These partitions could, e.g., be “full-age” and “under-age”, in the case of a parameter describing the age of a person. This drastically reduces the amount of input that needs to be tested. These partitions can then be automatically combined with different strategies, to cover as many cases as possible. While every partition should occur at least once, mostly more combinations for each partition are used. Combinatorial testing aims to select a small number of test cases that provide a high coverage. In a survey of combinatorial testing [33] different strategies to select these test suites are discussed. This includes variable strength interaction testing, which tries to cover mainly combinations with many mutual interactions.

This is applied to graphical and command line user interfaces by a study testing combinatorial testing on a widely used tool for combinatorial test generation [4]. The researchers use t-way combinatorial testing, which tests every combination of values of t parameters at least once. The papers shows that combinatorial testing can detect a high number of faults and has a good code coverage although setting up the model for input parameters may require significant work.

Because invalid inputs can mask test cases by leading into the same abort condition every time, there is a test case generation method that only allows one invalid input in every test case [15]. This technique can be used to enhance existing algorithms to include invalid value combinations. It uses soft-constraints to create test cases for over-constrained models.

4.4 State Transition Testing

State transition testing is a model-based testing technique. For model-based testing, a model of the system under test (SUT) has to be established which enables the derivation of test cases from it [34].

Models can be of formal or semiformal nature. Through their precise nature, the first allows the automatic generation of test cases, while later requires further human reasoning in order to receive tests [23]. But independent of their notation, models have to describe the behavior of the SUT with the right level of detail to be useful and abstract enough to capture the essence of it [40]. Therefore, a wide range of possibilities exists. Depending on the required expressiveness, the SUT can be modelled, among others, through Petri nets, finite state machines and state transition systems such as UML transition based systems [23, 40].

Finite state machines and state transition systems consist

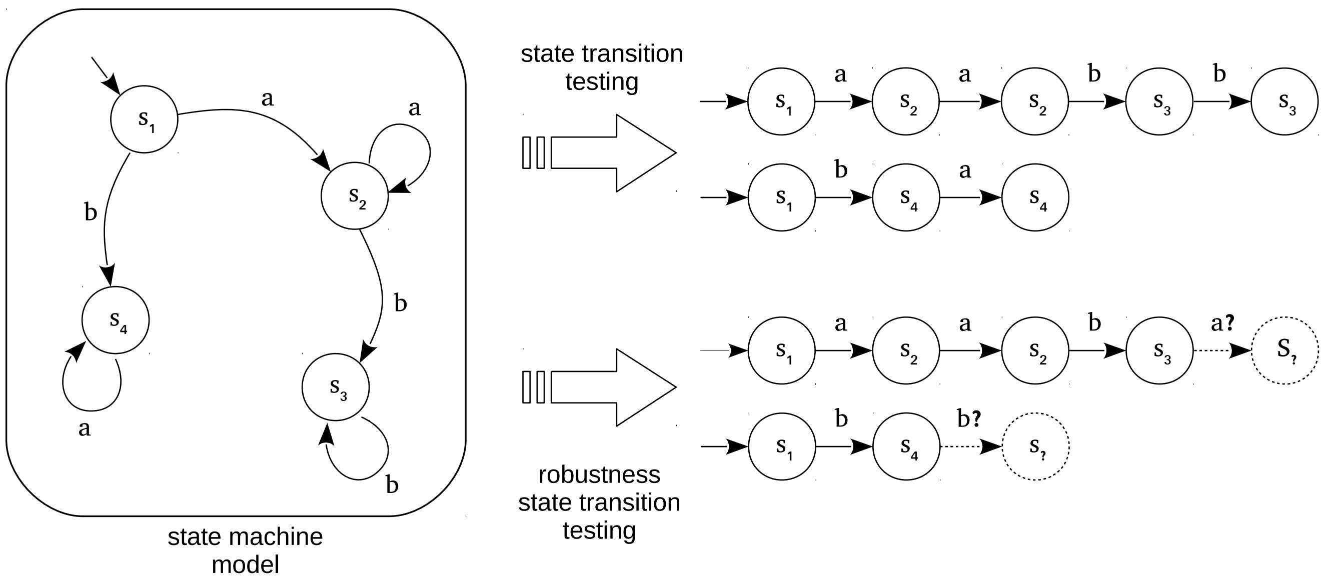


Figure 2: Fundamental model-based approach of (robustness) state transition testing

of nodes connected with arcs. Each node corresponds to a state of the SUT, while an arc describes an action possible from this state [40]. These actions are also called transitions. Depending on the exact notation, they can further be enhanced with guards. This way it is possible to describe conditions which have to be satisfied to take a transition.

In order to derive test cases, approaches with different test coverage criteria are available [40]. One can require, e.g., that the resulting test cases as whole have to interact with all states, all transitions, or also all inputs and/or outputs. In Figure 2, this workflow is depicted with test cases satisfying the transition coverage criteria.

Our literature search yields the second highest amount of contributions for the technique of state transition testing. Five of them deal with the technique as such, while only one reports its application. This coincides with the observation of the more increasing interest in model-based testing caused by the popularity of model-driven development [32].

In this context it is interesting to note that the majority of the contributions do not describe the relation between the model and implementation. In model-driven engineering, models are raised to main development artifacts [39], enabling an automated code and test generation at the same time. Instead of creating and using a model solely as intermediate artifact for testing, its usefulness is thereby increased. This aspect is only mentioned by two contributions [18, 36] and here also used during evaluation. The focus of other contributions lies mainly on the enhancement of a own tool suite [1, 30]. One contribution describes its findings as being of theoretical nature alone [14].

A framework for testing stateful components is presented by Lei et al. [30]. To receive test cases, first paths covering all transitions are generated. The actual cases are devised through divergence from path (calling methods with unspecified transition) or invalid method calls (arguments violating pre-conditions). Mutation through path divergence is depicted in Figure 2. The framework is evaluated with a toy example and the results were found to be superior to fuzz testing.

A very similar system is based on the Z3 solver [18]. Existing coverage criteria for robustness testing (all-transition coverage, pairwise transition coverage, predicate coverage, combinatorial coverage) are extended. The solver is used to determine the testing paths. Furthermore, attempts are made to speed up the execution by caching. The evaluation is performed with a generic HashTable implementation and a toy traffic light controller.

Comparable is the investigation of an aspect oriented approach [1]. Following the authors, robustness modeling in complex industrial systems without aspects is not scalable. Therefore, state machines are enriched with aspects through the UML profile AspectSM. However, robustness is understood as the ability of the system to recover from faults whose sources are primarily the environment and not invalid input. The work is evaluated using a simplified industry case study.

Savary et al. [36] present a fully model-based approach to robustness testing on the basis of state machines described in Event-B. Based on the valid input specifications of system, test cases are generated through negation and mutation. This devised approach is then applied to a Java bytecode verifier.

A quite different usage of models is proposed by Chen et al. [8]. Instead of requiring the manual construction of one describing the intended behavior of the SUT, existing method sequences are transformed to extended finite state machines. For testing case generation, existing paths are manipulated. This is done with two algorithms: Operations Conflict Sequences Generation and Conditions Conflict Sequences Generation. In a case study, this is applied to COM components and shown to have better effective rates compared to fuzz testing.

A theoretical framework model based robustness testing is described by Fernandez et al. [14]. Therefore, no precise method of generating test paths are presented.

The only practical contribution investigates the application of state transition testing in the automotive industry [6]. Based on the automotive realtime operation system OSEK, formal test models for parallel state machines execution are devised. Based on constraints of these state machines, illegal paths are found using a model checker. This technique is compared to specification based and concolic testing and found to be superior in an evaluation.

4.5 Miscellaneous

Our search resulted in a number of contributions which we were not able to categorize. This was either due to the non-comparability to other approaches, a mixture of different techniques or a meta-discussion of robustness. Independently, we consider majority of contributions in this category state their field of research as being model-based.

El-Attar and Abdul-Ghani [13] developed a general approach for improving security robustness. Instead of developing a concrete technique for testing application on the code level, they present the application of misuse case models. In addition to normal usage scenarios, the goals of a (malicious) misuser are taken into account. Thereby, effective robustness acceptance tests can be devised in a model-based environment. Its application is highlighted in a case study of a faculty search committee.

An approach to fully automate test case generation with the help of Markov chains can be considered as model-based engineering [3]. With their main focus on the automotive industry, they cover embedded systems with the challenge of time sensitivity. For fully automated testing, they devise a system which first predicts the results of input, then executes the test and decides whether to stop testing. It was applied in two case studies showing an increased bug detection as well as a reduced time consumption for testing.

There were five papers describing applications of robustness testing that did not fit in our categories. These mostly use a combination and/or modified versions of the other techniques.

Vernotte et al. [41] describe a testing process to find vulnerabilities in web applications. The process is called pattern-driven and model-based vulnerability testing. It uses a model to describe the tested application and applies vulnerability test patterns to this model in order to find vulnerabilities. This is especially useful for vulnerabilities like SQL injections and Cross-Site scripting that always use similar patterns. To accomplish this, they use a combination of a model and fuzzing. It was tested on a complex and freely accessible eHealth system.

A different paper explores the application of Automated Stress Testing for Autonomy Architectures (ASTAA) [19]. It is an example from the industry combining different techniques. This framework aims to automate the testing process by using a three phase system. This project expands the Ballista tool [27] to cover robotics and autonomy software.

Carozza et al. describe a system to test web services of an air traffic control application [7]. They create test cases based on the specification of the API and a predefined set of rules, which are applied to the specification. They developed a tool called WSRtesting, which helps to manage complex parameters and can automatically test XML formatted parameters. To test for errors, it first runs without invalid parameters, to capture the expected response. Then several tests containing invalid parameters are run. It also allows the user to analyze the results using the web interface of WSRtesting.

The SMArDT modeling system introduced by Drave et al. [12] is a testing solution for automotive software. It does not focus on robustness testing, but contains robustness as a metric. There also is a thesis describing robustness testing of AUTOSAR software components [22]. It uses the interfaces defined by the AUTOSAR standard to perform robustness testing based on the input datatypes. Different fault generation techniques including fuzz testing are supported by changing a setting.

5. DISCUSSION

This structured literature review gives insights in the research of robustness testing techniques of software systems since 2012. We were able to identify the four concrete techniques *fuzz testing*, *combinatorial testing*, *boundary value testing*, and *state transition testing*. Fuzz and state transition testing received the most attention since 2012 in the research community. However, we also found contributions which could not be assigned to a specific technique. Notably, the majority of those had an application focus.

Fuzz testing is a popular robustness testing technique which uses random input values to cause faults. We found many papers describing new generation strategies as well as applications for specific use cases [20, 10], which indicates a high interest in this technique. In general, random values are generated and inserted into user interfaces and directly into software components. Fault detection can be a simple crash/freezing detection of the target system. In many cases it is the easiest technique to deploy (high applicability) and requires the least amount of work, because no extensive model or documentation of the software is required. The effectivity expressed by the fault detection rate is not as good as state transition testing combined with an accurate model. Nevertheless, the number of bugs detected for a given amount of work is high, because very little setup is needed for most fuzz based techniques.

Boundary value testing is a well known technique from regular functional testing. Based on the division of input space, values are used for testing. For evaluating robustness, invalid values close to the border of permitted input space are chosen. This approach is commonly known and referenced in literature in all contexts [23, 34]. Therefore, it is not surprising to see only a few contributions in this area. The only active research contribution investigated the automatic determination of valid input space boundaries [11]. While being useful in theory, it requires an already correct functioning system. This makes of the idea of robustness testing a mockery and the idea not applicable in general.

Combinatorial testing is another technique for which we found only two results. In combinatorial testing, first partitions of input values are found and then combinations of these input parameters are tested. Therefore a model of the inputs is required for combinatorial testing. The selection of the combinations can be done using different strategies. Creating combinations with a maximum of one invalid value [15] can be beneficial to prevent masking by the first error. This technique requires the creation of a model for the inputs. Its applicability is between fuzz testing and state transition testing, because the model does not have to cover states.

The second active investigated technique by number of contributions is state transition testing. Since it is a model-based technique, an increased research activity caused by the popularity of model-based engineering seems consequential. This corresponds to only one paper focusing on its application. For state transition testing, the behavior of the tested systems is captured in models, mainly state transition systems. In these systems, nodes represent states and transitions method calls. Test cases are generated based on paths on this model. For robustness, diverging paths are tested and/or method parameters mutated. Thereby, stateful systems can be structurally evaluated. This makes state transition testing useful in the majority of software systems. The evaluation of toy examples and case studies indicates that in those systems, state transition testing results in a higher fault detection rate than fuzz testing. So far, only the application in automobile industry was investigated.

The miscellaneous papers investigating techniques, which we could not directly assign to a category, are also rooted in model-based engineering. Special model-based notations are devised to ensure security robustness. To increase test automation, a self-learning system with focus for the automobile industry was created. Only in the later cases, an evaluation with comparison was conducted to highlight the advantages.

These miscellaneous papers are often specific solutions for applications and environments that use multiple or modified versions of our categories. A main goal often is to automate and optimize the workflow to minimize additional work required for robustness testing. This is done by using existing specifications to generate test cases [22] or creating applications for other steps of robustness testing. These solutions are mostly targeted at a specific system, which reduces the applicability through optimization for a special case, but increases the effectivity. This specialization means that less work is required to set up the test tools.

While we did not find papers specifically exploring equivalence class based robustness testing, equivalence classes are an important tool, which is often used in the preparation of other techniques to represent partitions of input groups. This shows that the interest in using this technique has dropped, but the idea of equivalence classes in robustness testing is used by other techniques.

We saw that it is hard to directly compare the techniques, because there is a lack of comparisons using the same target. However, we found that fuzz based approaches are generally the least work and therefore have the best applicability, because no models are required. State transition testing on the other hand provides a better coverage and fault detection rate and therefore effectivity. Combinatorial testing and boundary value testing are regarding effectivity and applicability in between both of these. These results are summarized in Table 5. Additionally, it shows the types of applications for which the testing methods generally can be used. There are exceptions to this, but it is a good general guideline. While some papers remain not categorized, these papers use multiple techniques or focus on different areas allowing no general insights.

Technique	Required Effort	Effectivity	Application Context
Fuzz	+	o/+	Simple In-/Output
Boundary Values	o	o	Numeric Input
Combinatorial	o	o	Mutually dependent parameters
State Transition	-/o	+	Stateful systems

Table 5: Overview over the techniques we found and their properties regarding our research questions. "+" is better than other results, "o" normal results and "-" worse than average results.

6. CONCLUSIONS

We performed a systematic literature review on robustness testing techniques. For this we started at two reviews with a forward/backward search and searched five big search engines for terms we identified after an initial analysis. We identified four main techniques. These were then evaluated based on our research questions. The evaluated testing techniques have different advantages and disadvantages. Fuzzing is the least amount of work to setup and run, but does not always cover every combination. Therefore it has a high applicability, but low effectivity. Combinatorial and state based testing can be more complicated to setup, but have a better test coverage. A result of the increased coverage is a higher effectivity at the cost of more work. Combinatorial testing can be used when a model for the input parameters exists and can be useful to achieve a high coverage without a very complicated model. Boundary value testing is another alternative, which can be especially useful, if edge cases are of high interest. In general, fuzz testing is the least manual work and state transition testing achieves the highest coverage.

7. REFERENCES

[1] S. Ali, L. C. Briand, and H. Hemmati. Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. *Software & Systems Modeling*, 11(4):633–670, 10 2012.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing Algirdas. *Institute for Systems Research*, 0114:10, 2001.

[3] R. Awedikian and B. Yannou. A practical model-based statistical approach for generating functional test cases: application in the automotive industry. *Software Testing, Verification and Reliability*, 24(2):85–123, 3 2014.

[4] M. N. Borazjany, L. Yu, Y. Lei, R. Kacker, and R. Kuhn. Combinatorial Testing of ACTS: A Case Study. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 591–600. IEEE, 4 2012.

[5] R. C. Bryce, Y. Lei, D. R. Kuhn, and R. Kacker. Combinatorial Testing. In *Handbook of Research on Software Engineering and Productivity Technologies*, pages 196–208. IGI Global, 2010.

[6] T. Byun and Y. Choi. Automated system-level safety testing using constraint patterns for automotive operating systems. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC '15*, pages 1815–1822, New York, New York, USA, 2015. ACM Press.

[7] G. Carrozza, A. Napolitano, N. Laranjeiro, and M. Viera. WSRTesting: Hands-On Solution to Improve Web Services Robustness Testing. In *2011 Fifth Latin-American Symposium on Dependable Computing Workshops*, pages 41–46. IEEE, 4 2011.

[8] J. Chen, J. Chen, R. Huang, Y. Guo, and Y. Zhan. An approach of security testing for third-party component based on state mutation. *Security and Communication Networks*, 9(15):2827–2842, 2016.

[9] J. A. Cottam, L. Blaha, D. Zarzhitsky, M. Thomas, and E. Skomski. Crossing the Streams: Fuzz testing with user input. *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, 2018-Janua:4362–4371, 2018.

[10] C. Csallner and Y. Smaragdakis. JCrasher: An automatic robustness tester for Java. *Software - Practice and Experience*, 34(11):1025–1050, 2004.

[11] N. Debnath, A. Kruger, and M. Alexander. A Boundary Value Analysis Tool - Design and Description. In *2013 10th International Conference on Information Technology: New Generations*, pages 77–82. IEEE, 4 2013.

[12] I. Drave, S. Hillemacher, T. Greifenberg, S. Kriebel, E. Kusmenko, M. Markthaler, P. Orth, K. S. Salman, J. Richenhagen, B. Rumpe, C. Schulze, M. von Wenckstern, and A. Wortmann. SMArDT modeling for automotive software testing. *Software: Practice and Experience*, (May):1–28, 10 2018.

[13] M. El-Attar and H. A. Abdul-Ghani. Using security robustness analysis for early-stage validation of functional security requirements. *Requirements Engineering*, 21(1):1–27, 2016.

[14] J.-C. Fernandez, L. Mounier, and C. Pachon. A Model-Based Approach for Robustness Testing. In *Testing of Communicating*, pages 333–348, 2005.

[15] K. Fögen and H. Lichter. Combinatorial Testing with Constraints for Negative Test Cases. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 328–331, 2018.

[16] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun. DLFuzz: differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, pages 739–743, New York, New York, USA, 8 2018. ACM Press.

[17] M. M. Hassan, W. Afzal, M. Blom, B. Lindstrom, S. F. Andler, and S. Eldh. Testability and Software Robustness: A Systematic Literature Review. In *Proceedings - 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015*, 2015.

[18] P. Heckeler, H. Eichelberger, T. Kropf, J. Ruf, S. Huster, S. Burg, W. Rosenstiel, and B. Schlich. Accelerated model-based robustness testing of state machine implementations. *ACM SIGAPP Applied Computing Review*, 13(3):50–67, 2013.

[19] C. Hutchison, M. Zizyte, P. E. Lanigan, D. Guttendorf, M. Wagner, C. L. Goues, and P. Koopman. Robustness testing of autonomy software. *Proceedings of the 40th International Conference on Software Engineering Software Engineering in Practice - ICSE-SEIP '18*, pages 276–285, 2018.

[20] A. K. Iannillo, R. Natella, D. Cotroneo, and C. Nita-Rotaru. Chizpurfle: A Gray-Box Android Fuzzer for Vendor Service Customizations. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, 2017-Octob:1–11, 2017.

[21] IEEE (Institute of Electrical and Electronics Engineers). *ISO/IEC/IEEE 24765:2017(E) - Systems and software engineering: Vocabulary.* 2017.

[22] V. Jansson and J. Lindahl. *Robustness Testing of AUTOSAR Software Components (Master Thesis).* 2013.

[23] P. C. Jorgensen. *Software Testing - A Craftsman's Approach.* CRC Press, Inc., 4th edition, 2015.

[24] U. Kargén and N. Shahmehri. Turning programs against each other: high coverage fuzz-testing using binary-code mutation and dynamic slicing. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, pages 782–792, 2015.

[25] B. Kitchenham and S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01, Keele University and Durham University, 2007.

[26] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks. Evaluating Fuzz Testing. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*, pages 2123–2138, New York, New York, USA, 2018. ACM Press.

[27] P. Koopman, K. Devale, and J. Devale. Interface Robustness Testing: Experience and Lessons Learned from the Ballista Project. *Dependability Benchmarking for Computer Systems*, pages 201–226, 2008.

[28] P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz. Comparing operating systems using robustness benchmarks. *Reliable Distributed Systems, 1997. Proceedings., The Sixteenth Symposium on*, page 72–79, 1997.

[29] R. Kuhn, Y. Lei, and R. Kacker. Practical combinatorial testing: Beyond pairwise. *IT Professional*, 10(3):19–23, 2008.

[30] B. Lei, Z. Liu, C. Morisset, and X. Li. State Based Robustness Testing for Components. *Electronic Notes in Theoretical Computer Science*, 260:173–188, 1 2010.

[31] B. Marculescu and R. Feldt. Finding a boundary between valid and invalid regions of the input space. *arXiv e-prints*, pages 1–10, 10 2018.

[32] Z. Micskei, H. Madeira, A. Avritzer, I. Majzik, M. Vieira, and N. Antunes. Robustness Testing Techniques and Tools. In K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel, editors, *Resilience Assessment and Evaluation of Computing Systems*, pages 323–339. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[33] C. Nie and H. Leung. A survey of combinatorial testing. *ACM Computing Surveys*, 43(2):1–29, 2011.

[34] M. Pezze and M. Young. *Software Testing and Analysis: Process, Principles and Techniques.* John Wiley & Sons, Inc., 2008.

[35] S. Poulding and R. Feldt. Generating Controllably Invalid and Atypical Inputs for Robustness Testing. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 81–84. IEEE, 3 2017.

[36] A. Savary, M. Frappier, M. Leuschel, and J.-L. Lanet. Model-Based Robustness Testing in Event-B Using Mutation. *Communications of the ACM*, 51(9):54, 2008.

[37] M. Schneider, J. Grossmann, I. Schieferdecker, and A. Pietschker. Online Model-Based Behavioral Fuzzing. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 469–475. IEEE, 3 2013.

[38] A. Shahrokni and R. Feldt. A systematic review of software robustness. *Information and Software Technology*, 55(1):1–17, 1 2013.

[39] T. Stahl and M. Völter. *Model-Driven Software*. John Wiley & Sons, Inc., 2006.

[40] M. Utting and B. Legeard. *Practical Model-Based Testing - A Tools Approach*. Elsevier Inc., 2007.

[41] A. Vernotte, C. Botea, B. Legeard, A. Molnar, and F. Peureux. Risk-Driven Vulnerability Testing: Results from eHealth Experiments Using Patterns and Model-Based Approach. In *Risk Assessment and Risk-Driven Testing: Third International Workshop, RISK*, pages 93–109. Springer International Publishing, 2015.

[42] D. Zhang, D. Liu, Y. Lei, D. Kung, C. Csallner, N. Nystrom, and W. Wang. SimFuzz: Test case similarity directed deep fuzzing. *The Journal of Systems and Software*, 85:102–111, 2012.

[43] W. Zhou and Y. Xiang. SulleyEX: A Fuzzer for Stateful Network Protocol. *Journal of Network and Computer Applications*, 32(2):345–346, 2009.

Towards a Definition of Enterprise Architecture Debt

A Systematic Literature Review

Hendrik Höfert
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
hendrik.hoefert@rwth-aachen.de

Johannes Salentin
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
johannes.salentin@rwth-aachen.de

ABSTRACT

While Technical Debt is a widely known metaphor introduced by Ward Cunningham describing a mostly invisible result of past decisions about software that negatively affect its future, the idea of such debt is not applied to overall Enterprise Architecture. However, Enterprise Architecture and Enterprise Architecture Management in particular are established to support the decision-making process. Thus, we identified the possibility to expand the idea of the debt-concept to Enterprise Architecture in order to improve its quality. Several approaches of using EA have shown to be useful and facilitate a sustainable development of the enterprise. Especially a tool for making undesired aspects visible and drawing attention to them, as well as serving as a basis for communication is needed to improve an EA and its operating capabilities.

First, we conduct a systematic literature review (SLR) to get to know the metaphor of Technical Debt and important aspects to assess the quality of Enterprise Architectures. On the basis of the found work we define a new metaphor of Enterprise Architecture Debt, aiming at revealing weaknesses of the current state and drawing attention to opportunities of improvement. This should help to shape and undertake development of an enterprise in addition to already known frameworks. The debt concept should serve as a useful extension of those, encouraging sustainable and extensive assessment. Moreover, we propose future research topics to further enhance Enterprise Architectures and the usage of helpful tools.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management—*productivity, software quality assurance*

Keywords

Enterprise Architecture Debt, Enterprise Architecture, Technical Debt

1. INTRODUCTION

Nowadays, organizations continually get more complex, due to an ever increasing number of interwoven applications. Therefore, managing and maintaining sustainable development is difficult for stakeholders. Enterprise Architecture (EA) provides a holistic view on the enterprise and its elements that are needed to create value [26]. Stakeholders need to be aware of their whole application landscape and overall structure. In order to support this, we want to expand the known idea of Technical Debt (TD) and base our concept of Enterprise Architecture Debt (EAD) on it. Because there is no existing research on this topic yet, we want to tackle it to help improve EAs.

As Technical Debt has shown its benefits in estimating deficits of software construction, providing a tool for decision making and increasing the awareness of said deficits [15, 17, 19, 29], we want to transfer these benefits to the whole Enterprise Architecture. Metrics that are commonly used in Software Engineering (cf. TD estimation [6]) should not only be used on single systems and software, but rather be generalized to be applicable and helpful for the entire enterprise and every scenario [1, 5, 21]. This encompasses then the different layers of EA, including business and IT with its systems and strategies in particular. Thus, it gets possible to estimate the consequences of EA implementation failures so that with the new metaphor of EA Debt improvements and measurement of the quality gets possible. Furthermore, strategic decisions about future development based on urgency and dependency of debt can be facilitated.

For the purpose of introducing the metaphor, we need to find the answers to the following research questions. Here we consider our main goal in **RQ1**, with its two major subquestions **RQ1.1** and **RQ1.2** which have to be answered in order to answer the primary research question:

What is an adequate definition of Enterprise Architecture Debt? (**RQ1**)

How is Technical Debt defined and what criteria is related to it? (**RQ1.1**)

Which aspects are important for the quality of EAs and which layers are included in it? (**RQ1.2**)

Then we know how the metaphor of Technical Debt approaches to assess the relevant criteria and which aspects are crucial to EAs. Thus, we can define EA Debt aiming at including all relevant issues and becoming a powerful tool for EA management.

We structure our work as follows: After preparing our research design in section 2, we compare and point out related work in section 3. In the main part we define the new metaphor of Enterprise Architecture Debt with its important aspects (section 4) and then come up with two examples (section 5). Finally, we conclude our work in section 6 and propose future work that rests on the new metaphor of EA Debt (section 7).

2. RESEARCH DESIGN

We conduct a systematic literature review (SLR) by combining the approaches of Kitchenham [14] and Webster and Watson [33], in order to find answers to our research questions.

We searched for "technical debt definition" respectively "technical debt metaphor" in the metadata (abstract, title text and indexing terms) in order to solve our first research question (**RQ1.1**). We recognized searching simply for "technical debt" is too inaccurate for our purpose, because nowadays it is a widespread topic and numerous articles are published that do not touch our matter. We made sure that the papers actually come up with a definition or describe the main idea of Technical Debt.

For the second one (**RQ1.2**) we searched "enterprise architecture quality" in the metadata from 2010 to the present, because there are many articles addressing this topic and we want the quality issues to be up to date and most applicable to our scenario. Also, there was a tremendous increase of research on Enterprise Architecture since then. Additionally, we searched "enterprise architecture debt" just to be sure that no work exists yet, addressing our research question (**RQ1**).

Those searches were performed on both the IEEE Xplore Digital Library[1] and the ACM Digital Library[2], as we had full access to them through RWTH Aachen University. The following table 1 shows the number of papers found with the corresponding search string.

keywords	IEEE Xplore	ACM
technical debt definition	9	9
technical debt metaphor	45	33
enterprise architecture quality	515	46
enterprise architecture debt	0	0

Table 1: SLR: number of papers found

After analyzing the titles and abstracts we chose a first pool of seemingly best fitting articles [1, 2, 16, 19, 26], where many appeared in the results of the different search terms. Here our inclusion criteria are the citation count, so that the article appeared under the top ten papers, and the applicability and generalizability to our research questions based on the abstract, research questions and design and a brief look at the results. Besides we only took English articles into account that were available in the selected digital libraries. As a consequence the exclusion criteria correspond to the opposite of the inclusion criteria. Thus, we excluded, for example, articles that are not accessible for free or are written in another language than English. Regarding Technical Debt we also chose papers, that provided a better understanding of the term, like giving options to identify Technical Debt fitting to the given definition [18] or provide a look from a different perspective to the concept [30].

Searching back and forth we added frequently cited work fitting both our research questions [4, 7, 8, 13, 15, 25, 28, 34, 35]. Here, frequently means that an article belongs to the most cited throughout our already identified articles or is cited in more than one paper, that we added to our literature base so far. By sorting the search result by citation count, those articles were under the top ten and appeared useful. For that search we also used Google Scholar[3] to find the respective articles. In a last step we completed the literature base with related work known to us [24, 27, 31].

[1]`https://ieeexplore.ieee.org`
[2]`https://dl.acm.org/`

3. RELATED WORK

In this section we will point out and discuss related work. This is split up into two parts, as we have to consider Technical Debt (section 3.1) and Enterprise Architecture with its quality issues (section 3.2).

3.1 Technical Debt

We will start discussing the related work regarding *Technical Debt*: The metaphor was first introduced by Ward Cunningham in 1992 and mentions what we today would call "refactoring". He describes it as follows: "Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise." [4]. This first idea of not-quite-right code which we postpone making it right, is expanded by various people to display also other kinds of debts or ills of software development, such as test debt, people debt, architectural debt, requirement debt, documentation debt or an encompassing software debt [16, 29].

According to Kruchten et al. Technical Debt refers to invisible elements, because visible elements for improving like new features for evolution or repairing defects for maintainability issues should not be considered as debt. This is illustrated in Figure 1 where four types for improvement are shown: the tasks to attend to in the future, such as adding new features (green); investing in architecture (yellow); reducing the impact on value of defects (red); and Technical Debt (black). Whereas project backlogs often consist of the green elements, namely new features, the invisible elements should not be neglected [16, 17].

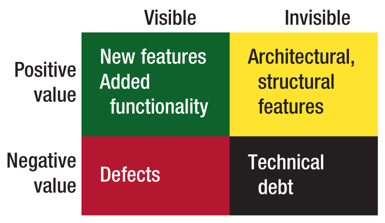

Visible
Invisible
Positive value
New features Added functionality
Architectural, structural features
Negative value
Defects
Technical debt

Figure 1: Four types of possible improvements [16]

[3]`https://scholar.google.de/`

Technical Debt should rather serve as a retrospect reflecting change of the environment, rapid success or technological advancements as a possible cause for debt. However, "the debt might actually be a good investment, but it's imperative to remain aware of this debt and the increased friction it will impose on the development team" [16]. Hence, tools are required to increase the awareness to identify debt and its causes and to manage debt-related tasks. Finally, the debt should not be treated in isolation from the visible elements of evolving and maintaining. So debt is "the invisible result of past decisions about software that negatively affect its future" [16].

Tufano et al. encountered the same phenomenon and found out that most code smells are introduced at their creation. Furthermore, the code often gets smellier due to new artifacts being build on top of suboptimal implementations. Even refactoring is often done wrong as it introduces further bad smells, which highlights the need for techniques and tools to support such processes [32].

Steve McConnell, for example, tried to categorize different types of Technical Debt. He points out that with this metaphor business and technical viewpoints can be emphasized, so that communication regarding specific problems can be enhanced. Technical Debt is used as a uniform communication tool, that allows us to measure and keep track of debt which eventually should help find a suitable solution to the upcoming challenges. In this case it should also reflect the different viewpoints, including the stakeholder's perspective, in order to allow an effective collaboration [19, 30].

Martin Fowler even came up with a categorization of Technical Debt with his "Technical Debt Quadrant" to identify different types of it [8]. The quadrant in Figure 2 shows the division into reckless/prudent and deliberate/inadvertent debt. This reflects the different scenarios where debt is taken and hints at debt being taken unconsciously or negligent sometimes.

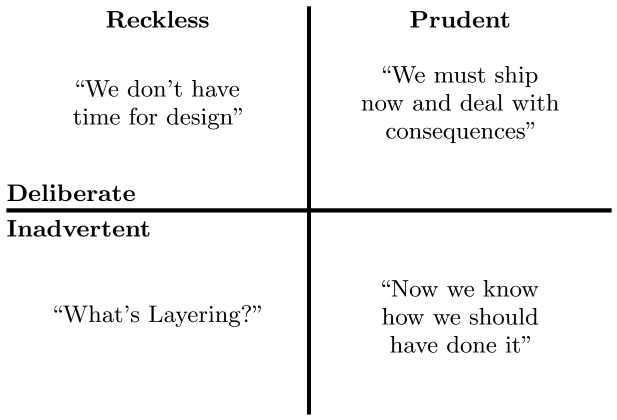

Reckless
Prudent
"We don't have time for design"
"We must ship now and deal with consequences"
Deliberate
Inadvertent
"What's Layering?"
"Now we know how we should have done it"

Figure 2: Technical Debt Quadrant by Martin Fowler [8]

For him the metaphor's primary task is to help "thinking about how to deal with design problems, and how to communicate that thinking" [8]. Nevertheless, it is used as a tool which can reveal possible drawbacks of current design decisions: the quadrant makes everyone aware that a certain decision could cause debt. This theoretical concept should then help to find a reasonable solution [25].

Technical Debt tries to help decide how to invest scarce resources: "Like financial debt, sometimes technical debt can be necessary" [2]. Most of the time this debt is not visible, as Kruchten also pointed out, leading to making debt visible as one purpose. Additionally, the value and present value play a role, including the difference between the actual state and an supposed ideal state as well as the time-to-impact. This involves a differentiation between "structural issues (the potential technical debt) and the effect it has on actual development (the effective technical debt)" [28], which could also be called problems and risks.

Overall debt has to be seen in its environment and it has to be determined if the debt was strategic or unintentional. Ultimately, this Technical Debt can be considered as an external software attribute which needs to be quantified [2, 18]. As a consequence one has to measure all criteria to estimate the debt and make a proper decision based on that information. It has been shown that reasonable decisions can be made more easily, if corresponding information (debt) is taken into account. Additionally, delaying a supposed "right" implementation has a significant impact on the cost of the project, so that an appropriate management of the debt concept is useful [3, 9, 22].

3.2 Enterprise Architecture

Now we want to focus on *Enterprise Architecture* and its quality issues: Before we proceed we want to clarify the term of quality for a system. In accordance with ISO/IEC 25010, quality "is the degree to which a product or system can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific contexts of use" [12].

As described in "Defining Enterprise Architecture: a Systematic Literature Review" by Saint-Louis et al. [27] the definitions diverge significantly, already hinting at differences in quality issues. Kappelman et al. pointed out that "The 'enterprise' portion of EA is understood by some as a synonym to 'enterprise systems', yet by others as equivalent to 'business' or 'organization'. [...] Even less uniform is the understanding of the meaning of 'architecture'. The most common understanding of the term is a collection of artifacts (models, descriptions, etc.) that define the standards of how the enterprise should function or provide an as-is model of the enterprise" [13].

Nonetheless, EA helps to face "ongoing and disruptive change" [27], by attempting to align IT and business strategy. In fact, 42% of the identified articles used in the SLR did not include an explicit definition of EA, making a uniform approach inappropriate [26, 27].

Despite the diverse perspectives and definitions, some articles focus on quality issues regarding EA. One could for example derive qualities that encompass IT system qualities, business qualities and IT governance qualities [26]. IT system qualities focus on performance, respectively efficiency, interoperability, availability, accuracy, maintainability and sustainability. The effectiveness, flexibility, integration and decision support evaluated in retrospect are qualities of the business itself. IT governance qualities include planning, organizing, monitoring and implementing IT-solutions while integrating them into the business process [26]. Of course the interaction of those factors and the resulting interoperability has to be treated as well.

Qualities / as-is situation		Situation A	Situation B	Situation C	Situation D
IT system	Performance	X		X	X
	Maintainability	X		X	X
	Sustainability	X		X	X
Business	Flexibility			X	
	Integration and Coordination		X	X	
	Decision Support			X	X
	Control & Follow Up			X	
	Organizational Culture			X	
IT organization	Plan & Organize		X	X	X
	Acquire & Implement			X	X
	Deliver & Support	X		X	X
	Monitor & Evaluate			X	X

Table 2: Top prioritized qualities for as-is situations A, B, C and D [26]

However, Henderson and Venkatraman's Strategic Alignment Model (SAM) identifies Business Strategy, Business Structure, IT Strategy and IT Structure as four key domains of strategic choice, the so-called "alignment domains" [10]. Based on this information an artifact-based framework for business-IT misalignment symptom detection was created in the article of the same name [24]. This framework also includes a first suggestion of catalogues for misalignment symptoms, EA artifacts and EA analysis methods, so that also rather invisible or underestimated elements are considered. Moreover, a link between those categories is established, in order to know which misalignment symptoms affect which artifacts and how this can be analyzed.

Addicks and Appelrath searched for key figures and their metrics in order to unitize the quality assessment of an application landscape. This approach can be applied to the EA as a whole, because business processes, for example, influence the application's quality. They stated that "All key figures must at least fit one of the following three conditions: (a) it must be used for indications of applications and be based on the application's attributes, (b) it must be an indicator of an application and its value is determined by attributes and relations from other EA artifacts (the applications' enterprise context), and (c) it must indicate a landscape's quality and therefore use all attributes of applications and their enterprise context" [1].

Since many aspects influence the system's properties, a unified meta model is difficult to create. Nevertheless, the results of a survey show that even different enterprise orientations with divergent focuses prefer certain qualities over others (Table 3):

Top qualities desired in to-be situation	
IT system	Interoperability
	Availability
	Security
	Usability
	Accuracy
Business	Efficiency
	Effectiveness

Table 3: High prioritized qualities in the to-be situation [26]

Despite the importance of these factors across multiple enterprises, there are also differences taking distinct as-is situations into account. In the survey those as-is situations were condensed into four clusters, such that for each cluster the top prioritized qualities can be seen (Table 2).

As-Is Situation A is technically oriented with IT/business alignment to be an important issue. Though, that alignment is regarded to be rather low and EA is not seen as a suitable instrument for such alignment. In *As-Is Situation B* business-driven planning is the most important subject and using EA instruments is only done in the last 2.5 to 4 years. *As-Is Situation C* focuses on all three identified factors, paying attention to business and IT-driven dimensions. Here the usage of EA instruments is quite high. The last *As-Is Situation D* is characterized by already high IT/business alignment and a fair history of EA usage (3.5 to 5 years). EA is considered as an adequate instrument for assistance and alignment [26].

In general, the survey shows that the striven qualities differ across enterprises, nevertheless, there are some general qualities that are desired in most situations. Thus, specialized prioritization and adaptation are needed for the EA and its IT/business alignment.

Kappelman et al. conducted a survey with the intention to find out what the purpose of EA is and what benefits it involves. The results show that with a strong agreement of the respondents the purpose and function of EA and the resulting benefits match the previously mentioned desired qualities. This means that EA can be considered as an appropriate tool for an enterprise. The respondents had to express their agreement with the suggested statements on a 5-point Likert scale.

Table 4 shows the agreement on five out of six suggested functions with the mean and the corresponding standard deviation. For example 92% considered that EA provides a blueprint for an organization's data, applications and technology; and 89% regard EA as a tool for organizational planning [13].

Purpose / Function	Mean	Std. Dev.
Provides a blueprint to the organization	4.397	0.682
A planning tool	4.249	0.676
Facilitate systematic change	4.016	0.739
A tool for decision making	4.134	0.709
An Alignment tool	4.083	0.857
Communicating organizational objectives	3.667	0.903

Table 4: Purpose & Function of EA [13]

Regarding the benefits of EA 20 statements had to be assessed. In this case, for example, 87% agreed that EA helps improve interoperability among Information Systems. 19 out of the 20 items had Agree and Strongly Agree responses accounting for more than 60%. The only suggested benefit that was not supported by the majority was related to improvement of organizational trust. The top five recognized benefits are shown in Table 5, again with the mean and the corresponding standard deviation [13].

Benefits	Mean	Std. Dev.
Improved interoperability among IS	4.336	0.718
Improved utilization of IT	4.252	0.696
Aligning business objectives with IT	4.139	0.801
More effective use of IT resources	4.024	0.738
Better situational awareness	3.946	0.715

Table 5: Benefits of EA [13]

More general approaches are followed by, for example, artifact based viewpoints of Winter and Fischer [34] or the TOGAF Standard [23]. They refer to domains like Business/Process Architecture, Software/Data Architecture, Application/Integration Architecture and Technology/Infrastructure Architecture.

Ylimäki goes even further and defines twelve critical success factors for Enterprise Architecture (can be seen in Figure 3).

These factors obviously influence EA and its quality, although they are different to previously known aspects. In this case high-quality EA is described with: "high-quality EA conforms to the agreed and fully understood business requirements, fits for its purpose (e.g. a more efficient IT decision making), and satisfies the key stakeholder groups' (the top management, IT management, architects, IT developers, and so forth) expectations in a cost-effective way understanding both their current needs and future requirements" [35].

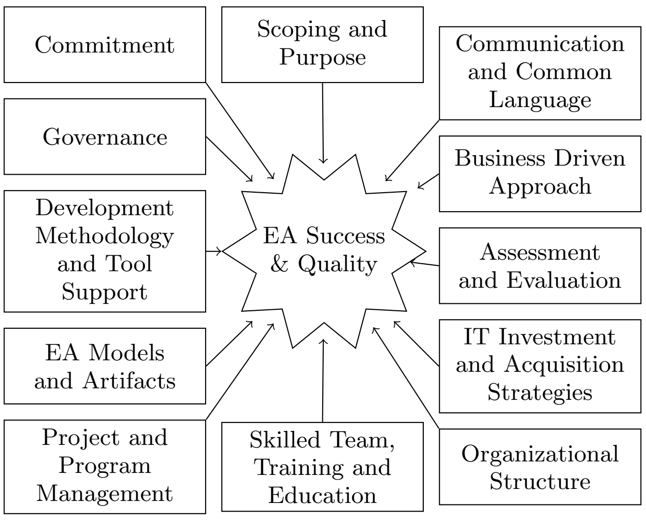

Commitment
Scoping and Purpose
Communication and Common Language
Governance
Business Driven Approach
Development Methodology and Tool Support
EA Success & Quality
Assessment and Evaluation
EA Models and Artifacts
IT Investment and Acquisition Strategies
Project and Program Management
Skilled Team, Training and Education
Organizational Structure

Figure 3: Set of potential critical success factors for EA [35]

Moreover, there is an Enterprise Architecture Model Quality Framework (EAQF) proposed by Timm et al. which builds upon six principles to assess the quality of EA models, namely the principles of validity, relevance, clarity, economic efficiency, systematic model construction and comparison. Each of the principles is augmented with quality attributes that can be assessed in order to determine the quality addressing the EA Model's purpose, a specific view of it and the overall EA Model [31].

This can also be seen in Figure 4 where the EA purpose, its objectives and the stakeholders' concerns affect the EA model's quality assessment. Here the EA model as a whole and each EA model view on its own are important to the quality, such that the interaction of multiple model views focusing on different issues should not be underestimated. Although this framework targets the quality of EA models and not the EA directly, it provides reasonable attributes for its quality assessment.

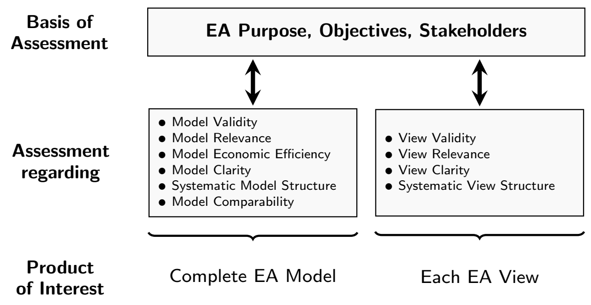


Figure 4: The Enterprise Architecture Model Quality Framework (EAQF) [31]

4. DEFINING ENTERPRISE ARCHITECTURE DEBT

As seen in section 3.1, in total the definitions of Technical Debt are mostly descriptive, naming properties and different types of debt, rather than explicitly defining the term. Nevertheless, Technical Debt is considered as a tool pointing at possible risks, measuring and tracking deficits and aiming at supporting the process of finding suitable solutions. The focus mainly lies on increasing awareness of also invisible or structural elements and providing a uniform basis for discussions and communication.

To outline the findings regarding Enterprise Architecture and its quality issues from section 3.2 we can say, in a nutshell, Enterprise Architecture is not clearly defined with a uniform description. Although it is more often described and its quality aspects are mentioned with respect to a specific viewpoint and enterprise orientation, the descriptions diverge. Hence, only a common understanding of EA and a basis for communication and discussion is established so that there is a need for proper adjusting of EA and its quality issues.

Therefore, the requirements differ for each enterprise, making a uniform approach according to EA models and quality issues impossible. Consequently, our definition of Enterprise Architecture Debt determines a technique providing some crucial factors in order to estimate an EA's quality for its specific purpose. This increases the awareness for possible suboptimal aspects that may cause severe drawbacks in the future. The metaphor of debt should serve as a common basis for communication and discussion, as well as pointing out differences in as-is situations and proposed ideal to-be situations [13]. The need for such a basis was observed before: Nan Niu et al. identified four possible situations regarding communication [20]:

1. consensus where stakeholders use terminology and concepts in the same way
2. correspondence where they use different terminology for the same concepts
3. conflict where they use the same terminology for different concepts
4. contrast where stakeholders differ in terminology and concepts

Furthermore, flexibility and robustness are identified as important factors in the dynamically changing and evolving environment of the enterprise and its structures. Hence, it can be used as a powerful tool, like Technical Debt, and help to manage a complex enterprise, because it allows to get a holistic overview and keep track of existing debt. All in all this should fulfill the purpose and function of EA as shown in Table 4 [13].

In order to achieve a scenario-unaware definition of Enterprise Architecture Debt we have to state what we refer to, when we are talking about EA. Saint-Louis et al. conducted a SLR and found multiple definitions of EA [27]. In this context we regard EA as a set of artifacts which are aggregated.

In the TOGAF Standard version 9.2 Business Architecture, Data Architecture, Application Architecture and Technology Architecture are identified as the four architecture domains [23]. Those roughly match the different layers of Winter and Fischer, although they named Process Architecture in particular (cf. Figure 5) [34].

The artifacts themselves and their importance for a specific enterprise may differ, so we focus on the following aspects in particular, which are general enough to be applied to the majority of EAs:

1. Enterprise Architecture layers
   1.1. Business Architecture
   1.2. Process Architecture
   1.3. Software Architecture and used services (SaaS)
   1.4. Technology Architecture (hardware)
2. Other influencing factors
   2.1. Stakeholders
   2.2. Guidelines and Standards

Regarding EA as a set of artifacts can contain more than the mentioned, but it is easy to add and remove artifacts later, so the metaphor of Enterprise Architecture Debt can be adapted to individual situations. Furthermore, there are aspects we explicitly will not take into consideration, like the abilities of people working on and with an EA, because a measurement and an evaluation are too subjective. We will focus on a definition evaluating the quality of an EA and not a cost-benefit-analysis, although it can be applied to our definition later.

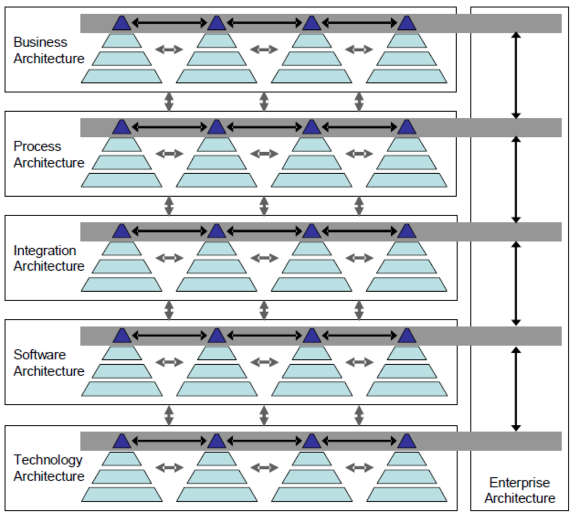


Figure 5: Enterprise Architecture as a Cross-layer View of Aggregate Artifacts [34]

Winter and Fischer point out that "Most of the artifacts [...] in EA can be represented as aggregation hierarchies" [34] and introduce the schema at Figure 5.

Having a look at Figure 5 and the definition of Technical Debt by Cunningham [4], EA Debt in general can be understood as an aggregation of taking debts in each layer, but just adding up each artifact would not give a concrete overview of the current situation, it could even whitewash huge issues in the whole EA. Therefore every artifact, and also every part an artifact consists of, has to be weighted. Obviously there is no uniform weighting function, because it depends on how much each artifact is represented in a concrete EA. On top of that we need to take into consideration, that the interfaces of the artifacts can cause debt, too. An artifact might be optimized and work perfectly, but if there is a huge overhead caused by interfaces respectively multiple lines of reporting, the EA does not.

As seen in Table 3 businesses prioritize efficiency and effectiveness so we assume, that "Assessment and Evaluation" presented by Ylimäki as a critical success factor for EA & Quality [35] is not causing any debt. Of course the assessment and its quality are important, but evaluating those evaluations is an unnecessary overhead. We assume that with EA Debt and other existing frameworks this aspect can be ignored.

Now, that we clarified our conditions and assumptions, we come up with our definition for Enterprise Architecture Debt:

According to Hurley a definition consists of "the definiendum", the word to be defined, and "the definiens", the words that do the defining. Those again can be split into "generic elements" and "specific elements" or "characteristics" [11], so that our definition reads as follows:

Enterprise Architecture Debt is a metric, that depicts the deviation of the currently present state of an enterprise from a hypothetical ideal organizational position and the principles governing its design and evolution. This includes the different layers, respectively domains, of Enterprise Architecture and its associated artifacts.

Based on this definition we can explain and characterize appropriate objectives and details:

Enterprise Architecture Debt arises, when debt is taken in an artifact, which an EA consists of. This means that an element is not implemented or executed optimally in relation to the supposed ideal situation. Taking debt in a low hierarchy can be helpful and pay off, but it has to be "repaid" as fast as possible. Otherwise the whole EA would rely on that debt and use faulty or considered bad artifacts. This entails a high risk of additional debt and hinders the development. EA Debt is further increased by bad interfaces or bad interoperability and different priorities of stakeholders, not conform with an EA that is considered good by evaluation approaches.

Again a focus on mainly invisible elements can be helpful, because these factors may not even be recognized or their impacts are underestimated. By increasing the awareness for such issues the overall inadvertent debt can be reduced. Assuming a prudent management, this would lead to debt mostly being taken consciously and planned strategically. This means that possible repercussions are weighed and less trade-offs are accepted, which unnoticeably impair the enterprise.

Hence, the identified differences between the current state and the supposed ideal situation can be managed, such that the sometimes necessary debt, namely deliberate prudent debt, holds the largest share. The characterization of debt in the Technical Debt Quadrant (Figure 2) is of course also valid for Enterprise Architecture Debt, because EA Debt is an expansion of Technical Debt [2, 8].

The process and effort going into evaluation of artifacts and the whole EA does not belong to EA Debt as mentioned before, although the importance of correct assessment and its quality should not be neglected.

4.1 Evaluating EA Debt

Applied standards and guidelines are an indication for reliable artifacts and therefore lower EA Debt. The Enterprise Architecture Model Quality Framework (EAQF) [31] assesses the quality of an EA model with six principles. These principles can be used to construct an ideal situation and detect the differences to the current as-is situation. In this way also rather invisible or unconsidered elements can be found which then form a good entry point for EA Debt estimations. Yet again, the awareness of invisible or hardly noticeable elements has to be increased, because they in particular entail high risk of further debt in the future.

Dóra Őri proposed an artifact-based framework that aims at detecting misalignment in an undesired state of the enterprise, which can be used to detect also EA Debt. Based on the strategic alignment perspectives Strategy Execution, Technology Transformation, Competitive Potential and Service Level (from Henderson and Venkatraman's Strategic Alignment Model (SAM) [10]) the framework decomposes those perspectives into corresponding perspective components, namely the "alignment matches". Then they are connected to typical misalignment symptoms, using a misalignment symptom catalogue as a reference. After that, relevant containing artifacts are identified, again using an artifact catalogue. Finally, suitable EA analysis types are collected respecting the affected artifacts. This article already set up catalogues based on other research that can be used as a reference [24].

Despite those frameworks present helpful approaches to evaluate EA and its models, there are also some shortcomings. They can be referred to EA itself or to our new idea of EA Debt. As mentioned already before, the development of a suitable ideal situation for an enterprise is still a hard task. Furthermore, there exists no uniform solution, so that an enterprise can hardly assume the structure of another use case.

Some observations are made by Schmid regarding shortcomings of Technical Debt, that can be transformed and extended to EA Debt: "Technical debt should be evaluated with respect to future evolution. [...] We need to differentiate between the structural issues (the potential technical debt) and the effect it has on actual development (the effective technical debt). [...] There is nothing like a technical-debt-free system." [28]. In general for the entire organization and our concept of Enterprise Architecture Debt this implies:

1. EA Debt should be evaluated with respect to future evolution, because further development may rest upon the current suboptimal implementations and structure.
2. We need to differentiate between the structural issues (the potential EA Debt) and the effect it has on actual development (the effective EA Debt). One could also refer to problems and risks that can have diverging impacts on the EA. Even though something is not implemented in the best possible way, it may not have a severe effect on the quality.
3. There is nothing like an EA-Debt-free system. So the hypothetical ideal state will never be reached. Only reducing debt can underline a good development, as long as the debt was valued correctly.

4.2 Possible impacts on EA Debt

As mentioned before there are multiple artifacts that have an impact on EA Debt, and identifying them should be an outcome of EA Debt evaluation. Having a look at the development of artifacts over time [1] known to cause debt can be one way to approach the search of new impacts to the current EA Debt situation. Finding a slow or even stagnating development in an artifact A means there is almost no effort to lower the debt and therefore artifacts relying on A will increase the overall EA Debt. Starting a search at A can help finding impacts on EA Debt. On top of the already mentioned artifacts that are able to cause debt, there are more hidden or invisible aspects. We propose some of them:

1. Communication overhead (documentation)
2. Interface bottlenecks (incompatibility)
3. Contradictory goals of stakeholders
4. Integrity problems or information inconsistency

Adapted to individual EAs there can be many more, these are just some artifacts that can have additional impact on EA Debt.

Another impact that has to be taken into consideration is that EA Debt itself should not be ignored while evaluating other artifacts. Even deferring the evaluation can, but not has to, increase EA Debt exponentially. Ideally, everyone working on artifacts of an EA should at least know the concept, so everyone is aware that his acting can cause EA Debt.

5. EXAMPLES

We come up with two made up and simplified examples, one showing in which cases taking EA Debt is useful and one showing that EA Debt is lowering the efficiency of the whole enterprise and should be removed as fast as possible.

5.1 Useful EA Debt

An enterprise started as a start-up and used one database to manage all data they want to keep with the according Data Models. Due to enormous success and much growth the initial database (DB) and database management system (DBMS) needs to be replaced, because when starting the business the founders did not have a fast and huge growth in mind. Currently they are relying on the old DB and DBMS, which causes other applications to run slow, resulting in increasing EA Debt.

The founders are aware of the problem and want to buy multiple new servers, and a new DBMS, which will keep the old data models, if the growth of the business continues at least one month. Updating the interface of the new DBMS is already in progress, thus it should work with the current EA, especially with the existing software systems, so it can be applied fast when the decision is made.

Relying on the old DB and DBMS is consciously taken EA Debt and useful in this case because it can keep the enterprise from making a bad investment. In this case it is an artifact that indeed causes other artifacts to be not as performant and efficient as they can be, but the problem can be improved without touching any other artifact.

5.2 EA Debt hinting at suboptimal decision

A small company A in the chemical industry has developed an efficient process to produce one certain product. The process is highly optimized for a certain input and therefore works really well, if it stays with the given input. Another bigger company B, dealing with chemicals with law restrictions, bought the smaller company A, because the highly optimized process works for B's products well too. Integrating A's IT systems into B's caused EA Debt in the beginning, but the integration was planned and prepared well and successfully.

What B did not have in mind was, that not every worker at company A is authorized to work with their law restricted chemicals and replacing them would cause more debt, because training new employees to learn the optimized process takes too much time. Taking EA Debt here in the business processes would not pay back. The company cannot use the efficiency of the process which was the original reason for them to buy A. Also scaling up is difficult, because of the high optimization they would need to duplicate everything that is needed for production, so the employees and training them is causing debt. Assuming there is no different option for B, they have to take debt in any way to lower the loss and hopefully made an investment that pays off in the future.

Taking debt in this case is different from the first example in section 5.1, because the enterprise takes debt to lower the risk of a misinvestment and therefore, in the worst case, lowers the income. Here in the second example (section 5.2) the investment is already done and EA Debt has to be taken, to make the investment a possibly good investment, so the debt should be removed as fast as possible. Thus EA Debt also emphasizes this risk.

In two made up and simplified examples we showed, that EA Debt can be a tool, like monetary debt, to grow (Example 1, section 5.1), but it can also be dangerous, if it potentially can not be repaid and consequently does not pay off (Example 2, section 5.2). In contrast to monetary debt, EA Debt can suddenly occur, but does not have to. This means that enterprises have to be careful with taking EA Debt and even more with not lowering it. Still EA Debt points at those risks and problems to facilitate a sustainable development. Another example, illustrating an as-is situation with EA Debt in an EA would be rather uninteresting, because there are no universal guidelines yet, how to deal with EA Debt (see future work, section 7) but every enterprise has to deal with EA Debt themselves. Nevertheless, lowering EA Debt should be the main goal in such a situation. With the two examples we demonstrate a way how to use the concept of EA Debt as a tool to prevent bad decisions regarding the whole EA.

6. CONCLUSION

We defined a new metaphor of *Enterprise Architecture Debt* on the basis of the idea of Technical Debt and quality issues regarding Enterprise Architectures and its models. We generalized the approach of Technical Debt as a tool for software engineering to make EA Debt a tool for the whole enterprise architecture. Therefore, Technical Debt definitions and descriptions are inspected in order to find a way to define our new metaphor. EA Debt consequently is an expansion to Technical Debt. Furthermore, diverging definitions of EA and thus different quality attributes are pointed out. EA Debt then refers to those quality issues and serves as a common basis for discussion and communication. This means, both IT and business departments can manage the structure of the enterprise in collaboration. Besides, the debt concept should facilitate the alignment of the enterprise, because it draws attention to suboptimal and mostly invisible elements.

By assessing the quality and measuring the emerging EA Debt identifying differences between the actual as-is state and a supposed ideal state, the debt concept can help to decide and increase the awareness of suboptimal aspects. As our definition builds upon EA being a set of artifacts, this set can be adapted to the specific enterprise, making the idea of EA Debt a universal approach that can be tailored to meet the particular needs of the specific enterprise. EA itself can be considered an appropriate concept to achieve certain goals and improvements, which is now extended by the idea of debt to further increase the capabilities to fulfill the enterprise's requirements. Consequently, EA Debt is flexible in its use, so that one can decide what is to be managed and to what extend by this concept. Needless to say, the idea targets to support the overall enterprise with its EA to benefit from all capabilities of EA Debt and maintain sustainable development.

The main focus is on managing the overall Enterprise Architecture, which includes the awareness of existing problems in particular. As already seen, mainly invisible elements entail a high risk of further debt, which again impairs the EA's quality. By the EA Debt term, one can keep track of undesired aspects, namely the debt, which serves as a holistic overview over potential tasks and improvement possibilities. Of course some sort of ideal situation has to be established as a goal state, such that it can be pursued with Enterprise Architecture Debt and other suitable frameworks.

Finally, we presented two small and simplified examples to demonstrate the idea behind the concept of Enterprise Architecture Debt. Additionally, two different cases, in which taking and lowering the debt affects the enterprise in multiple ways, are highlighted. Based on the diverging complexity of an enterprise and its artifacts, also the EA Debt can differ.

7. FUTURE WORK

In this paper we defined the idea of Enterprise Architecture Debt. However, further and a more detailed categorization of possible factors should be developed as an equivalent to the well-known "Code Smells" in software respectively the Technical Debt. This would be like a catalogue for typical bad decisions and bad habits and should include technical aspects, EA quality aspects and in particular aggregations of them. The catalogues from Dóra Őri of misalignment symptoms, EA artifacts and EA analysis methods can be used as a suggestion [24].

Furthermore, EA Debt items regarding different types of debt could be identified based on EA layers respectively the artifacts. This includes the elaboration of a way of identifying EA Debt with use cases and case studies. All of this could result in an EA Debt classification tree with different debt types and affected artifacts.

Another valuable approach could be an adaption to existing EA Frameworks or guidelines, such that there is no doubt about what belongs so EA Debt and what does not in the context of different models for EA. Working with the concept of EA Debt in future could also give insight on aspects, never considered before belonging to EA Debt or EA in general.

Additionally, some kind of weighting or an approach for an appropriate weighting should be established, such that the impact of certain aspects on EA Debt and thus the EA's quality can be derived. In this context Portfolio Theory can be used to calculate the "optimal" fields for improvement. This would of course lead to a more sophisticated tool that facilitates and supports the entire EA. This could also help to establish some kind of guide, how to structure the process of lowering EA Debt or a cost-benefit approach on dealing with EA Debt. Overall, the management can be based on more analytic methods to ensure development and prevent risks as well as problems.

Eventually, EA Debt evolves to a versatile toolbox enhancing quality assessment, planning and decision-making for the entire enterprise. This requires further research on EA Debt Management as an upgrade to the general EA Management.

8. ACKNOWLEDGEMENTS

We thank Simon Hacks, our research supervisor, for his valuable and constructive suggestions during the development of this work. The given critiques have been appreciated. We also want to thank Yoon Chow Yeong for her support.

9. REFERENCES

[1] J. S. Addicks and H.-J. Appelrath. A method for application evaluations in context of enterprise architecture. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 131–136, New York, NY, USA, 2010. ACM.

[2] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 47–52, New York, NY, USA, 2010. ACM.

[3] Z. Codabux and B. Williams. Managing technical debt: An industrial case study. In *2013 4th International Workshop on Managing Technical Debt (MTD)*, pages 8–15, May 2013.

[4] W. Cunningham. The wycash portfolio management system. In *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*, OOPSLA '92, pages 29–30, New York, NY, USA, 1992. ACM.

[5] B. Curtis, J. Sappidi, and A. Szynkarski. Estimating the principal of an application's technical debt. *IEEE Software*, 29(6):34–42, Nov 2012.

[6] B. Curtis, J. Sappidi, and A. Szynkarski. Estimating the size, cost, and types of technical debt. In *2012 Third International Workshop on Managing Technical Debt (MTD)*, pages 49–53, June 2012.

[7] M. Fowler. Technical debt. `https://martinfowler.com/bliki/TechnicalDebt.html`. Last accessed on 2018-10-16.

[8] M. Fowler. Technical debt quadrant. `https://martinfowler.com/bliki/TechnicalDebtQuadrant.html`. Last accessed on 2018-10-16.

[9] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. B. D. Silva, A. L. M. Santos, and C. Siebra. Tracking technical debt — an exploratory case study. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 528–531, Sept 2011.

[10] J. C. Henderson and H. Venkatraman. Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal*, 38(2.3):472–484, 1999.

[11] P. Hurley. *A Concise Introduction to Logic*. Available Titles CengageNOW Series. Cengage Learning, 2005.

[12] ISO/IEC 25010. Systems and software engineering - systems and software quality requirements and evaluation (square) - system and software quality models. Standard ISO/IEC 25010:2011, International Organization for Standardization, Geneva, CH, 2011.

[13] L. Kappelman, T. McGinnis, A. Pettite, and A. Sidorova. Enterprise architecture: Charting the territory for academic research. *AMCIS 2008 Proceedings*, page 162, 2008.

[14] B. Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

[15] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams.

An enterprise perspective on technical debt. In *Proceedings of the 2Nd Workshop on Managing Technical Debt*, MTD '11, pages 35–38, New York, NY, USA, 2011. ACM.

[16] P. Kruchten, R. L. Nord, and I. Ozkaya. Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21, Nov 2012.

[17] P. Kruchten, R. L. Nord, I. Ozkaya, and D. Falessi. Technical debt: Towards a crisper definition report on the 4th international workshop on managing technical debt. *SIGSOFT Softw. Eng. Notes*, 38(5):51–54, Aug. 2013.

[18] L. Lavazza, S. Morasca, and D. Tosi. Technical debt as an external software attribute. In *Proceedings of the 2018 International Conference on Technical Debt*, TechDebt '18, pages 21–30, New York, NY, USA, 2018. ACM.

[19] S. McConnell. Managing technical debt slides, 2007.

[20] N. Niu, L. D. Xu, J. C. Cheng, and Z. Niu. Analysis of architecturally significant requirements for enterprise systems. *IEEE Systems Journal*, 8(3):850–857, Sept 2014.

[21] R. L. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas. In search of a metric for managing architectural technical debt. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pages 91–100, Aug 2012.

[22] F. Oliveira, A. Goldman, and V. Santos. Managing technical debt in software projects using scrum: An action research. In *2015 Agile Conference*, pages 50–59, Aug 2015.

[23] Opengroup. The togaf standard, version 9.2. `http://pubs.opengroup.org/architecture/togaf92-doc/arch/`. Last accessed on 2018-11-21.

[24] D. Őri. An artifact-based framework for business-it misalignment symptom detection. In J. Horkoff, M. A. Jeusfeld, and A. Persson, editors, *The Practice of Enterprise Modeling*, pages 148–163, Cham, 2016. Springer International Publishing.

[25] K. Power. Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options. In *Proceedings of the 4th International Workshop on Managing Technical Debt*, MTD '13, pages 28–31, Piscataway, NJ, USA, 2013. IEEE Press.

[26] J. Saat, U. Franke, R. Lagerstrom, and M. Ekstedt. Enterprise architecture meta models for it/business alignment situations. In *2010 14th IEEE International Enterprise Distributed Object Computing Conference*, pages 14–23, Oct 2010.

[27] P. Saint-Louis, M. C. Morency, and J. Lapalme. Defining enterprise architecture: A systematic literature review. In *2017 IEEE 21st International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 41–49, Oct 2017.

[28] K. Schmid. On the limits of the technical debt metaphor: Some guidance on going beyond. In *Proceedings of the 4th International Workshop on Managing Technical Debt*, MTD '13, pages 63–66, Piscataway, NJ, USA, 2013. IEEE Press.

[29] C. Seaman. Measuring and monitoring technical debt, 2013.

[30] T. Theodoropoulos, M. Hofberg, and D. Kern. Technical debt from the stakeholder perspective. In *Proceedings of the 2Nd Workshop on Managing Technical Debt*, MTD '11, pages 43–46, New York, NY, USA, 2011. ACM.

[31] F. Timm, S. Hacks, F. Thiede, and D. Hintzpeter. Towards a quality framework for enterprise architecture models. In *Proceedings of the 5th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2017) co-located with APSEC*, volume 4, page 14âAS21, 2017.

[32] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. D. Penta, A. D. Lucia, and D. Poshyvanyk. When and why your code starts to smell bad. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 403–414, May 2015.

[33] J. Webster and R. T. Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly*, pages xiii–xxiii, 2002.

[34] R. Winter and R. Fischer. Essential layers, artifacts, and dependencies of enterprise architecture. In *Enterprise Distributed Object Computing Conference Workshops, 2006. EDOCW'06. 10th IEEE International*, pages 30–30. IEEE, 2006.

[35] T. Ylimäki. Potential critical success factors for enterprise architecture. *Tietotekniikan tutkimusinstituutin julkaisuja, 1236-1615; 18*, 2008.

Comparison of Failure Diagnosis Techniques in Combinatorial Testing

Felix Engel
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
Felix.Justus.Engel@rwth-aachen.de

Umair Munir
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
Umair.Munir@rwth-aachen.de

Konrad Fögen
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
Konrad.Foegen@swc.rwth-aachen.de

ABSTRACT

Testing a software can be very difficult and time consuming. The combination of combinatorial testing, which massively reduces the amount of test cases that need to be covered, and failure diagnosis, which helps the developer to find the actually interesting and problematic parameter combinations, can cut down the time and complexity of this task considerably. There are multiple algorithms that employ this approach but implement different strategies to localize those faults. We study 9 different approaches and categorize them in order to identify and compare differences.
To do so we first give an introduction to combinatorial testing and fault diagnosis, explain our categories and categorize each approach and explain it in detail. Then we discuss noteable differences, benefits and downsides comparatively.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management—*programming teams, productivity, software configuration management*

Keywords

fault localization, fault characteriztaion, combinatorial testing, software debugging, approach comparison, delta debugging

1. INTRODUCTION

Software testing and debugging is very important and often expensive part in software development process and can be very tedious for the developers. Combinatorial testing is one of the techniques that is used for testing software efficiently. Under the assumption that all faults in the system are caused by not more than n parameters, it allows to efficiently cover the whole application by only testing all n-parameter interaction combinations, which is significantly easier than exhaustive testing while still detecting all faults.

Failure diagnosis aims to not only detect the existence of faults but also to algorithmically determine the causing combinations of values without further input from the developer. Failure diagnosis as described by Nie et al. [13], on which we will focus in this paper, is an important part of failure diagnosis and is also known as fault characterization. It isolates which parameters and values actually cause the failure of the test.

We will perform a literature review on such strategies, showcase their workings, compare them with one another and discuss their effectiveness for certain requirements.

Section 3 lays out our methodology of searching for literature. In Section 4 we explain different methods and algorithms that uses combinatorial testing strategies. In Section 5 we compare these methods. In Section 6 we discuss the conclusion of our search.

2. BACKGROUND

In this chapter we give an overview of combinatorial testing and failure diagnosis in combinatorial testing and terminologies we will use throughout.

The system or software under test is generally referred to as *SUT*. Given a test case with n parameters, a *combination* (or parameter combination) is a combination of k values for each parameter with $1 \leq k \leq n$, such that the parameters fit the signature of the test case and all values are elements of the domains of the parameter they represent. A set of these combinations is called a *test suite*. A *fault* is an error in the SUT caused by code or other aspects of the SUT. A *failure* is the failing of a test case, i.e. the returned test result did not match the expected one. A *failure-inducing combination* simply is a parameter combination which always causes its associated test case to fail when passed to it. A set of failure-inducing combinations derived from a set of test cases is *complete* if it contains every combination that cause the test cases to fail. Algorithms which work by removing non-inducing combinations from a set of combinations to identify the failure-inducing combinations also define something like a *suspicious combination* which just is a combination that is involved in a failing test, but could not yet be determined to be the cause of the failure.

Combinatorial testing tries to reduce the amount of test cases which need to be executed by using *t-way combinations* of input parameter values where t is the number of input values. A combinatorial t-way test set is designed to cover all interactions of t different parameters. There could be possibly one or many such combinations that can effect the execu-

Table 1: Values of Input Parameters

Browser	OS	Connection	DB
Firefox	Linux	LAN	DB/2
IE	Windows	ISDN	Oracle
Opera	Macintosh	Modem	Access

Table 2: Combinations of Input Parameters

No.	Browser	OS	Connection	DB
1	Firefox	Linux	LAN	DB/2
2	Firefox	Windows	ISDN	Oracle
3	Firefox	Macintosh	Modem	Access
4	IE	Linux	ISDN	Access
5	IE	Windows	LAN	Oracle
6	IE	Macintosh	Modem	DB/2
7	Opera	Linux	Modem	Oracle
8	Opera	Windows	LAN	Access
9	Opera	Macintosh	ISDN	DB/2

tion of a system to generate failures. t is typically relatively small ($t \leq 6$) because the large number of software failures are caused by interactions of only a few parameters as proposed by Kuhn, Wallace and Gallo [11]. For example execution of a program can be effected by Browser, OS(Operating System), Connection and DB(Database) which can be possible input parameters for test cases [16]. Suppose, if we have 3 possible values for each parameter as shown in Table 1. If we apply exhaustive testing that covers all possible combinations, then it will require $3^4 = 81$ test cases but *2-way combinatorial testing* requires only 9 test cases to cover all pairs as shown in Table 2. Mentionable combinatorial test set generation tools, which are mentioned by many of the approaches we will discuss in this paper, are AETG which is proposed by Cohen et al. [3] and ACTS which is proposed by Hagar, Wissink et al. [8].

Failure diagnosis is a technique to aid the developer in the debugging process of a system or software by not just telling him which tests failed but also which parameter value combinations caused a test to fail, significantly cutting down on the time and resources the developer needs to spend to pinpoint those combinations themselves. For example suppose the test 4 (IE, Linux, ISDN, Access) failed. In a worst case with only exactly 2-way interactions considered, the developer would have to test $\binom{4}{2} = 6$ different combinations, and with up to 3-way interactions, so including 1/2-way interactions, they would have to look at up to $\binom{4}{1} + \binom{4}{2} + \binom{4}{3} = 14$ different combinations. Those numbers can get overwhelming fast with more complex systems.

3. METHODOLOGY

In this chapter we expound how we conducted our literature review for our study.

In our search for failure diagnosis (more specifically failure diagnosis) approaches in combinatorial testing we began with forward and backward searches on the BEN papers by Ghandehari et al. [7], [6], [5] which led to the AETG system proposed by Cohen et al. [3], [1], a widely used combinatorial test set generation tool, on which we performed a very successful forward search. We especially used the keywords "failure diagnosis", "fault localization", "fault interaction", "fault detection", "t-way interactions", "pair-wise testing" to filter the large amount of references. The related works in some of the more recent papers by Nishiura et al. [14] also proved very helpful to find significant results.

Overall we tried to focus on newer approaches, but a lot of the fundamental research dates back to and even before the 2000s.

We considered including a paper if it either directly provided an algorithmic approach for failure diagnosis in combinatorial testing or contained further information about an already included approach.

We included nine such approaches, with one notable exception to the latter; Colbourn et al. [4] introduced LDA which was a very extensive and fundamental work on non-adaptive failure diagnosis approaches and was included even though it did not directly propose an algorithmic solution, because it provides important knowledge for the approaches propsed by Nagamoto et al. LDA(1,2) [12] and LDA(d,t) [10].

In total we considered around 60 research papers and articles regarding the general topic of fault localization and failure diagnosis in combinatorial testing.

4. CATEGORIZATION AND RESULTS

In this chapter we present our results, categorize the algorithms we have considered for our study, discuss each one and explain their basic approach to failure diagnosis.

Combinatorial failure diagnosis approaches can be classified into two categories: (1) Test generation, which is generation of new test cases, and (2) Result Analysis, which is only the analysis of combinatorial testing results. The division of approaches into these categories if important because the generation of new test cases is a major expense, which has to prove it is worthwhile making.

The approaches we will cover in this chapter fall under those categories as follows

Test Generation:	BEN, AIFL, IterAIFL, FIC, FIC_BS, DDmin, LDA
Result Analysis:	FCI, FROG

Approaches in the category test generation can be further distinguished into adaptive and non-adaptive. We call an approach *adaptive* when the new test cases it generates are based on the results of previously run tests in some way. This includes BEN [6], AIFL [15], IterAIFL [16], DDmin [17], FIC and FIC_BS [19]. We call it *non-adaptive* when the generated test cases are not based on previously run tests. This includes the LDA approaches [4],[12],[10]. We make this distinction because generating tests adaptively can potentially mean a larger expense as we have to execute first, in many of the iterative approaches multiple times, the test cases, but it can also help in providing much more accurate results.

The adaptive approaches can be even further distinguished by looking at whether they work on the whole set of test cases and results or on one test case at a time. We call the former behaviour *parallel*. Parallel approaches are BEN [6] and IterAIFL [16] (AIFL technically is as well, but as it only does one iteration of its process, we omit it here). See Figure 1 for an overview.

Differences, benefits and problems of all categories will further be discussed in Section 5.

4.1 BEN

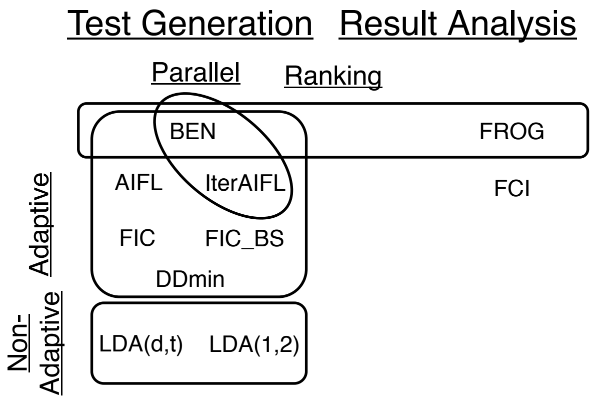


Figure 1: An overview of algorithms and their properties

Ghandehari, Chandrasekaran et al. [5] introduced the BEN tool which is a combinatorial testing-based fault localization tool, which consists of two phases of which only the first one proposed by Lei and some others [7] is relevant to our study as it deals with the failure diagnosis. It employs an iterative, adaptive approach which ranks parameter combinations by their probability to be failure-inducing.

Initially all suspicious combinations are identified and ranked through a first, full-coverage combinatorial-testing set. This ranking of suspiciousness is based on the two concepts of suspiciousness of combinations and suspiciousness of the environment of combination. Basically, the higher the suspiciousness of a combination, the lower the suspiciousness of its environment, the higher the combination is ranked. Next a set of new tests is generated for each suspicious combination (can be restricted to only the most suspicious ones), such that the suspiciousness of the environment of each combination is minimized. If those test fail, it is more likely that the combination is in fact inducing. This can be repeated until an arbitrary stopping condition is satisfied.

Because not as highly ranked (according to a specified threshold) combinations might be ignored by the algorithm for next a next iteration, the result set of combinations is not necessarily complete. This represents a trade-off between completeness of the result and swiftness of calculation, a user of the tool can balance for themselves.

4.2 FROG

Nishiura, Choi and Mizuno [14] introduced FROG (**F**IL based on **R**egression coefficients **o**f logistic re**g**ression analysis) which takes input of all interactions to identify the minimal failure-inducing interactions. It applies logistic regression analysis on input combinations of a given test suite to identify the logistic regression coefficient and excludes the potentially failure-inducing interactions which have a coefficient value below the given threshold limit to produce minimal set of failure-inducing interactions with suspicious combinations.

This coefficient is calculated for each input parameter via data learning technique from combination of input parameters in each iteration. It applies k-way testing on input combinations. It starts iteration with $k = 1$ and increases the value of by 1 in each iteration until maximum number of input values is reached. In each iteration it excludes the *failure-inducing interactions* which are below the given threshold value and are considered as less suspicious. By this way FROG returns only the set of most suspicious input combinations.

Table 3: 4-way Test Suite and their testing result

tc	p1	p2	p3	p4	p5	result
#1	1	2	1	2	2	pass
#2	1	1	2	2	2	fail
#3	1	2	2	2	3	pass
#4	2	1	1	1	2	pass
#5	1	1	1	2	3	fail
#6	2	2	2	1	1	pass

Table 4: Logistic regression coefficients for 1-tuples

1-tuple	logistic regression coefficient
$(p_1.1)$	16.341
$(p_2.1)$	15.990
$(p_3.2)$	-13.679
$(p_4.2)$	17.839
$(p_5.2)$	-1.277
$(p_3.1)$	-13.834
$(p_5.1)$	-1.403

As an example suppose we have a test suite as shown in Table 3 which has passed and failed test cases. FROG initializes a suit of test combinations TC, failure-inducing combinations FC and passed combinations PC.

$$TC = \{\#1,\#2,\#3,\#4,\#5,\#6\}$$
$$PC = \{\#1,\ \#3,\ \#4,\ \#6\}$$
$$FC = \{\#2,\#5\}$$

From FC combinations it creates TP which are all the n-way possible parameter combinations from TC with their values. In Table 3 combination #2 and #5 are the failure-inducing combinations. Thus, set of failure-inducing interactions TP after *1-way* iteration would be:

$$TP = \{(p_1.1), (p_2.1), (p_3.2), (p_4.2), (p_5.2), (p_3.1), (p_5.3)\}$$

Then for each parameter it will calculate the regression coefficient by using input data as calculated in Table 4. For further iterations it will take only those parameters which have a high coefficient value greater than the given threshold which is *0*. From Table 4 sTP which is a set of selected failure-inducing interaction is given as:

$$sTP = \{(p_1.1), (p_2.1), (p_4.2)\}$$

Thus, for the second iteration extract all 2-tuples in FC to identify TP

$$TP = \{(p_1.1, p_2.1), (p_1.1, p_4.2), (p_2.1, p_4.2)\}$$

and FI would be:

$$FI = \emptyset$$

Table 5: Logistic regression coefficients for 2-tuples

2-tuple	logistic regression coefficient
$(p_1.1, p_2.1)$	8.812
$(p_1.1, p_4.2)$	10.116
$(p_2.1, p_4.2)$	9.735

and after running the regression coefficient the coeffiecient value of each paramter is given in Table 5. It will again calculate the sTP:

$$sTP = \{(p_1.1, p_2.1), (p_1.1, p_4.2), (p_2.1, p_4.2)\}$$

Because each coefficient value of each parameter is greater 0 so all paramters in Table 5 will be considered in next iteration. If the combination in TP is not in PC then it will be added into *FI*. Where FI would be the set of minimal failure-inducing interaction. The process will continue to n-way iterations to identify the FI where n is 4 for the given data.

It will again do the same process for iteration 3. In third iteration for *3-tuples* only following combination will be considered for TP:

$$TP = \{(p_1.1, p_2.1, p_4.2)\}$$

and thus the most suspicious failure-inducing interactions FI calculated in iteration 3 would be:

$$FI = \{(p_1.1, p_2.1, p_4.2)\}$$

At this step sTP is empty so the algorithm finishes.

4.3 FIC

Zhang, Zhiqiang and Jian [19] introduced FIC (**f**aulty **i**nteraction **c**haracterization) which uses adaptive approach to reduce the number of test cases. If system has k parameters, then It uses at-most k adaptive test cases to locate failure-inducing interaction. These algorithms not only focuses on interaction coverage and reducing test cases but also focuses on diagnosis. FIC runs the test cases on given number of parameters and identifies for which parameters the failure-inducing interaction occurs by iterative approach. The maximum number of adaptive test cases run by FIC are equal to number of given parameters.

It runs test case on given parameters and identifies the combinations for which the failures occur and marks those combinations as fixed parameters. Then it uses this test case as seed test case and runs iteratively by changing the value for each parameter and set those parameters as free parameters until no failures occur. If a failure-inducing interaction is discovered by changing the value of parameter, then these parameters are marked as a candidate parameter.

4.4 FIC_BS

FIC_BS (**f**aulty **i**nteraction **c**haracterization with **b**inary **s**earch) introduced by Zhang, Zhiqiang and Jian [19] uses the FIC with binary search. Instead of exhaustive approach it iteratively distributes the parameters into halves until the failure-inducing parameters are found for each interaction. It iteratively search for each half for failure-inducing interaction. It locates the fixed parameters for each interaction then apply test for each fixed parameter to identify candidate parameter. Once the candidate parameter is found, the rest of the parameters are marked as free parameters. The process continues until failure-inducing parameters are identified. The same procedure is applied for each half. For each test case it returns either pass or fail on each iteration.

4.5 DDmin

The DDmin (**D**elta **D**ebugging algorithm that generalizes and simplifies the failing test case to a **min**imal test case) introduced by Zeller and Hildebrandt [17] is an algorithm which works by minimizing the parameters to reach the failure inducing part of a combination by checking each possibility of parameters that can effect the result of the test case. Isolating failure inducing difference between failure and a passing test case. It iteratively distribute the parameters in a binary way until the failing part is found. It can be used in any situation in which it effects the program execution.

Chances that the test fails depends on the chosen subset of failing test case parameters. If the subset is smaller then there is a lower chance that the failure-inducing partial combination is found, else the change is higher. It uses a minimizing algorithm to find failure inducing partial combinations of a test case by divide and conquer approach. It splits the total set of changes binarily, tests each half to find the failure inducing combination and continues to split until the inducing combination has been identified.

If the failure inducing partial combination is not found by iteratively checking and dividing the set of changes then the algorithm uses the complements of changes in a combination for further steps and increase the granularity at each step. If for the larger combinations no failure is found it returns unresolved state until all pair of combinations are tested. The problem with this approach is that the number of tests required are increased with the number of inputs. It simplifies and isolates the failure inducing inputs by automated test procedure [17] and can be used any time when a test case fails.

4.6 FCI

In combinatorial testing a parametric model is used to test the system and identify FCIs (**f**ailure-inducing **c**ombinatorial **i**nteractions). It is a black-box testing for a system that have different components which interact with each other. In CT a set of test cases is generated to achieve different kind of coverage i.e pair-wise testing. These test cases are then executed to identify failed test cases and then their cause of failure is identified by the combination of parameters.

The interaction of these parameters is called failure-inducing combinatorial interactions and are the namesake of this algorithm proposed by Zhang, Feifei and Zhiqiang [18]. This algorithm takes input from the result of generated test cases and produces the solutions for each combination, which becomes the criterion to measure the precision of the solution. FCI uses constraint satisfaction problem(CSP) or satisfiability to solve identification problem [18].

A definition of FCI proposed by Jian, Feifei, and Zhiqiang [18] is as " A faulty combinatorial interaction (FCI) is a CI such that all possible test cases containing it will fail" whereas CI is defined as "A combinatorial interaction (CI) is a vector of length k, which assigns t parameters to specific values, leaving the rest $k`-'t$ parameter values undetermined (undetermined values are denoted by '$-'$). Here t is the size of the CI."

Suppose we have an FCI P1, and P2 contains P1. Then all test cases containing P2 will contain P1, and all of these test cases will fail. Thus P2 is an FCI. For example, suppose (1,-,2,-) is an FCI, then (1,1,2,-) and (1,-,2,3) are also FCIs. [18]

4.7 AIFL

Liang, Changhai and Baowen [15] introduced AIFL (named this way by Wang, Xu et al. [16] in IterAIFL approach and standing for "**A**daptive **i**nteraction **f**ault **l**ocalization") is a rather simple, but adaptive failure localization approach. It assumes that only one fault is present in the SUT.

The algorithm is given a test suite which is run. Then only the parameter combinations present in failed tests of that suite are isolated. In an attempt to reduce that set of possibly failure-inducing combinations, a new set of n test cases, where n is the number of parameters of the test case, is generated for each failed test. Every one of these n test cases has one of its parameter values changed to a new value. Combinations that are present in passing tests of this new test case set are not inducing and therefore removed from the set of possibly inducing combinations.

As the approach generates n new test cases for each failed test, the amount of test cases can become very large very quickly. Additionally, as only parameter combinations which violate the necessary condition to be inducing are removed the likelihood that non-inducing combination are also present in the answer is relatively high. But the resulting set will be a subset of the set of all inducing combinations which can be found with the original test set. Therefore the AIFL result is complete.

4.8 IterAIFL

Wang, Xu et al. [16] introduced IterAIFL which is a continuation of AIFL. Specifically it employs iteration to improve on the precision of the result of AIFL. Therefore is, just as AIFL, adaptive, complete and in addition, other than AIFL, iterative and is able to handle multiple faults in the SUT.

The algorithm initially executes AIFL is described in section 4.7 and then iteratively generates more test cases and removes suspicious combinations found to be not inducing. This iteration can be terminated prematurely when the size of the suspicious combination set reaches a given threshold. Otherwise the algorithm will terminate when the last iteration did not reduce the set of suspicious combinations, at which point the result set, except for some cases in which parameters and their values overlap, 100% precise. Furthermore is the result always complete.

There are two test case generation approaches defined. The first one called IterAIFL$_1$ mutates previously failed test cases by changing m parameter values, where m is the mutation strength which is increased by one for every iteration up to one less than the number of parameters. To given an example consider the test case (IE, Linux, ISDN, Access). The newly generated test cases for the 2-th iteration, so using the mutation strength 2) would be the following:

$$t'_1 = \text{(Opera, Windows, ISDN, Access)}$$
$$t'_2 = \text{(Opera, Linux, Modem, Access)}$$
$$t'_3 = \text{(Opera, Linux, ISDN, DB/2)}$$
$$t'_4 = \text{(IE, Windows, Modem, Access)}$$
$$t'_5 = \text{(IE, Windows, ISDN, DB/2)}$$
$$t'_6 = \text{(IE, Linux, Modem, DB/2)}$$

The second one called IterAIFL$_2$ simply always generates new test cases based on the failed set of the previous iteration with mutation strength 1.

None of the two test case generation methods is superior to the other in a general case. The step of new test case generation however is the key to reduce the cost of failure diagnosis, but the effectiveness of any chosen approach is very dependent on the specific software under test and it may be hard to predict which one will perform better.

4.9 LDA

Colbourn, Charles and McClary [4] introduced LDA (**Lo**cating and **D**etecting **A**rrays for Interaction Faults) which is a non-adaptive approach to failure diagnosis, meaning that all test cases a generated beforehand and executed in parallel. Proposed are (d, t)-locating arrays, which can locate failure-inducing t-way interactions under the assumption of a maximum of d in the SUT. So in general the number of faults in the SUT have to be know a priori or at least an upper bound has to be estimated. This is because how many different faults are to be detected does affect their size of the locating and detecting arrays and how they are constructed. Even though it has been shown that the growth of these arrays is asymptotic, and the basis of array based testing, covering arrays, have been researched extensively by Chateauneuf, Hartman et al. [2] [9], the actual construction of detecting arrays with fewest test is a challenging problem. However, there have been algorithms proposed, of which we will discuss the following two.

4.9.1 LDA(1,2)

LDA(1,2) introduced by Nagamoto et al. [12] is a specific implementation of the general LDA approach (cf. Section 4.9) and therefore non-adaptive. It allows for the detection of a maximum of one fault caused by a 2-way interaction of parameters. These very limiting requirements make this approach relatively unsuited for software testing, but might be applicable enough for more limited, component based systems as given as an example by the authors.

The basic idea of the approach is to separate all interactions into equivalence classes over equality of the set of failed tests for a given 2-way interaction (which is an equivalence relation) and construct a test suite such that each equivalence class consists of only one interaction. This is achieved by adding test cases which separate interactions from the other interactions in an equivalence classes.

4.9.2 LDA(d,t)

Konishi et al. [10] proposed algorithm to find minimum (d,t)-locating arrays using a SAT solver. Again the problem of a priori knowledge of the number of faults in the SUT persist for any practical use case.

5. DISCUSSION

In this chapter we will compare the different approaches regarding their assumptions, strength, weaknesses and other differences. We will give an overview over properties attributable to the approaches and highlight differences and similarities we deemed noteworthy.

As all our reviewed approaches are based in combinatorial testing, they are all black-box testing methods. 7 out of 9 of the reviewed approaches fall into the *test generation* category (defined in Section 4) and 6 out of those 7 are adaptive. This shows a clear bias towards adaptive methods. A cause for this bias might be that adaptive approaches are easier to develop in the sense that less "thinking ahead" is necessary as one can rely on the results of the execution of the previous tests and only take them into account, while in a non-adaptive or a result analysing approach one has to make sure that all additional test cases, necessary to cover all desired combinations, are generated without such knowledge.

Almost all approaches require that the chosen t, maximum size of failure-inducing interactions, is chosen, or rather predicted, correctly. Some approaches, like AIFL and LDA(1,2), only support pair-wise testing ($t = 2$) which is too harsh a limitation for a lot of more complex systems. While others, like BEN and LDA(d,t), become inefficient when t is chosen too big. How big the interactions are one wants to considers is therefore a very important matter that can have significant impact on what algorithm should be chosen for the task.

As mentioned before in Section 4, the main distinction between test generating approaches and ones that analyse the results (see Figure 1) is important because the generation of new test cases is a significant expense compared to a mere analysis of the results. But performing failure diagnosis based on only the given results of a small test suite can be very hard or even impossible to do accurately. That is where test generating (and especially adaptive) approaches can still be very accurate. With sufficiently large test suites though result analysing approaches should have a significant performance advantage.

The differences in adaptive and non-adaptive approaches result in very different strengths and applicability. For example, the test execution in non-adaptive tests can theoretically be highly and easily parallelized due to the fact that all test cases, which will be executed by the algorithm, are calculated before any test is executed, but also have to generate a significant amount of test cases in the first place. Other in adaptive approaches where newly generated test are based on the result of previously executed tests. But the big downside of non-adaptive testing as of now are the significant restrictions put on the SUTs. LDA(1,2) for example can only handle systems that at most have one fault which is caused by the interaction of only two parameters, which is definitely not enough for a test suite for a modern software. So this approach is limited to rather simple component based systems.

AIFL and LDA in general are rather limited approaches because of their very basic nature, and a not very promising (at least for complex software testing) approaches respectively. AIFL has been improved and arguably superseded by IterAIFL by being able to detect more than one fault in the SUT and achieve a 100% accurate result in more circumstances by basically only adding additional iterations of the same method. LDA as the only non-adaptive approach trying to solve the problem of failure diagnosis with arrays, and even though two algorithms for the generation of such localizing arrays have been proposed, has the very big drawback of having to know the amount of faults that want to be detected a priori. It is fairly limited in it's application, especially for larger systems due to the high amount of test cases needed and limitations imposed on it.

FCI a uses combinatorial test suit as input and executes test cases with input parameters. To identify the failure-inducing parameters FCI uses constraint solving and optimization techniques on pair wise testing while FROG uses regression coefficient to identify the failure-inducing interactions. FCI uses iterative approach to reduce the number of test cases by changing the value of each parameter in failure-inducing interaction while FROG uses input of all failure-inducing interactions and excludes the interaction which are less suspicious in each iteration.

FIC and FIC_BS iteratively execute the failure inducing input parameters on test cases by changing the value of each parameter in each iteration and identify the failure-inducing parameter if the test case passes on a given combination. FIC takes all inputs in failure inducing combination and iteratively runs the test case by changing value of each input. However, FIC_BS uses binary search by dividing combination of these input parameters into two halves and executes the iteration on each half by changing value of each parameter.

Correctness of FIC and FIC_BS depends upon some assumptions which are briefly discussed. In some cases the status of test case is undetermined and may produce unresolved result. The outcome of executing a test case on the SUT is either pass or fail [19]. It might happen that some test cases that matches the failure-inducing interaction may pass due to some error dominations over others but in FIC and FIC_BS each test case that matches the failure-inducing interaction must fail. All parameters are independent of each other [19]. All failure-inducing interactions that appear during FIC are failure-inducing interaction of failing test cases.

DDmin works very similar to FIC. However, instead of finding failure-inducing parameter in each iteration it first reduces the number of test cases by changing the value of parameter in each iteration and determines the failure-inducing parameters based on result of failed test cases. It uses the set of already tested parameters in execution and then increases granularity for further test cases until the failure-inducing combination is found. Unlike FIC, DDmin returns *unresolved* if the status of test case is undetermined. DDmin only focuses on changeable circumstances and assumes that by changing the parameters will cause different program behaviour.

BEN and FROG impose the least restrictions on the SUT or test cases of all approaches from the test generation category. They also are also notably the only two approach that rank their results by each combinations suspiciousness. Not only does this help a developer with debugging by allowing him to focus on the most likely failure-inducing combinations, but it also enables the algorithms to omit some of the less likely to be inducing combinations and possibly significantly reduce the amount of test cases that need to be considered or additionally generated. When suspicious combinations are omitted it of course can no longer be guaranteed that the resulting set of combinations is complete. But with limited testing resources being able to achieve a higher precision cheaper might be more interesting. IterAIFL can be terminated early as well, but as it lacks the ranking of BEN and FROG, and therefore the pre-filtering by suspiciousness, it is more likely to be less accurate, but still complete.

FROG uses logistic regression with a configurable threshold to identify the suspiciousness of interactions. Because of the nature a logistic regression the threshold for the FROG algorithm is not intuitive and can at least theoretically never result in a complete set. BEN can, not just in practice, be configured to give a complete result. This configurability of both FROG and BEN make them very useful for practical use-cases, as their completeness can be traded for resource consumption, i.e. higher precision can be reached cheaper.

Along with the fact that FROG imposes minimal restrictions as well, both FROG and BEN are the most practically usable algorithms, despite their very different approaches. As none of the papers provide comparable benchmarking data, and benchmarking is not the scope of our study, we can not determine which of the two performs better.

6. CONCLUSION

Our literature review yielded a total of 9 different algorithmic approaches for failure diagnosis in combinatorial testing. The efficiency of these algorithms depends upon the selection of suitable algorithms on the basis of SUTs nature. We have compared these approaches because it is an important step to identify the suitability of an approach for any given SUT.

Main characteristics of these algorithms is that they either use tests generation or result analysis for failure diagnosis. Algorithms from the tests generation category uses adaptive/non-adaptive approaches for further test case generations which works very good for large number of combinations as we have larger test suite for failure diagnosis in larger programs but its too expensive to run this approach for smaller number of input combinations. Result analysis on the other hand is efficient for smaller combinations but much more expensive for larger combinations.

Most algorithms employ adaptive and iterative, considering a very wide and complete set of suspicious combinations and iteratively removing ones which are not (or not enough) suspicious, approaches. With larger and more complex SUTs BEN and FROG provide good approaches because of their ranking techniques focusing on only combinations which are likely to be failure-inducing and thereby reducing the amount of suspicious combinations considered, cutting down on resource consumption, which is especially valuable for a practical use case.

The non-ranking approaches, namely IterAIFL, FIC(_BS), DDmin, LDA, FCI are less interesting for practical usage. Their results are generally complete, but this makes the gained precision too expensive in a resource constricted context. BEN and FROG, the algorithms that include ranking, do a better job of achieving higher precision whilst using less resources due to their ranking nature. Thus ranking algorithms generally

Non-adaptive approaches to failure diagnosis do not appear to be very promising, because they impose very restrictive assumptions upon the SUT which makes them unfeasible for most more complex SUTs.

In the future we assume that especially adaptive approaches will be continued to be pursued, although FROG represents a very successful approach from the category of non-adaptive approaches. But adaptive approaches are easier to develop and implement then non-adaptive ones.

To expand on our work we propose the inclusion of possibly newer approaches, analysis and categorization of different types of SUTs and systematic benchmarking of the algorithms for these different types of SUTs.

7. REFERENCES

[1] K. L. Burr and W. Young. Combinatorial test techniques : Table-based automation , test generation and code coverage. 1998.

[2] M. A. Chateauneuf, C. J. Colbourn, and D. L. Kreher. Covering arrays of strength three. *Designs, Codes and Cryptography*, 16(3):235–242, May 1999.

[3] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The aetg system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, July 1997.

[4] C. J. Colbourn and D. W. McClary. Locating and detecting arrays for interaction faults. *Journal of Combinatorial Optimization*, 15(1):17–48, Jan 2008.

[5] L. S. Ghandehari, J. Chandrasekaran, Y. Lei, R. Kacker, and D. R. Kuhn. Ben: A combinatorial testing-based fault localization tool. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–4, April 2015.

[6] L. S. Ghandehari, Y. Lei, R. Kacker, D. R. Kuhn, D. Kung, and T. Xie. A combinatorial testing-based approach to fault localization. *IEEE Transactions on Software Engineering*, page 1, 2014.

[7] L. S. G. Ghandehari, Y. Lei, T. Xie, R. Kuhn, and R. Kacker. Identifying failure-inducing combinations in a combinatorial test set. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 370–379, April 2012.

[8] J. D. Hagar, T. L. Wissink, D. R. Kuhn, and R. N. Kacker. Introducing combinatorial testing in a large organization. *Computer*, 48(4):64–72, Apr 2015.

[9] A. Hartman and L. Raskin. Problems and algorithms for covering arrays. *Discrete Mathematics*, 284(1):149 – 156, 2004. Special Issue in Honour of Curt Lindner on His 65th Birthday.

[10] T. Konishi, H. Kojima, H. Nakagawa, and T. Tsuchiya. Finding minimum locating arrays using a sat solver. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 276–277, March 2017.

[11] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering*, 30(6):418–421, June 2004.

[12] T. Nagamoto, H. Kojima, H. Nakagawa, and T. Tsuchiya. Locating a faulty interaction in pair-wise testing. In *2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing*, pages 155–156, Nov 2014.

[13] C. Nie and H. Leung. A survey of combinatorial testing. *ACM Comput. Surv.*, 43(2):11:1–11:29, Feb. 2011.

[14] K. Nishiura, E. Choi, and O. Mizuno. Improving faulty interaction localization using logistic regression. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 138–149, July 2017.

[15] L. Shi, C. Nie, and B. Xu. A software debugging method based on pairwise testing. In V. S. Sunderam, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, editors, *Computational Science – ICCS 2005*, pages 1088–1091, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[16] Z. Wang, B. Xu, L. Chen, and L. Xu. Adaptive interaction fault location based on combinatorial testing. In *2010 10th International Conference on Quality Software*, pages 495–502, July 2010.

[17] A. Zeller and R. Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 28(2):183–200, Feb 2002.

[18] J. Zhang, F. Ma, and Z. Zhang. Faulty interaction identification via constraint solving and optimization. In A. Cimatti and R. Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, pages 186–199, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[19] Z. Zhang and J. Zhang. Characterizing failure-causing parameter interactions by adaptive testing. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ISSTA '11, pages 331–341, New York, NY, USA, 2011. ACM.

Classification and Comparison of Approaches to Software Architecture Reconstruction

Robert Barisic
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
robert.barisic@rwth-aachen.de

Josef Hoppe
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
josef.hoppe@rwth-aachen.de

ABSTRACT

Documentation of software architecture is often missing or outdated in software projects. To counter this, many approaches have been developed and improved, utilizing different artifacts and methods to gather information about a given software system. These approaches produce different views on the systems, are designed to work on different sizes of software, and have different intended uses. When generating those views, approaches utilize different ways of processing their input data. While many approaches focus on clustering units like methods, classes, or files into logical subsystems, others employ a dynamic analysis on an object- or even service-level.

Ideally, all artifacts used for designing a software architecture beforehand could be recovered. The approaches considered in this paper recover the logical structure of methods, classes and files; dynamic views (like UML Activity Diagrams) detailing the run-time interaction of software components and dependencies between services based on network communications.

Additionally, the conformance of a piece of software to its previously defined architecture can be monitored to detect and correct deviations or update the architecture accordingly.

This paper classifies the methodology, used and generated artifacts, and intended application of software architecture reconstruction approaches. Additionally, we provide a comparison between the approaches, comparing focus and methodology as well as appropriate scale and accuracy.

In comparing the approaches, we found non-clustering approaches are usable or close to usable in a production environment, while pointing out challenges to overcome in order to create viable clustering techniques in the future.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management—*productivity, programming teams, software configuration management*

Keywords

Software Architecture Recovery, Software Architecture, Documentation, Dependencies, Quality Assurance

1. INTRODUCTION

A wide variety of software architecture recovery approaches has been developed for many different architectures and using different methods.

The most researched approach is to analyze the software's source code and extract a graph from that. This graph can then be clustered to receive a logical partition of source files. Graph clustering is a graph theory problem that precedes software architecture recovery. The largest differences between the static analysis approaches are the way of obtaining edges and weightings on the graph: While most utilize dependencies, others compare the similarity of keywords in a file's contents. From now on, we will refer to these as *Graph-based clustering approaches*.

Other approaches use runtime information (or *dynamic analysis*) in addition to the source code to extract different perspectives of the software. Dynamic analysis allows for the recovery of dynamic views on the software system, for example UML activity or sequence diagrams.

While different, these are all in stages of research. Later, we will refer to them as "research prototype", meaning the program requires some work to set up, but does produce results; or "algorithm", meaning the program cannot be run on actual sources but needs additional work such as extracting the structure first.

Sonargraph is the only commercially available product in our comparison. It helps its user to refine the recovered base structure, but is incapable of extracting an abstract representation of the architecture by itself.

1.1 Definitions

Architect. Software Architect working on a software from which he wants to recover the architecture.

1.2 Paper Structure

First, we will shortly present all sources used by any of our considered approaches to give you an understanding of the sources available to reconstruct the architecture of a software system.

Following that, we discuss the general methods employed by the approaches we considered. We only discuss methods that work on the information. For example, generating a diagram directly from the base information will not be layed out here because it can be described quickly enough to fit into the approach overview.

In the overview section, we describe the concept of each approach, including the methods used and any specific steps or concepts.

For our comparison, we considered 5 different aspects. Although not all approaches are directly comparable in every aspect, we present the results of the comparison in a qualitative Kiviat Diagram.

1.3 Related Work

Comparisons between some or all of the graph-based clustering approaches have already been done, including comparisons regarding accuracy and performance. We included [13] into our comparison because it is the most recent comparison that compares all graph-based clustering algorithms. There are earlier comparisons [4, 9, 12] which we didn't consider for the results because some of the approaches are under continuous development, making earlier comparisons obsolete.

2. ARTIFACTS & SOURCES AVAILABLE AS INPUT

All software architecture reconstruction approaches use the source code as input. But there is various information that can be extracted from it and the reconstruction approaches differ in what they make use of. Next to the static source code, there are also other sources of information. We classified them into static or compile-time sources and dynamic or runtime sources.

2.1 Compile-time / static Sources

Compile-time or static sources of information can be directly extracted from the source code or other documents of the software project.

2.1.1 System Documentation

The originally intended architecture can be used to detect deviations from it that occur during a software system's lifecycle. When the current architecture deviates from this baseline, this can be automatically detected in order to notify the architect [8]. This can be useful to combat architecture deterioration or keep the documentation updated when necessary.

2.1.2 Code Dependencies

Code dependencies are a very common source of information about the structure of a given system. There are two kinds of dependencies: file dependencies and symbol dependencies[13]. File dependencies are based on `include`, `import` or respective statements while symbol dependencies inspect the code and check for usage of symbols in other files. This can yield a more complete and meaningful graph because some file dependencies are not used while some symbol dependencies (e.g. fully qualified names in Java) are not detected by file dependency analysis [13].

The analysis of code dependencies yields a dependency graph that can be further analyzed [10, 19].

2.1.3 File Structure and Filenames

Often, folders represent logical groups (for example packages in Java). These groups can relate to subsystems, but grouping by other factors (such as a collection of header files) exists and may be more helpful than folder-based grouping[19].

Folder and file names can also be utilized to name identified subsystems, although extracting the best name can be difficult [19].

Still, the file system graph directly represents a logical grouping declared by the developers of the system and can for example be used to supplement other structural information [19].

2.1.4 Comments and Identifiers

Programmers in general try to use expressive names for classes, functions, attributes and parameters. Comments often contain useful information too. The information obtained from the comments and this identifiers can be used for clustering. [5]

2.1.5 Keywords

To understand the intended function of a given piece of code, keywords and combinations of keywords can be used. By comparing frequencies of different keywords, topics of a given file (or any piece of text) are retrievable[10].

This information can be used later, when grouping files (grouping by topic results in more meaningful groups than simply grouping by dependencies according to [10]) and to determine a better description of a module.

With further analysis of the file structure, keywords can be divided into zones, e.g. function names, comments, etc. [5, 6].

2.1.6 Static configuration

For dynamic analysis, retrieving the static configuration of an application can be important: For example, when analysing network traffic, identification of services in the network is necessary to extract a relation [11]. Generally, this info is used as a basis for dynamic analysis.

2.2 Dynamic / Runtime Sources

Next to the static sources of information, there is also a way to extract information during the runtime of a software system.

2.2.1 Dynamic dependencies / calls

In order to get more meaningful information about the cohesion and coupling between software entities, it can be very useful to include the information about how often for example functions are called during the runtime of the system [22].

Additionally, the interconnection and interaction of software components can be recovered in this way [7].

2.2.2 Docker configuration

Docker is a tool for packing applications and services into containers (similar to virtual machines, but more lightweight) and running them in composition. From the configuration, IP-Adresses, Ports, permanent Storage, and other containers (services) reachable per network can be extracted.

The Docker configuration is particularly useful for systems consisting of multiple small services, because connections between different services are more important than with monolithic software [11].

This information can be used to attribute recorded network traffic to the correct container.

2.2.3 *Network Communication*

Analysis of network communication ensures that, given a sufficient test scenario, no links between different services can be missed. Especially when using a service discovery service, dependencies between services cannot be recovered using static analysis [11].

2.3 Architect Knowledge

Correctly identifying subsystems has not been reliably done automatically and can be essential for further analyzing the whole system [11, 19].

Therefore, architect input at critical points during the process can be extremely useful. Architects often have an idea of the whole system or can at least identify useful labels and central concepts of complex systems.

This intelligent input can be used both for more understandable labelling of finished views [19] as well as fitting the partially analyzed system into a good pattern or identifying central elements of a pattern, making further analysis more accurate [11].

3. RECONSTRUCTION METHODS

In the following, we present all methods for reconstructing software architecture used in any considered approach. While no considered approach uses all of these, some do use multiple.

3.1 Graph Analysis

Analyzing the structure of a system as a graph has the advantage of being able to pick from the large number of algorithms developed for graphs.

A possible approach is the identification of different structural patterns in order to find software architecture design patterns. For example, one of the patterns searched for by ACDC [19] is the *Leaf collection pattern.* It is identified by a number of leafs only accessible via a single node and not interconnected. Using this collection of patterns, common design choices made by software architects can be recovered.

In an analysis of the whole software system, cluster analysis is often utilized. Cluster analysis searches for subgraphs with a high number of connections inside and a small number of connections to nodes outside the subgraph. These so called clusters represent logical subsystems. "Orphaned" nodes, i.e. nodes not belonging to any cluster, can then be added to the most suitable cluster. Different approaches use different parameters for clustering, for example desired or maximum cluster size [19]. Other metrics, such as keywords contained in the source files [10] or the file structure [19] can be utilized to extract clusters more related to the logical organization of the project.

A weighting of edges can be applied, thereby allowing for more meaningful clusters [17].

3.2 Hierarchical Clustering

Hierarchical clustering is used by many software architecture reconstruction approaches. It starts with many single entities and iteratively combines the two most similar entities with each other to one group.

Before entities can be clustered it has to be defined what is seen as an entity. For software systems an entity can be for example the source code files or functions. Next, the features of the entities must be considered. If for example functions are defined as the entities, their features could be what functions and global variables they use and which they don't use. If two functions use the same functions and global variables, these functions have to be similar to a certain extend. And what clustering is about, is to group such similar entities.

The basic approach of hierarchical clustering algorithms is to first calculate the feature vectors of all the single entities. Next the similarity between every pair of entities is determined using a predefined similarity measure. In the third step, the two most similar entities are grouped together and represent a new entity. Lastly the second and third step are repeated until the number of desired clusters is reached or there is only one cluster left [16].

3.3 Name extraction

Finding an expressive name improves the understanding of the resulting model. Thus, multiple methods have been proposed.

Keywords can be extracted from one or multiple files. Using their frequency, a topic can be determined via look-up in a table. Said topic often provides a meaningful title for a given cluster [17].

Alternatively, finding common substrings in file or method names (such as 'Car' in 'CarFactory', 'Car', 'CarModel', 'CarMake', etc.) often lead to suitable cluster names as well[19].

3.4 Network inspection

When the direct communications between networked components such as microservices have been discovered (see section 2.2.3 Sources / Network Communication), some indirect connections may still be masked. Service discovery services or other proxies forward communications between different microservices. This means that a direct (unanalyzed) extraction of network traffic will result to identify the actual endpoints of communication, incoming and outgoing requests of the service discovery service are compared, thus detecting the logical communication [11].

4. OVERVIEW OF APPROACHES

Before comparing and classifying the current approaches we considered, they are shortly presented and summarized in this section.

We considered 6 conceptually similar clustering approaches: ACDC [19], ARC [10], Bunch [14, 15], LIMBO [3], WCA [17], and ZBR [6], [5]. These approaches all extract a graph from the systems source code and then apply a clustering algorithm to that graph, yielding a logical structure of the given software system.

In contrast to those 6, we also considered three unique approaches: Aramis [2, 7, 8, 18], which creates views of runtime interactions; MicroART [11], which specializes in microservice architectures; and Sonargraph [1], a commercial product which assists manual software architecture recovery.

4.1 ACDC

ACDC is short for **A**lgorithm for **C**omprehension-**D**riven **C**lustering. The algorithm tries to decompose a given software system into clusters or subsystems, in order to improve the level of comprehension of the software system. This happens within two stages.

In the first stage the algorithm identifies a fundamental structure of subsystems. These subsystems are constructed working through different patterns, corresponding to common software architecture design patterns, in an order of precedence. It is ensured that the subsystems are named reasonably and don't exceed a defined size limit to maintain a certain clarity.

In the second stage, orphan adoption is used to assign the remaining resources to the subsystems created in stage one. This technique looks for the most appropriate subsystem to place a resource in. The traditional task of the orphan adoption was to handle the evolution of a system by assigning new components to an already existing subsystem, but according to the authors, it still produces accurate results for a large percentage of adopted nodes.

In total, ACDC aims to generate a model with meaningful names, consisting of clusters of manageable size and representing useful patterns in software architecture.

4.2 ARC

Architecture **R**ecovery using **C**oncerns is another clustering approach. The algorithm represents a software system as a set of documents and the concerns are the topics of these documents. These concerns are recovered using the statistical language model LDA. The input for the LDA is obtained by the source codes comments and identifiers, which act as the words of the documents. Combining the concerns and the structural information obtained from the source code, the algorithm decides, whether a software entity belongs to a cluster or not. Instead of clusters the algorithm uses the term bricks. Such a brick implements either a component or a connector. In order to classify them, the algorithm distinguish their concerns as application-independent and application-specific. Furthermore, it considers the usage of connector-implementing libraries, like the socket library and the bricks involvement in design patterns that provide interaction service as adapter, proxy or the observer design pattern. The result is a component-connector model of the systems architecture.

4.3 Bunch

Bunch is a tool which produces a clustering based on a systems source code. The tool starts with a such called module dependency graph, illustrating the relationships between the components in the source code. In this graph, bunch searches for an optimal partition according to the modularisation measurement (see below). Therefore, they introduced a measurement for Intra-Connectivity and Inter-Connectivity.

Intra-Connectivity is a measurement for the connectivity within a cluster and Inter-Connectivity is a measurement for the connectivity between clusters. The modularisation quality function combines these two measurements in a form that results in higher value if the clustering has high cohesion within the clusters and a low coupling between the clusters.

In order to get the best clustering this modularisation quality function needs to be maximized. Therefore Bunch uses a hill-climbing algorithm. Starting with a random partition, the algorithm looks for the neighbouring partitions of it. If a neighbouring partition with a higher modularisation quality is found, this neighbouring partition is set as the new partition and the step is repeated until there is no neighbouring partition of the current partition with a higher modularisation quality. Bunch also allows the architect to supply it with predefined clusters in order to include their knowledge to improve accuracy, making Bunch an optionally semiautomatic assisted recovery tool.

4.4 LIMBO

LIMBO is an algorithm that tries to minimize the information loss while performing a clustering of a software system. It tries to include a large set of input sources and considers structural information as well as non-structural information.

LIMBO is short for sca**L**able **I**nfor**M**ation **BO**ttleneck and uses the **A**gglomerative **I**nformation **B**ottleneck algorithm which is designed to find a clustering of small cardinality and large information content. As this algorithm doesn't perform well on large scaled systems, LIMBO first creates a set of such called Summary Artifacts in order to compromise the size for the AIB algorithm. AIB is then employed on this set of Summary Artifacts.

4.5 WCA

The **W**eighted **C**ombined **A**lgorithm is another hierarchical clustering approach. It extracts a similarity graph on the level of routines and global variables. Contrary to other techniques like ACDC, the edges don't represent dependencies between entities but rather how many common dependencies they share. On this graph, WCA searches for clusters using a weighted similarity algorithm, introducing a new distance measure.

4.6 ZBR

Zone **B**ased **R**ecovery (the name was not given by the original authors but by Garcia et al [9]) is a clustering approach utilizing lexical information of class files first introduced in [6]. Specifically, all words (except common english words and Java keywords), their frequency, and their corresponding "zones" (Javadoc, comments, signatures, variables) in a class are extracted into a vector of word frequencies, weighted by their respective zones (although this weighting is dynamically calculated on the given set of files, for example a word contained in a class name will usually have a higher significance than a word contained in a comment).

These frequencies are used to compute the similarity of classes. Using those similarities, clusters are formed around randomly selected nodes using the *K-Medioids* algorithm. If the resulting clustering is not satisfactory (measured by the amount of clusters that are too small), a new set of nodes is selected for clustering, until it is either satisfactory or a maximum number of iterations is reached.

The approach was improved one year later. The clustering algorithm was exchanged for a hierarchical clustering algorithm in order to get rid of the randomness which the *K-Medioids* algorithm contains. In addition the zones were extended to the six zones: Class names, attribute names, function names, parameter names and source code statements [5].

4.7 Aramis

Aramis is a dynamic analysis approach that focuses on continuous evaluation of the software during its lifecycle in order to detect any deviations from the intended architecture as soon as possible. In order to do that, Aramis utilizes a set of rules, specified by the architect, the software system has to adhere to.

Aramis employs a modular analysis approach, where the data gathered from a software system is converted into a standard format which is then analyzed. By performing the dynamic analysis using multiple use-cases, a dynamic view on each of those use-cases can be extracted, thereby also verifying the adherence to the software standard.

During the evolution of the software system, the design rules need to be kept updated. Aramis simplifies this by instantaneously alerting the architect of any deviation.

4.8 MicroART

MicroART is a specialized software architecture reconstruction approach for microservice based architectures. It provides an overview of the architecture at the service level.

MicroART searches the source code of the services for *Docker* configuration: `dockerfile` is a script-like file with instructions for Docker to create a container which, among other information, defines the ports a service uses. `docker-compose.yml` defines which services exist in a system and, to some extent, how they are connected.

Because many microservice based architectures use a service discovery service, static analysis isn't always sufficient to recover the complete architecture. Therefore, MicroART analyzes runtime artifacts of the system: By combining network traffic and IP adresses retrieved from the Docker runtime state, the precise communication between services is discovered.

4.9 Sonargraph

In contrast to the other approaches, Sonargraph is a commercial product, already in use in the industry. According to its website, the architect recovers the structure with Sonargraph providing assistance such as detecting common patterns like cyclic dependencies. Sonargraph furthermore supports architects when evaluating software systems, such as approximating technical debt based on best practices.

5. CLASSIFICATION & COMPARISON

The developed approaches are in various states of usability. While Sonargraph is already a commercial product, many others are research prototypes and require extensive knowledge to use them or thorough preprocessing of information. Table 1 gives a comparison between approaches.

The *Status* column in the table refers to the Status as detailed in the Introduction to this paper. Except for Sonargraph, all approaches provide a single algorithm, while Sonargraph is a commercial suite of different extraction algorithms and tools.

Scope refers to a conceptual limit of the scope, meaning no entry in the Scope column corresponds to the approach (potentially) being used on all software projects, although the current implementation may only support one or a few languages or adhere to other constraints. For MicroART, the Scope "Microservice Architectures" means that conceptually, MicroART can only ever be used on this type of software architecture.

Concept is a short summary of the idea a particular approach uses to extract a logical architecture. For more info, see the corresponding section for each approach.

Static and *Dynamic* refer to the utilization of static (meaning any file, including source code and configuration) and dynamic (meaning network traffic, function calls, etc.) sources for reconstructing the software architecture.

Additionally, different foci exist, as detailed below.

5.1 Foci and Base Concepts

ACDC, ARC, Bunch, LIMBO, WCA, and ZBR work on the same overall principle: Extracting a graph from source code which can then be analyzed and clustered to recover the logical software architecture. Therefore, these 6 approaches can be compared systematically and automatically. In contrast, Aramis, MicroART, and Sonargraph all have unique approaches.

Aramis utilizes a model extracted by another recovery approach in order to generate views on interactions between software components and monitor the adherence of a system to its design specifications.

MicroART exclusively analyzes microservice architectures. It detects dependencies between microservices based on their network communication, but doesn't apply a further grouping or clustering.

Sonargraph, the only commercial product in our comparison, extracts dependencies from code, but relies on the architect to retrieve the architecture, only assisting him in doing so.

Among the 6 "Graph Clustering Approaches", further classification is required: ACDC is the only considered approach that uses a software architecture specific clustering algorithm, while the other 5 approaches differ in the extraction of the base graph and the chosen clustering algorithm.

ACDC specifically searches the (unweighted) dependency graph of the system for a number of patterns that correspond to common software architecture design patterns. In contrast, Bunch applies a non-specific clustering algorithm to the dependency graph, allowing for restrictions by the architect in order to improve the resulting clustering.

LIMBO condenses the base information into "Summary Artifacts" and then applies a badly scaling general algorithm, taking into account many pieces of information.

The remaining 3 algorithms all work on weighted graphs.

WCA works on the number common dependencies between methods as the basis for the weighting for each edge of the base graph.

ARC and ZBR both utilize the words contained in each file to calculate the similarities. ARC identifies topics and their relevance to the file and compares files based on those to obtain their similarity (weighting) and clusters based on that. ZBR splits each source code file into 6 logical zones: class names, attribute names, function names, parameter names, comments and code statements. Similarities are based on geometrical interpretation of the ratio of meaningful (non-fillers and non-Java keywords) common words between a single document and all documents.

WCA as well as ZBR were compared to unweighted variants of their approach. In both cases the weighted approaches performed better, what shows that including weights in general can be a way to improve the accuracy of the algorithm.

5.2 Accuracy of the recovered architecture

Evaluating the accuracy of a given approach is a complex task. Firstly, even if the as-is or ground-truth architecture is available, a comparison is not trivial: The similarity of the resulting clustering and the ground-truth architecture can be measured via *MoJoFM* [20] or by comparing the similarity of clusters [13], but valuable metadata such as the quality

Name	Status	Scope	Concept	Static	Dynamic
ACDC	Algorithm		Pattern-based Clustering	✓	✗
ARC	Research Prototype		Content Similarity based Clustering	✓	✗
LIMBO	Algorithm		Information-theroetic clustering	✓	✗
WCA	Research Prototype		Weighted Hierarchical Clustering	✓	✗
Aramis	Research Prototype		Continuous Updates, Dynamic Views	✓	✓
MicroART	Research Prototype	Microservice Architectures	Single Pattern	✓	✓
Sonargraph	Commercial		Assisted Manual Recovery	✓	✗

Table 1: Framework Overview

of extracted names cannot be evaluated automatically.

Additionally, if the general architecture of a system is known, related patterns can be extracted more easily, making an approach specializing in a single pattern (such as MicroART) more accurate for that specific purpose, but less accurate or even totally unfit for other systems. This makes direct comparisons of MicroART with most other tools impossible.

Lutellier et al [13] compared ARC, ACDC, WCA, LIMBO, Bunch (without architect input), and ZBR. They ran them on 5 different pieces of software and different kinds of dependencies, yielding widely different results. These results vary depending on the method used to compute dependencies. The authors used different kinds of dependencies (include and symbol dependencies, among others), with different software systems being recovered more accurately on one dependency kind than another, but there is no overall best kind of dependency.

When measuring using MoJoFM (a common graph clustering similarity measure; compared to clustering provided or verified by each software's developers), ACDC and ARC were the overall best algorithms [1], although even their best MoJoFM similarities were 78%[2] (ACDC) and 56%[3] (ARC). Overall, their results show a big difference in accuracy based on the software system, with algorithms performing differently for different systems: When evaluating ArchStudio, ACDC scored the overall best similarity of 78%[2], while LIMBO only scored 25%[2]. While evaluating Chromium however, LIMBO outperformed ACDC at 79%[2] to 73%[2].

Overall, Bunch is the third most accurate clustering approach evaluated by Lutellier et al., followed by ZBR, WCA, and LIMBO.

Another important result from this study is that the MoJoFM results vary a lot from system to system. ACDC scored the best MoJoFM value for ArchStudio with 78%[2] and performed worst for Hadoop with 24%[4]. ARC has a range from 24%[3] to 56%[3] and the other approaches vary in their performance, too. This shows that the accuracy of automated recovery approaches are unpredictable and that they are therefore not yet reliable when considering real projects.

Using a different technique for measuring the accuracy of the recovered architecture, *Cluster-to-Cluster Analysis*, Garcia et al. still found ARC and ACDC to be the two most accurate approaches, but Bunch now fell behind significantly, on average only outperforming LIMBO.

[1]average over all dependency types
[2]Best of all dependency types
[3]Not based on dependencies
[4]Worst of all dependency types

Overall[5], ARC and ACDC are the most accurate[6] clustering approa-
ches. ACDC is also the only evaluated algorithm to use pattern-based clustering, indicating that understanding typical patterns and using that knowledge when clustering leads to more accurate architectures.

MicroART cannot be compared to the other approaches because of its widely differing scope. Judged by its method, MicroART should give an extremely accurate dependency graph if all possible interactions between services occur during its runtime analysis. For the service discovery service pattern, MicroART should therefore be the best tool possible with little architect input. On the other hand, MicroART only provides a top-level view on the different containers used and doesn't group any services at all. In contrast to all other approaches, MicroART also recovers developers working on projects, outlining responsibilities of developers and teams. Overall, MicroART is a useful tool for recovering top-level views on this specific architecture.

Aramis retrieves dynamic views on the software. These are accurate (the runtime interactions are visualized, but no further analysis is required), but only as complete as the test cases used during the runtime analysis.

Compared to MicroART, Aramis recovers a more complex structure, but requires more test cases: Every significant use case needs to be covered for Aramis whereas MicroART only requires every possible combination of communications between the microservices to be recorded. This of course leads to a less complex and meaningful structure than Aramis, but also makes recovery easier. Note however that MicroART and Aramis are still too different in concept to compare in more detail, even though both utilize runtime analysis.

Sonargraph, the only considered commercial (and only production-ready) approach, is not an automatic or architect-assisted clustering technique but rather a toolset providing developers with a file-structure based overview. Architects can automatically find certain undesirable patterns such as cyclic dependencies, but the more intricate structure has to be found by the architect. Because the structure is recovered by the architect, Sonargraph's accuracy is perfect (manually recovered architectures were used as baselines in all evaluations).

This leads to the conclusion, that some software structures are easier to recover (or at least, better techniques exist for them) than others. ACDCs high accuracy suggests that pattern-based clustering is a good approach [13], so an expansion of ACDC patterns may improve its accuracy with different architecture styles. Further research into why ARC's content-based clustering proved to be more accurate

[5]Average of all dependency types
[6]measured using MoJoFM

than the content-based clustering of ZBR can serve to improve the accuracy of content-based clustering.

5.3 Scalability and Calculation Time

When evaluating deviance from a planned architecture or changes between versions of the Architecture, the time an approach takes to compute the pattern is significant for its practical use. We don't consider Sonargraph for this comparison because it assists the developer in recovery, meaning that the time it takes to recover virtually equals the time the developer uses the Sonargraph Program. All other approaches require only little input from the developer compared to the whole recovery process.

All static approaches can be compared relatively easily. Lutellier et al [13] tested the scalability of ACDC, WCA, Bunch, LIMBO, ZBR, and ARC to different program sizes, using both file and symbol level dependencies for comparison. The following times were obtained on the server of Lutellier et al in multiple runs of the programs and may differ, but suffice for a relative comparison. They found that ACDC was the most scalable technique, taking only 70 to 120 minutes to compute the clustering for the Chromium Browser. The next fastest methods were WCA and ARC, both between 8 and 14 hours. Bunch took 20 to 24 hours to terminate in one variant (Bunch-NAHC) and didn't terminate at all in its other variant (Bunch-SAHC) before timing out after 24 hours. LIMBO was forcefully terminated after failing to do so for more than 4 days. Lastly, ZBR ran out of memory after using more than 40GB.

Aramis follows a different approach: By assisting in the manual updating of the architecture, it avoids long recalculations, thereby enabling almost real-time dynamic analysis [8]. The necessary input for Aramis only needs to be computed once and manually modified and updated during a software's life cycle, meaning a longer time to retrieve its automatically recovered component is feasible. Aramis' runtime analysis can take up some time in order to cover all relevant use-cases, but the duration depends on the amount of testing / analyzing done after each modification.

MicroART primarily needs time for the dynamic analysis. The time increases with the number of test cases that need to be performed so that all communications between services can be recovered. We were not able to find a study which provides a thorough analysis of MicroART in terms of performance, but judging from the description of the approach, the analysis should complete fast. This is mostly due to the fact that MicroART doesn't aim to cluster elements but rather retrieves a clarified dependency (communication) graph in architectures where communications are often masked.

5.4 Architect Input

Different approaches allow or require different amounts of input from the architect.

ACDC, ARC, LIMBO, WCA, and ZBR all work without any knowledge input from the architect. Bunch can work without such input, but accepts predefined clusters to improve its accuracy.

MicroART only relies on human input to identify the *service discovery service* which is central to the microservice architectures it exclusively works on. Additionally, the architect has to cover all use-cases with tests in order to retrieve all connections dynamically.

Aramis requires the architect to provide a formal description of architectural constraints in a XML-based domain-specific language. Additionally, all use-cases that need to be tested need to be executed.

Finally, Sonargraph relies most of all considered approaches on the input from software architects, solely supporting the architect in the recovery process.

5.5 Stability of the clustering algorithms

As software architecture reconstruction is applied during the evolution process of a software system in order to remain up to date with the current architecture, the used clustering algorithm needs to be stable. The algorithm is considered as stable, if slight changes to the input of the algorithm just have small impact on the resulting clustering. [16]

Different studies include the measure of stability into their comparison of software architecture reconstruction approaches [21], [16]. ACDC turned out to be very stable, whereas LIMBO, WCA and Bunch didn't perform well in this point. Especially Bunch couldn't perform well, because of the randomness included in the approach.

In the first version of ZBR [6], the clustering part of the approach, similar to the approach of Bunch, started with a random partition of entities and then improved that partition by reassigning the entities. Only one year later, the approach was adjusted and one of the changes was to use a hierarchical clustering algorithm in order to be more stable [5], underlining the importance of stability.

5.6 Summary

In Figure 1, we gathered all of the previously discussed results. Note however, that the diagram doesn't contain a scale for any measure because none can be accurately quantified to a numerical value.

The amount of architect input is not accurately quantifiable, because very different forms of input are required from identifying critical components over a variable and optional amount of pre-defined clusters to assisted recovery, the architect input is too heterogeneous to precisely quantify.

No single measurement for the accuracy has been found, with different measurements giving wildly different results. Therefore, we have based the scale on a direct comparison. Furthermore, only the 6 considered static clustering approaches can be meaningfully compared at all.

In terms of stability, we first ranked the 4 clustering approaches ACDC, Bunch, LIMBO and WCA as we found related studies. As we could not find enough studies comparing all of the 6 approaches and MicroART in terms of stability, we also tried to rank the approaches based on their procedure. Regarding stability, neither Sonargraph nor Aramis can be compared to the other approaches, as they do not simply perform a clustering of a system's architecture. However, we tried to rank them on this scale too. In order to do so we look at this scale as: "how good does the approach manage the problem of stability?". And as Sonargraph as well as Aramis are tools assisting the software architect during the evolution of a system, we ranked them that high.

For scalability, the 6 clustering approaches cannot be compared to the other approaches. Additionally, we only found a single measurement for scalability that doesn't allow a conclusive rating.

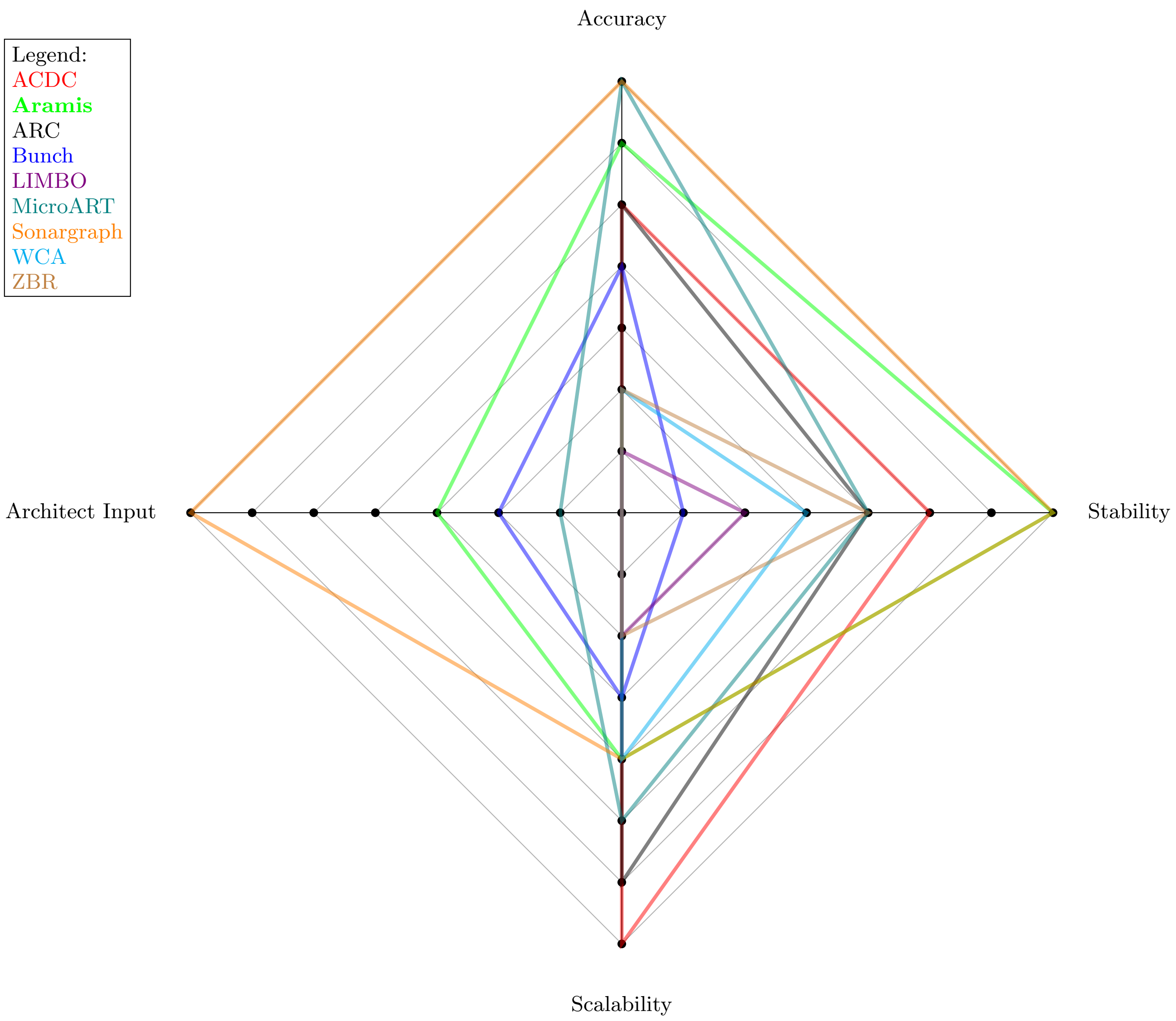


Figure 1: Kiviat Diagram of the quantifyable properties of the considered approaches

6. CONCLUSION AND FUTURE WORK

Automated recovery is not yet usable in actual projects because it lacks reliability on the recovered architecture. In order to develop better approaches, more common structural patterns can be discovered, further improving ACDC's accuracy. These patterns could be selected by architects prior to the recovery, which would be easier if the structural patterns could be directly linked to design philosophies.

In addition, a combination of the structural approach of ACDC and the document similarity approach of ARC could be an improvement. For example, ACDC relies on adding "orphaned" nodes to the most related (connected) cluster. ARC's high accuracy suggests that, especially in this instance, the document similarity could yield better results.

Since different software systems yield their most accurate results using different kinds of dependencies, it stands to reason that their structure differs fundamentally. This means that either fundamentally different views extracted from a software system can both be accurate, for example because they have different foci, or that the architecture of systems is actually different enough that none of the currently available approaches are capable of recovering both from the same kind of source. This questions needs to be researched further in order to improve current approaches or develop new, more accurate approaches.

The fact that ACDC, the second oldest considered automated recovery algorithm, is still among the most accurate shows that our understanding of what makes an approach accurate is still lacking. General clustering based on dependencies is, although a common approach, less accurate than ACDC's approach of clustering based on specific patterns present in software architecture or ARC's approach of lexicographical similarity. Ultimately, a clustering showing high cohesion and few connections to other clusters doesn't necessarily correspond to a logical and helpful view of the architecture.

Despite their usefulness for understanding the behaviour of software, dynamic views on the software are so far only recovered by Aramis, which recovers a low-level interaction analysis. We propose that more high-level diagrams that can be understood faster could be obtained by masking and grouping calls into logical groups, similar to structural clustering used in static analysis. However, without further research into this the practicability of this idea is unclear.

Specialized tools like MicroART yield accurate results already, although the work saved by automatically recovering a service-level dependency graph is considerably less than the work required to recover a full ground-truth architecture, enriched by clustering the modules. For now, relying on developers to understand the overall design choices of the system and biasing the recovery process seems to be the best way to improve accuracy, although the developers may be mistaken about the structure of their system. Sonargraph makes the most use of currently production-ready tools by providing assistance to architects when recovering, evaluating and changing software architecture.

Although measuring the performance of architect-assisted recovery methods is difficult, a study correlating invested architect time and architecture accuracy of different assisted reconstruction approaches, contrasted with manual recovery and fully automated recovery, would quantify the added value of assisted recovery approaches.

When evaluating the quality of recovered models, only similarities between the recovered clusterings and a baseline architecture have been compared. We propose that the value of the model for the architect isn't only determined by its similarity to the original on a graph level, but that certain disparities are more significant than others: Incorrectly assigning a core file of a cluster is worse than incorrectly assigning a file that serves as a connection between two modules. Furthermore, the cluster to which a file is incorrectly assigned plays a role as well: Assigning it to a similar or tightly connected cluster is less bad than assigning it to a totally different cluster. Many approaches produce hierarchical clusterings, which have also not been taken into account. Finally, no Metadata of the recovered model is taken into account: ACDC aims to recover meaningful names, but all research up until now has only evaluated the automatically comparable graph clustering.

7. REFERENCES

[1] Sonargraph architect. `https://www.hello2morrow.com/products/sonargraph/architect9`. Accessed: 2018-12-17.

[2] P. Alexander, A. Nicolaescu, and H. Lichter. Model-based evaluation and simulation of software architecture evolution. In *Proceedings of International Conference on Software Engineering Advances, ser. ICSEA, Barcelona*, pages 153–156, 2015.

[3] P. Andritsos and V. Tzerpos. Information-theoretic software clustering. *IEEE Transactions on Software Engineering*, (2):150–165, 2005.

[4] R. A. Bittencourt and D. D. S. Guerrero. Comparison of graph clustering algorithms for recovering software architecture module views. In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*, pages 251–254. IEEE, 2009.

[5] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello. Investigating the use of lexical information for software system clustering. In *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, pages 35–44. IEEE, 2011.

[6] A. Corazza, S. Di Martino, and G. Scanniello. A probabilistic based approach towards software system clustering. In *2010 14th European Conference on Software Maintenance and Reengineering*, pages 88–96. IEEE, 2010.

[7] A. Dragomir and H. Lichter. Model-based software architecture evolution and evaluation. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, volume 1, pages 697–700. IEEE, 2012.

[8] A. Dragomir and H. Lichter. Run-time monitoring and real-time visualization of software architectures. In *Software Engineering Conference (APSEC), 2013 20th Asia-Pacific*, volume 1, pages 396–403. IEEE, 2013.

[9] J. Garcia, I. Ivkovic, and N. Medvidovic. A comparative analysis of software architecture recovery techniques. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, pages 486–496. IEEE Press, 2013.

[10] J. Garcia, D. Popescu, C. Mattmann, N. Medvidovic, and Y. Cai. Enhancing architectural recovery using concerns. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 552–555. IEEE, 2011.

[11] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, and A. Di Salle. Microart: A software architecture recovery tool for maintaining microservice-based systems. In *IEEE International Conference on Software Architecture (ICSA)*, 2017.

[12] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidovic, and R. Kroeger. Comparing software architecture recovery techniques using accurate dependencies. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 2, pages 69–78. IEEE, 2015.

[13] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidović, and R. Kroeger. Measuring the impact of code dependencies on software architecture recovery techniques. *IEEE Transactions on Software Engineering*, 44(2):159–181, 2018.

[14] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*, pages 50–59. IEEE, 1999.

[15] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *Program Comprehension, 1998. IWPC'98. Proceedings., 6th International Workshop on*, pages 45–52. IEEE, 1998.

[16] O. Maqbool and H. Babri. Hierarchical clustering for software architecture recovery. *IEEE Transactions on Software Engineering*, 33(11), 2007.

[17] O. Maqbool and H. A. Babri. The weighted combined algorithm: A linkage algorithm for software clustering. In *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on*, pages 15–24. IEEE, 2004.

[18] A. Nicolaescu and H. Lichter. Behavior-based

architecture reconstruction and conformance checking. In *Software Architecture (WICSA), 2016 13th Working IEEE/IFIP Conference on*, pages 152–157. IEEE, 2016.

[19] V. Tzerpos and R. C. Holt. Acdc: an algorithm for comprehension-driven clustering. In *Reverse Engineering, 2000. Proceedings. Seventh Working Conference on*, pages 258–267. IEEE, 2000.

[20] Z. Wen and V. Tzerpos. An effectiveness measure for software clustering algorithms. In *Program Comprehension, 2004. Proceedings. 12th IEEE International Workshop on*, pages 194–203. IEEE, 2004.

[21] J. Wu, A. E. Hassan, and R. C. Holt. Comparison of clustering algorithms in the context of software evolution. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 525–535. IEEE, 2005.

[22] C. Xiao and V. Tzerpos. Software clustering based on dynamic dependencies. In *Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on*, pages 124–133. IEEE, 2005.

Partitioning Data for Massive Data Processing Applications: an Overview over Data Sharding Techniques

Benedikt Conze
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
benedikt.conze@rwth-aachen.de

Elder Magalhaes
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
elder.magalhaes@rwth-aachen.de

ABSTRACT

Sharding, the process of splitting a database into smaller partitions, is becoming more prevalent as the amount of data that needs to be processed increases. Sharding may be used to split databases across multiple systems to reduce the load each system has to handle. Furthermore, sharding helps to accommodate growth of a database by distributing the data onto different systems and thus allows for horizontal scalability. Another use case where sharding is of importance is cluster computing where distributing data across multiple systems can improve parallelism. In this paper we provide an overview over sharding algorithms which are used in the areas of cluster computing and database management. The reasoning behind this paper is the lack of such an overview which might be useful in order to choose the right algorithm for a given task. Therefore, we will describe which algorithms might be best suited for which situation.

Keywords

sharding, data partitioning, data processing

1. INTRODUCTION

Every year the amount of data that data processing applications have to handle and database management systems have to store increases. In 2012 alone 2.5 exabytes of data were created every day[17]. Therefore, the issue of partitioning the given data in a sophisticated manner becomes more prevalent. Partitioning data is splitting the data into smaller chunks. This paper focuses on horizontal partitioning, that is the displacement of whole database rows to chunks.

Partitioning serves many purposes: it is often necessary to split the data across multiple machines to reduce hardware costs, since smaller cheaper machines are used instead of more powerful and more expensive ones, and to increase the scalability and availability of the data. Partitioning can also be useful to process large amounts of data in parallel to increase performance. Partitioning is not a trivial task but has to deal with lots of use case-specific problems like (time-varying) data skew, load balancing and cross-partition transactions. Therefore many specialized algorithms exist to tackle these problems in different circumstances. We want to present a concise overview over the subset of these algorithms that target databases or data processing of list- or graph-structured databases.

1.1 Problems

The reason for there being so many different partitioning algorithms comes from the fact that different problems can arise in different use cases and each algorithm tries to mitigate different aspects of the problems. Frequent problems with data partitioning are data skew, workload skew and temporal skew.

1.1.1 Data Skew

Data skew occurs when one partition contains much more data than other partitions. This can result in one partition being queried more often than others or reduce the ability to process data in parallel. Also the algorithms have to make sure that the partitions fit into the machine they were assigned to.

1.1.2 Workload Skew

Workload skew happens when a partition gets queried much more often than others, or when a processing unit has to process more data than other units. One possible source of this problem comes from data skew, however it can also arise when some part of the data is needed more often than other parts.

1.1.3 Temporal Skew

Temporal skew is the time-dependent uneven distribution of workload or data over several machines. An example for this is Wikipedia partitioning its database by language[19]. This way Wikipedia minimizes the number of cross-partition data requests, but on the other hand the workload is subject to a strong temporal skew: the Wikipedia sites of a language are mostly accessed when it is daytime in the regions the language is spoken in and are mostly idle when it is night. Thus the workload fluctuates between small at night and high at day. A better solution might be to store different languages on the same partition, so that the workload is roughly the same throughout the day. In section 3.1 we describe an algorithm that is specifically designed to create partitions when a strong temporal skew is present.

1.2 Scope of this Paper

This paper focuses on partitioning algorithms for databases and processing frameworks. In both areas we have algorithms for list- and graph-partitioning. Our decision to restrict this paper to those domains came from the fact that these are well studied areas of research with many different techniques, however in both of those domains there is a lack of overviews over the existing methods. There are a plethora of partitioning algorithms for different use cases and to describe them all is not achievable in this paper. As such we only looked into relevant partitioning algorithms that were specialized for data processing or databases.

1.3 Related Work

There already exist some overviews of different partitioning techniques, however these papers mostly compare simple methods the authors implemented themselves and don't discuss other more popular methods.

Bertolucci et al.[4] discuss multiple partitioning algorithms in Apache Spark. However all of them are very basic and the comparison and performance test were in regards to how fast the different algorithms finished using one of the proposed partitioning algorithms. No other partitioning methods except for their own were included in the tests.

Devi and Sindhura[8] analytically compare different methods that seek to reduce partitioning skew in Hadoop MapReduce. Our paper will discuss two different methods in this domain, not described in [8], in greater detail than the methods discussed there.

Ancy et al.[3] compare locality based partitioning algorithms. However, this paper doesn't compare the methods with each other and some of them are only shortly described, not giving a full picture about the capabilities. In our paper we describe LEEN, which is also a locality based partitioning algorithm.

1.4 Structure of this paper

The rest of the paper is organized as follows. In section 2 we introduce basic partitioning algorithms. In section 3, we discuss the algorithms related to or that can be applied to partitioning data in relational or graph databases. In section 4 we discuss the algorithms that partition data in data processing environments. We decided to divide the algorithms into these two categories since depending on the use case the priorities of what the partitioning algorithm should achieve are different. In section 5 we conclude our work and discuss in what direction future overviews of sharding techniques should go.

2. BASIC PARTITIONING ALGORITHMS

To partition data there exist many different algorithms. Three common algorithms are range partitioning, hash partitioning and round-robin partitioning[9]. Even though these methods are easy to implement, they have shortcomings that create the need of more advanced methods that handle certain problems, like data skew. However, these advanced methods are often built upon the standard partitioners.

2.1 Range Partitioning

For range partitioning the user determines a range of keys for each partition which are then assigned to it[9]. For example, a table consisting of two columns, Name and Age, may be split into two partitions in the following manner:

```
Data: Keys K to be partitioned, partition $n_i$ to be
      partitioned to
forall $k$ in Age do
    if $k < 18$ then
        assign k to $n_1$;
    else
        assign k to $n_2$;
    end
end
```

Algorithm 1: A simple range partitioning implementation

A problem that may arise from the example shown in Algorithm 1 is that if there are only ten people which are younger than 18 and thousand that are older, then the partitions will have disproportionate sizes. Computations done on the partition containing the people which are older than 18 will therefore take longer creating an unbalanced load on the network. Another problem with range partitioning is that the conditions are very rigid and it is difficult to maintain such a partitioning. If the number of partitions is changed then the algorithm has to be adapted to the new conditions, the already partitioned data however might not conform to the new partitioning conditions hence that data has to be moved, which is cost intensive. Furthermore, mitigating partitioning skew becomes dependent on the defined ranges and even if fair partitioning ranges are initially found, new data might cause skew such that the ranges need to be recalculated in order to mitigate the skew.

2.2 Hash Partitioning

Another partitioning method is hash partitioning. With this method keys are mapped to a partition using a hash function. By hashing the keys, the data is randomly distributed among all partitions. This reduces the risk of partitioning skew regarding the frequency of a key compared to the range partitioning. One disadvantage of distributing the data randomly is that it is possible that keys that are requested often will be assigned to the same partition, or that some partitions get assigned more keys than others resulting in partitioning skew. Also related data is not stored in the same partitions[9], thus if related data is requested then different partitions will be queried, creating more load than would else be necessary.

2.3 Round-Robin Partitioning

In round-robin partitioning each key k_i is assigned to a partition with index $i\%n$ for n partitions. This method is fairer in comparison to range and hash partitioning in the context that every node receives approximately the same amount of keys. However if the data is skewed and some keys are requested more often, then this will also lead to partitioning skew. One such case could be that the names of popular people are looked up more often than not popular names. Round-robin partitioning ignores other aspects of the data it partitions and thus it could result in creating a partition that requires more bandwidth from the network than other partitions.

3. DATABASE PARTITIONING

Database partitioning algorithms in general create a partitioning that the underlying data is going to stay in for a long amount of time. To achieve good performance in interaction with this data different techniques are used, which are described in this section. Additional algorithms are possible and required if the data is structured in a graph database.

3.1 Horticulture

Horticulture is a database partitioning algorithm especially designed for online transaction processing (OLTP) systems[19]. OLTP systems have a workload that consist of a large number of small transactions, which still have to satisfy ACID guarantees[11], i.e. have strong transactional and consistency requirements. A transaction in this context is an operation that accesses and/or modifies the database. Furthermore these transactions are short-lived. This means that most transactions stand for themselves instead of being part of a long chain of transactions, where the memory would have to stay consistent all the time. The transactions are repetitive, so that the required workload can be approximated in advance. Horticulture uses these characteristics to extrapolate an expected workload out of a given sample workload. As Horticulture takes time-varying frequency of transactions into account, it is also well suited for databases where this effect plays a big role.

3.1.1 Functionality Overview

Horticulture tries to minimize the number of transactions that span over multiple partitions, because the necessary communication between the partitions comes with a high network overhead. It also tries to balance the workload of all partitions taking time-varying workload into account, as described in section 1.1.3. It explicitly does not try to spread data across partitions for a more parallelized workflow as other algorithms do[18], because this is in general not worth it on an OLTP workload[19].

Horticulture tries to find a good partitioning scheme by executing a large-neighborhood search. This works by first creating an initial partitioning out of the sample workload. After that it changes a subset of the partitioning parameters and performs a local search around this changed solution. If the new partitioning is better, the original solution is replaced by the new solution and the process is repeated. The first change of the parameters, after a new solution is found, needs to be big so that the algorithm does not get stuck inside of a local minimum. The different database designs are compared by a newly created metric, which takes the database scheme, the given common procedures and the given sample workload into account.

3.1.2 Primary and Secondary Indices

In its process to partition the database, Horticulture creates index lists. An index list is a sorted array of pairs of attribute values and pointers, that point into the table row with the attribute value. An index list is said to be primary, if the table is sorted according to the attribute value of the index list, and secondary if not. There can only be one primary index per table. Indices allow for fast table look-up given only the attribute value, because a search in a sorted list takes logarithmic time, and especially a primary index allows for fast range queuing in the database.

3.1.3 Algorithm

The input of the horticulture is a set of stored procedures, a sample workload including timestamps and the database itself. Stored procedures are functions with input parameters, that contain multiple transactions. In this summary of the Horticulture algorithm we will disregard them. First of all an initial solution is created. The given workload is traversed to find the attributes that are most commonly accessed. Every table is then range partitioned after its most frequently used attribute with automatically determined range boundaries. For the second most frequently accessed column, a secondary index table is created. The remaining unused space of each partition is filled up with replications of the read-only tables with the highest temperature. Temperature is calculated as the number of transactions on the table divided by the number of rows of the table[1]. Additionally to the temperature the tables are chosen with the time skew of transactions on them in mind.

In all steps to incrementally improve the first solution a big number of partitioning attributes, secondary index-attributes and table replications are semi-randomly changed. From there on a new solution is searched for with little changes. This technique tries to prevent the algorithm to be trapped inside local minima. If the solutions found by the small changes are all worse, they are discarded and the next step starts with the same initial solution. If a better partitioning is found, the initial solution is replaced with the new solution in the next iteration.

Horticulture uses a customizable cost model to compare solutions with each other. The cost of a solution with a given workload basically boils down to its ”coordination cost”, which describes how well a solution minimizes the number of cross-partition transactions, times its ”skew factor”, which rates how little workload skew exists in the database at any point in time[19]. The database administrator can manually decide how important both factors are by using two parameters in the cost formula.

3.1.4 Conclusion

Horticulture has a narrow area of application, that being OLTP systems with time skewed transactions. The authors however claim to excel in this field and prove their success by multiple benchmarks, which amongst other things show that Horticulture can produce partitionings four times faster[19] than the Schism algorithm[7], which we present later in this paper.

3.2 Balanced Label Propagation

After having looked into a partitioning technique for lists we now discuss techniques to partition graphs. The main difference between graphs and lists is that graphs can have multiple connections to other data points in the data set.

The main goal of many graph database partitioning algorithms is to minimize the number of edges between partitions. This is because the edges correspond to cross-references between data nodes inside of the database. When accessing a node inside a partition it is likely that one may access the neighboring nodes. If these nodes are in another partition this increases expensive cross-machine communication.

The problem of partitioning graphs into a given number of partitions with minimal edges in-between partitions is known as the min-cut problem[20], and the variant of this with balanced partitions is known as balanced min-cut. This is known to be a NP-complete problem[10], but a lot of research went into approximating a solution for this problem

[1]In the horticulture paper this is defined the other way around[19].

in a reasonable time frame. Subsequently we will show some of these algorithms that are trimmed to produce the best results on the kind of large graphs databases often represent.

3.2.1 Motivation

Balanced label propagation is an algorithm that tries to build partitions of nodes by labeling neighboring nodes with the same label. The nodes with the same label get assigned into the same partition, each label corresponds to a partition. This however introduces the need for a look-up-table, which lists the partition for every label.

Ugander et al.[25] created their algorithm for massive databases, specifically for Facebook's "People-You-May-Know" (pymk) database. In the case of such massive data graphs increasing the locality and therefore reducing the need of querying different partitions increases the performance more than the decrease caused by the look-ups. Locality in the context of this paper defines the percentage of edges that are connected with nodes within the same partition, as was described in the introduction of this section.

One problem with typical balanced label propagation is the stopping of the propagation. The propagation needs to be stopped in order for the partition to fit into the machine they were assigned to, else a trivial solution would be to only create a single partition containing every node, which isn't feasible due to the limited space of the machines. In traditional label propagation this is done by using cost penalties. In their paper Ugander et al.[25] however define a set of constraints. These constraints not only allow to balance the partitions, making them the same size, but also to create unbalanced partitions if that is desired.

3.2.2 Constraints

The first constraint is to define a lower boundary S_i and upper boundary T_i, with $0 \leq i < n$, for n partitions. The boundaries are determined by the user. To initialize the algorithm, Ugander et al. assign labels to nodes randomly. The label represents the partition the node will later be assigned to, respecting the constraint $S_i \leq |V_i| \leq T_i$, $\forall i$ where V_i is a partition of the graph $G = (V, E)$. An optimization of the initialization will be described later on. The algorithm creates n partitions.

After the initialization Ugander et al. then define a maximization problem. Every time data wants to move from a partition to another the utility of this relabeling is calculated. The utility here is defined as the amount of neighbors with the same label. They try to maximize the utility gained from relabeling the data while respecting these constraints. The third constraint, here given as equation 2, is that after the moving of data, the sizes of the partition are still within S_i and T_i. The fourth constraint is that no more data is moved than data that wants to be moved. Data wants to move if the utility gained from the movement is higher than if it keeps the label it currently has.

The maximization problem can be formulated as follows:

$$max_x \sum_{i,j} f_{ij}(x_{ij}) \qquad s.t. \tag{1}$$

$$S_i - |V_i| \leq \sum_{j \neq i} (x_{ij} - x_{ji}) \leq T_i - |V_i|, \forall i \tag{2}$$

$$0 \leq x_{ij} \leq P_{ij}, \forall i, j \tag{3}$$

f_{ij} determines the utility from moving data. $x_{i,j}$ is the amount of nodes that will be relabeled from label i to label j. P_{ij} is the amount of data that wants to be relabeled from i to label j.

3.2.3 Balloon Partitioning and Geographic Initialization

With the random initialization of the labels Ugander et al.[25] could only achieve 44.8% locality after 8 iterations. In order to improve their algorithm, a different initialization is described. Instead of randomly assigning nodes to partitions, the nodes get labeled according to the city where the users come from. This is useful in the Facebook pymk database, since people are more likely to be friends with people that live nearby[25].

Then the balloon partitioning algorithm is applied. This algorithm tries to create n partitions. These can contain multiple cities c. A partition P has a cost of $\frac{\sum_c Cost(c)}{n}$ $\forall c \in P$. The cost of a city could be defined as the amount of nodes with that city as label. However for the Facebook database specifically, the cost of a city is calculated using the degree of its nodes. People from some countries and people living in cities tend to have more friends[25], thus the friend-of-friend calculation is more costly. Therefore an objective was that a partition containing a popular city contains fewer nodes.

In the beginning no city is assigned to a partition. Then the city c with the largest unassigned cost is determined, after which the remaining cities with a non-zero cost are sorted according to their distance to c, and assign c to a partition. After this, they then assign the following cities to the partition containing C as long as the cost is below $\frac{\sum_c Cost(c)}{n}$. When the next city does not entirely fit, then that city gets partially assigned to the partition. The portion of the cost that was assigned to the partition gets deduced from this city. When a partition is full, then the algorithm finds the next city with the largest unassigned cost and assigns that city to a new partition, then the algorithm repeats again until every node is assigned to a partition[25].

With this initialization a much higher locality is achieved in less iterations when compared to the random initialization. When initialized with this algorithm the initial locality is 52.7%, with random initialization a locality of 44.8% was only reached after 8 iterations[25].

3.2.4 Oversharding

One point discussed by Ugander et al.[25] is oversharding. A problem of this partitioning algorithm is that it is possible that the most popular cities get partitioned into the same partition. Even if the cost might be balanced this partitioning would not account for the workload at peak times. Therefore more, thus smaller, partitions are created than there are machines and then the partitions are sorted according to their most populated city. Then they cyclically assign the partitions to n machines. This results in partition 1 and partition $(n + 1)$ being assigned to the same machine[25]. This is done to fight temporal skew.

3.3 METIS

METIS is a state-of-the-art[20] graph partitioning algorithm used in community discovery in large graphs, load balancing in parallel computing and partitioning of graph databases. It promises a high-quality partitioning in little time while being very customizable to the specific use case[1].

3.3.1 Algorithm

The METIS algorithm works in three phases. The broad idea is to first coarsen the graph by reducing the number of vertices, partition the resulting smaller graph and scale the result back up to the original graph.

The coarsening phase is done in a large number of iterations. In each step at first a maximal matching is calculated, which is a subset of edges of a graph so that the selected edges have no common adjacent vertex. Two vertices that are adjacent in the resulting set are then merged together in the original graph. Each vertex and edge receives a weight to keep track of the number of vertices or edges that were merged into this vertex or edge. METIS provides four different algorithms to compute the maximal matching.

When the number of vertices in the graph is small enough to perform an inexpensive partitioning, the second phase begins. A minimum k-way edge cut is calculated while taking the vertex and edge weights into account to balance the partitions. METIS again provides four different algorithms to calculate the minimum edge cut. After that the un-coarsening phase begins. It is again incrementally done in the same amount of steps phase one was completed. In each iteration, each vertex in the finer graph is assigned to the partition the corresponding vertex in the coarser graph was assigned to. Additionally a partition refinement algorithm is used, which tries to swap the partitions of two sets of vertices and tests if the new allocation leads to fewer cross-partition edges. Again, multiple algorithms are provided[14][16].

3.3.2 Use Case

METIS becomes very customizable by dividing its process into clearly confined phases. Since each step may be optimized for the specific needs of an use case, METIS can be used in a lot of situations. Some of the prebaked variants of METIS include hMETIS with a focus on quality and kMETIS with a focus on speed. There also exists ParMETIS, which makes the algorithm able to concurrently run on multiple machines[1].

3.4 Ja-be-Ja

Ja-be-Ja is another min-cut algorithm whose main focus lies in parallelizing the process of finding the best partitioning. It performs no global operations on the graph, which would require costly synchronization between the machines. It also requires no strict synchronization, which means that changes on subsets of all nodes don't have to be propagated to the rest of the graph. With all of these parallelizing mechanism in place, Ja-be-Ja still claims to have a result comparable to the one of the METIS algorithm[15] while having a much lower execution time on its application fields[20].

3.4.1 Algorithm

Below we will roughly describe the steps of the Ja-be-Ja algorithm. Just like in the original paper we will refer to the partition of a node as the color of the node.

In its initial step Ja-be-Ja initializes each node with a random color according to the users requirements. Since Ja-be-Ja later on only performs color swaps on the graph, it is guaranteed that the initial color distribution will be kept until a solution is found. This also makes it possible to create partitions with different, predetermined sizes, by adjusting the frequency of the colors in this first step.

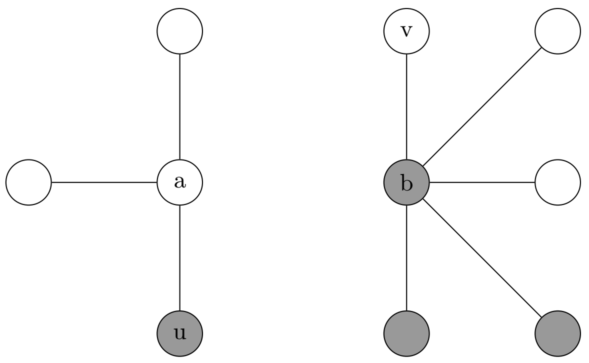


Figure 1: Nodes a and b swap their colors only if $\alpha > 1$

Next, for every node each of its neighbors and some nodes from a uniform random sample from the graph are stored for later reference. Ja-be-Ja uses the Newscast algorithm[24] for its examples and benchmarks, because it can take a uniform sample of the graph with low cost and without global operations or strict synchronization. Rahimian et al.[20] list however multiple other techniques to take a uniform sample of a graph.

Now the core mechanism of Ja-be-Ja comes into play: Each node iterates through its stored samples of nodes and selects a node according to a given algorithm, which we will describe in more detail below. In short, each node checks if a swap of colors would result in overall less cross-color-edges in the neighborhood of the two nodes. If both nodes come to the conclusion that a swap is beneficial, it is executed. This algorithm is performed for each node of the graph in parallel, until a given time limit is reached. The whole algorithm is then repeated many times with different initial random distributions of colors following the Generic Multistart Paradigm[23] and the result of each iteration is compared in the following manner: every time an edge-cut of the graph is calculated via a gossip-based aggregation method[13] and that is used as a metric to compare the solutions to find the one that overall performs best.

3.4.2 Customization

The formula used to decide whether or not two graph nodes should swap their colors makes use of a lot of parameters that can be fine-tuned based on the application field and the type of graph. Two nodes p and q swap their color if the following formula holds true:

$$(x_p(\pi_q)^\alpha + x_q(\pi_p)^\alpha) * T > x_p(\pi_p)^\alpha + x_q(\pi_q)^\alpha \quad (4)$$

In this inequation $x_a(\pi_b)$ stands for the number of neighboring nodes of node a that have the color of node b. This means that if we choose $\alpha = 1$ and $T = 1$, two nodes will swap their colors if they have strictly less nodes of their own color in their neighborhood than they have nodes with the color of the other nodes in their neighborhood.

In the example given in figure 1 the nodes u and v would swap their colors if one of them holds a reference to the other one. This is because $x_u(\pi_v) = 1$, $x_v(\pi_u) = 1$, $x_u(\pi_u) = 0$ and $x_v(\pi_v) = 0$ and therefore because of $T \geq 1$

$$(1^\alpha + 1^\alpha) * T = 2 * T > x_p(\pi_p)^\alpha + x_q(\pi_q)^\alpha = 1. \quad (5)$$

α is a parameter that forces the exchange of colors on nodes if one node profits a lot from the change, even if the overall number of cross-color edges stays the same or even goes up. In the example given in figure 1 this would affect the swap of the nodes a and b: Although a swap of these two nodes does not result in a lowering of the number of cross-color edges ($x_a(\pi_b) = 1$, $x_b(\pi_a) = 3$, $x_a(\pi_a) = 2$ and $x_b(\pi_b) = 2$, $1 + 3 \not> 2 + 2$), the colors of the two nodes are swapped if $\alpha > 1$ ($1^\alpha + 3^\alpha > 2^\alpha + 2^\alpha$ for every $\alpha > 1$). A swap in this kind of situation was found to often be desirable, and how to tune this parameter correctly is described in great length in the original paper[20].

T is a temperature parameter as taken from the Simulated Annealing technique[23]. It decreases over time towards 1 and tries to prohibit local optima. This is achieved by allowing unfavorable swaps in the beginning which can get the graph out of a local optimum. Later, when the Temperature T approaches one, swaps have to be more conservative so that the currently targeted optimum can be reliably reached. The Temperature T can be tweaked by setting the initial temperature T_0, which has to be greater or equal one, and the decrease of temperature per round δ. The temperature at round r is then calculated by $T_r = max(T_{r-1} - \delta, 1)$.

3.4.3 Usage Example

Ja-be-Ja is not only used in the partitioning of graph-structured databases or in the partitioning of list structured databases (as we will later see with Schism[7]), but is also used in the context of graph processing. E. Carlini et al.[6] propose the use of Ja-be-Ja in the graph processing framework Apache Spark because of Ja-be-Ja's focus on parallelism. However the authors had to alter the Ja-be-Ja's algorithm to accommodate the use case circumstances: while Ja-be-Ja allowed every node to compute its next swaps in its own pace, this implementation synchronizes all computation and swaps between the states of computing the next swap and communicating to the target node. This implementation also needs every node to know the neighborhood of every node in their selection, which further adds memory and time overhead.

3.4.4 Conclusion

Ja-be-Ja claims to have a similar result in terms of quality like METIS[15], while parallelizing the process of finding a solution and therefore improving the performance on distributed graphs[20], which makes it ideal to use on for large graph data structures. It furthermore offers multiple connection points for the potential of customization. For these reasons it is a widely used algorithm in database partitioning and will be used in some of the other algorithms that we will present in this paper.

3.5 Schism

Schism is a database partitioning algorithm that works on relational databases and tries to find a partitioning strategy that works best in the given situation. What makes Schism stand out of the database partitioning algorithms is its internal use of a graph partitioning algorithm and its attention to the specific workload of the given database, which it extrapolates from a given sample workload. It may be used as partitioning scheme for a database management system[7].

Schism takes a representative workload (f.e. a SQL trace) and a database as input and first constructs a graph with a node per database row. For each access of two rows in the same transaction in the given workload a graph edge is added. After that a balanced graph partitioning algorithm is applied on the graph to produce a potential partitioning. At last the sample workload is analyzed to find the most frequently accessed attributes and adds them to special dedicated partitions.

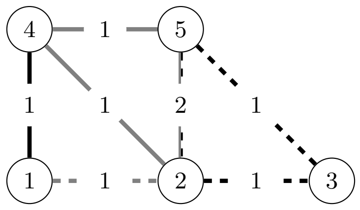


Figure 2: Schism graph for a sample table and workload

3.5.1 Shared-Nothing

Schism is optimized to operate on shared-nothing databases[7]. This means that each machine has sole access to the data it holds and is therefore the only machine that operates on and works with the subset of the data it holds. In a pure shared-nothing architecture there exist no shared memory or data storage. This has the advantages of having no single point of failure (if a machine fails only the data it works on is temporarily not available) and being very scalable, as enlarging the database can simply be done by adding more machines.

3.5.2 Graph Partitioning

Figure 2 shows an example corresponding to a single table with five rows, which were assigned indices from 1 to 5. Edges connect nodes that are used in the same transaction in the sample workload. If two nodes are used in two transactions together, the weight of the edge that connects them is incremented. In the above example, each style of an edge stands for a common transaction. For example, the nodes 1 and 2 may share a transaction that had the following lines in it:

```
SELECT * FROM table
    WHERE id < 3
```

Two nodes with index 2 and 5 where both used in two transactions, so their node weight is set to two.

After that a graph partitioning algorithm is used to create non-overlapping workload- and data size-balanced partitions. To make the partition data size-balanced, each node can be provided with a weight. A partition weight then is the sum of its node weights, which the graph partition algorithm tries to equalize. Furthermore it tries to minimize the number and the weight of the cross-partition edges.

Furthermore Schism tests for each row if it should be given solely to one partition or if it should be replicated over all partitions and afterwards constantly be updated on every machine if it changes. This would reduce the traffic between nodes. In general, tuples that change often according to the sample workload are not replicated because the cost of updating its value on every partition increases with the number of updates.

Schism uses the METIS algorithm[15] as a balanced graph partitioning algorithm but notes that any graph partitioning algorithm may be used here, possibly including ones like Ja-be-Ja[20], parMETIS[1] or KaFFPaE[22], which offer a better performance than METIS because of a distributed workflow with a similar result[20].

To further improve performance Schism uses a look-up table on a separate machine that stores the key of each element and maps to the partition it is stored on. This significantly improves performance of SQL WHERE-queries that search for specific keys as they do not have to traverse the whole database anymore.

3.5.3 *Conclusion*

Schism in considered the state-of-the-art academic approach for shared-nothing database partitioning[19]. It can be used in many situations as it does not depend on the underlying structure or schema of the database and can be customized in many aspects.

4. PARTITIONING ALGORITHMS FOR DATA PROCESSING

After having described some algorithms to partition data into databases, we will now discuss partitioning in the context of data processing. When talking about data processing with large amounts of data, then one important point is parallelization. This allows to process more data at the same time on multiple machines. This however only works if the load is balanced fairly amongst all the machines. How the data will be partitioned and how the load can be balanced will be discussed in the following, in particular in the context of the MapReduce algorithm which allows the user to implement a custom partitioner.

4.1 MapReduce

4.1.1 *The Algorithm*

A popular algorithm to process large amounts of data is the MapReduce algorithm. This algorithm allows to process data in parallel and thus processing it faster. Data is stored in partitions which are distributed over different machines. In the map phase, data stored on such a partition is passed through a mapping function which outputs intermediate $< key, value >$ pairs. This phase runs in parallel on every machine. The second phase is called the reduce phase, in this phase the outputs from the map phase get combined together to create the desired output. Between those phases there is a partitioner, which takes the intermediate $< key, value >$ pairs and decides which key gets assigned to which reducer, the keys are not unique and pairs with the same key get assigned to the same reducer[5]. How this decision is made depends on the partitioner, this partitioner can either be the standard partitioner or another custom partitioners, how some of these partitioners work and what their advantages and disadvantages are, are discussed later in this paper. On every machine, the reduce phase can only happen after the map phase, however the reduce phase can start before every mapper is finished, this is what Ibrahim et al.[12] define as synchronous implementation of MapReduce since the map phase and the reduce phase can run concurrently[12].

4.1.2 *Standard Partitioner in Hadoop*

A popular framework implementing the MapReduce algorithm is Apache Hadoop[2]. As explained in section 4.1.1 to partition the keys in MapReduce the user can either create a custom partitioner or he can use the standard partitioner. To mitigate the problems of the range partitioning and use the benefits of both hash and round-robin partitioning, the standard partitioner in Apache Hadoop determines the reducer to which the keys are partitioned with the following formula[2][5]:

$$hash(intermediate\ key)\ modulo\ n$$

n is the amount of available reducers. By hashing the intermediate keys it is assured that the keys are randomly distributed and the modulo assures that each reducer receives the same amount of keys. However the combination of both methods still doesn't solve the problem of partitioning skew, the frequencies of keys is ignored, a key that is much more frequent will produce a higher load, and thus the standard partitioner is not sufficient if data skew is present. To solve the partitioning skew problem different methods were developed[12][27] which we will discuss in the following section.

4.2 LEEN

LEEN is an algorithm that can be used as a partitioning algorithm in Hadoop's MapReduce as discussed in section 4.1.1 In their paper Ibrahim et al. discuss how partitioning skew results in excessive data transfer when the keys get assigned to reducers. This algorithm tries to solve this problem by being locality-aware, in the context of [12] this means the algorithm tries to keep the data on the machine where it originally was. Furthermore this algorithm is also fairness-aware, meaning that it tries to assign the same amount of data to process to each reducer.

4.2.1 *Asynchronous MapReduce*

As explained LEEN was developed for the MapReduce algorithm, and as explained in section 4.1.1 the reducers can start working before every map function is finished. In order to achieve the best results LEEN proposes an asynchronous implementation of the MapReduce algorithm. Asynchronous in this context means that the reduce phase can only start after every mapper is finished, in opposition to the synchronous implementation of MapReduce in which the reduce phase can start before the map phase is completely finished. The asynchronous implementation is used in order for LEEN to have information on all the keys frequencies and distribution and thus achieve better locality and fairer distribution[12].

4.2.2 *Algorithm*

The objective of LEEN is to partition the keys by trying to balance the locality and the fairness. So we first define those terms in this context.

Locality defines how the keys are distributed. A key that has a high frequency in few partitions is more local than a key that is distributed amongst many partitions, it is more "localized".

Fairness in this context means that the load is fairly distributed amongst all the reducers, so that resources are used more efficiently. So fair in this context refers to the reducers doing their fair share of the work.

LEEN now tries to find a partitioning which takes into account the locality of the keys while at the same time trying to assign the keys fairly.

[2]Some papers define this function differently.[12][27]

Since finding the best solution is too costly, LEEN applies a heuristic. It sorts the keys according to their potential impact on the network bandwidth, and then partition the keys with the highest impact first. The impact is determined by the frequency of a key and how it is distributed. A key that is more frequent than other keys and at the same time is distributed among many partitions, will require much bandwidth to transfer them to the assigned partition.

After sorting the keys the LEEN algorithm now tries to find the best partition for that key. It sorts the partitions according to the frequency of the key that is to be assigned. After that LEEN calculates a *fairness-score*. This score expresses how the fairness will change if the key gets assigned to that partition that the calculation is performed for. The score is calculated for the first partition, and then for every following partition until the score gets worse. As soon as the score gets worse the LEEN algorithm takes the partition with the best score and assigns the key to it. Then the algorithm gets repeated for the next key until each key is assigned to a partition. That the partitions get sorted according to the frequency is so that the partitions where the key is most local get checked first if it would be fair to assign the key to that partition, thus saving bandwidth.

4.2.3 Problem

One major problem of this algorithm is the need to use asynchronous MapReduce. With heavy data skew a mapper might take particularly long slowing the whole algorithm down. Also there is a need of a central controller to monitor the key frequencies, with big amount of data this might also lead to a performance degradation[27]. Furthermore, since the keys are not split up, if there is heavy partitioning skew it might not be possible to fairly balance the load among all the reducers, as shown in figure 3. No matter how the keys get assigned, the partition that gets key 1 will have a much bigger load, since key 1 is much more frequent than the other keys.

However, these shortcomings are acknowledged in [12] and they claim that even though there might be slowdowns, the objective is to win time because of the reduced needs for shuffling the data. In fact in their paper they achieve up to 40% faster execution times when compared to the standard Hadoop partitioner[12].

4.3 Sampling Based Partitioning

Like LEEN the sampling based partitioning algorithm is an algorithm that was designed to be used as a partitioner in the MapReduce algorithm. An objective of this method is to handle partitioning skew, such that the load gets balanced between the different reducers. The main idea of this algorithm is to be able to estimate the frequencies of the keys. This estimation is based on a sample of the data to be processed. One disadvantage of LEEN was that in order to partition the data, they have to wait until each mapper is finished. This however can lead to long waiting times if one mapper takes much longer than the other mappers due to data skew. Therefore Xu et al.[27] propose an algorithm that utilizes the benefits of synchronous MapReduce.

4.3.1 The Sampling

To utilize the benefits of synchronous MapReduce, instead of waiting for each mapper to finish, the map and reduce phases can run concurrently. As soon as the first mapper is finished the first reducer can begin without the need to wait for every mapper to finish. The sampling based algorithm runs in two phases. In the first phase it performs a MapReduce on a sample of the data. This means it takes an equally sized sample from every data set and performs a MapReduce on the samples. The output of this phase is a partitioning scheme, which takes $< key, value >$ pairs and assigns them to a reducer[27].

In the second phase the partitioning scheme created in the first phase is used as a partitioner, thus intermediate keys can immediately be assigned to a reducer. This allows the map phase and the reduce phase to run concurrently without the need to wait for every mapper to finish, in the case of LEEN. To make a load balanced partitioning scheme, they need an estimation of the distribution of the keys.

To obtain a sample Xu et al. select keys at random with a probability of $p = \frac{1}{\epsilon^2 N}$[27]. Here N is the amount of keys in this partition and $\epsilon \in (0, 1)$ determines the size of the sample. If ϵ gets smaller than the probability of a record to be selected rises and thus the samples become bigger. To estimate the frequency of keys in a partition they calculate the frequency of keys in the sample as:

$$s(k) = \sum_i \delta(k_i, k) \quad \forall i \text{ in the sample}$$

$$\delta(k_i, k) = \begin{cases} 1, & \text{if } k_i = k \\ 0, & \text{otherwise} \end{cases}$$

with k being the key being counted. This gives the frequency of key k in the sample. To now estimate how frequent a key appears in a partition, they calculate:

$$\hat{c}(k) = \frac{1}{p} s(k)$$

In their paper they formally prove that the error between the estimation and the real value is less than $3\epsilon\epsilon_1 N$ with $\epsilon_1 \in (0, 1)$[27]. If a smaller error is required then it suffices to decrease ϵ resulting in a bigger sampling[27].

4.3.2 Partitioning Schemes

After the sampling and calculating the estimation, Xu et al.[27] describe two different schemes to partition the data.

The first scheme proposed by [27] is called the *Cluster Combination Optimization*. In this scheme the main idea is to partition the most frequent key to the least busy reducer. All $< key, value >$ pairs with the same key form a cluster. The clusters are sorted descending according to their size, then the first n clusters get sequentially assigned to n reducers. After this step the reducers are sorted in descending order according to the frequencies of the keys they hold. Then the clusters get sequentially assigned to the last reducer, as this reducer has the least amount of keys. After each assignment the reducers get sorted again[27].

The second proposed method is called *Cluster Partition Combination*. In the first method a problem still arises if the data is heavily skewed. With the Cluster Combination Optimization the reducer would be unbalanced which might lead to a reducer needing significantly longer than the others.

In figure 3 we want to partition two clusters onto two reducers. However Cluster 1 contains many more values than Cluster 2, thus no matter the partitioning the load will be unbalanced as the reducer getting Cluster 1 has to compute

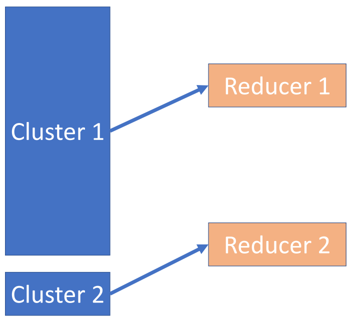


Figure 3: Based on an example of the problem of partitioning skew[27]

more. To resolve this Xu et al. propose a second method. For this method, what they propose is to split up the large clusters. This however violates the rule that each cluster should only be processed by one reducer[27]. Therefore they introduce a second reduce phase which merges the results.

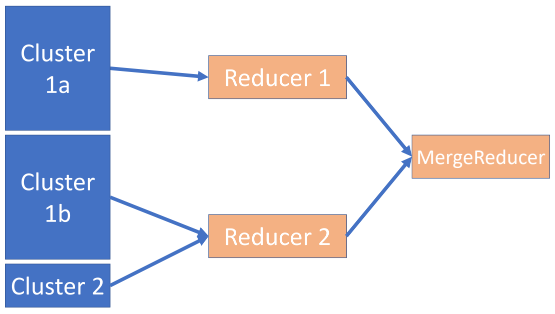


Figure 4: Based on an example of Cluster Partition Combination[27]

As can be seen in figure 4 the divided chunk gets reduced by two different reducers, however with the MergeReducer one achieves that each value from a cluster gets processed by the same reducer, thus not violating the rule. Furthermore, through splitting a large cluster up we see that the reducers get more even loads, as can be seen in figure 4 compared to 3. However this requires a second reduce phase which means data has to be transferred from the first reducer to the second one which might require more bandwidth.

4.3.3 Comparison with LEEN

The sampling-based algorithm and LEEN are both methods to partition data in MapReduce, however both have different approaches. The sampling-based algorithm tries to achieve load balancing between the reducers. This is also an objective of LEEN, however LEEN further tries to reduce the data transfer during the shuffle phase, which is ignored by the sampling based algorithm. Both describe different methods to gather information about the key distribution. LEEN waits for the map phase to finish and then partition according to the resulting distributions[12], however, to achieve this LEEN has to use an asynchronous implementation of MapReduce, which results in a slower execution since it has to wait for the slowest mapper. In comparison, to avoid the waiting the sampling based algorithm performs a first MapReduce on small samples and then creates a scheme on how to partition the $< key, value >$ pairs. Furthermore the sampling based algorithm proposes two schemes depending on how skewed the data is. This however requires the user to know how skewed the data in order to choose the best scheme. In LEEN this is not necessary.

Both papers have advantages: with LEEN the amount of data shuffled is reduced which reduces the load on the network and thus achieves faster results, while the sampling based algorithm loses time while performing the first MapReduce but then regains that time by being able to use the synchronous property of MapReduce[27]. So if the data to be processed has locality properties the application of LEEN might be better suited, however if not and if network bandwidth is not a problem then the sampling based algorithm might be best.

5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this paper we first presented the basic partitioning techniques and showed why they are not suitable for every case. We then presented three basic and eight more advanced techniques for different use cases. We ended the section of the database techniques with Schism, an algorithm that in itself combines database with graph partitioning. After that we presented different techniques for the MapReduce implementations like Apache Hadoop. Even though many of these techniques give significantly better results than the standard partitioning algorithms, none could provide ideal solutions, mostly because finding the ideal solution is a NP-hard problem. Therefore many techniques use heuristics and approximations in order to achieve acceptable results. In conclusion the reader now should have a broad notion of many different techniques which each emphasize different aspects of either the data as presented in section 3 or the processing as in section 4.

5.2 Future Work

This paper created an overview of different partitioning techniques, however no performance evaluation was made, because that would go beyond the scope of this paper. For future research extensive overviews for specific domains could be created that compare different partitioning techniques using many data sets, allowing for extensive analysis on when to use which method. Further research could also go in the direction of looking for possible performance increases by combining algorithmic partitioning techniques with hardware accelerated techniques[26].

Another aspect to increase performance is to look into how the data is partitioned on the individual nodes. The nodes inside a cluster of computers often contain multiple processors, partitioning the workload among these processor in a sophisticated manner can also lead to further performance increases[21]. Future research might also test the combination of such techniques, with techniques presented in this paper.

6. REFERENCES

[1] METIS homepage. `http://glaros.dtc.umn.edu`. Accessed: 2018-12-18.

[2] Tutorial describing MapReduce in Apache Hadoop. https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Accessed: 2018-12-16.

[3] S. Ancy and M. Maheswari. Locality based data partitioning in map reduce. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 4869–4874, March 2016.

[4] M. Bertolucci, E. Carlini, P. Dazzi, A. Lulli, and L. Ricci. Static and Dynamic Big Data Partitioning on Apache Spark. Technical report.

[5] B. â. Cambridge, â. Farnham, â. Köln, â. Sebastopol, â. Taipei, and â. Tokyo. Hadoop: The Definitive Guide Tom White foreword by Doug Cutting. Technical report.

[6] E. Carlini, P. Dazzi, A. Esposito, A. Lulli, and L. Ricci. Balanced graph partitioning with apache spark. In L. Lopes, J. Žilinskas, A. Costan, R. G. Cascella, G. Kecskemeti, E. Jeannot, M. Cannataro, L. Ricci, S. Benkner, S. Petit, V. Scarano, J. Gracia, S. Hunold, S. L. Scott, S. Lankes, C. Lengauer, J. Carretero, J. Breitbart, and M. Alexander, editors, *Euro-Par 2014: Parallel Processing Workshops*, pages 129–140, Cham, 2014. Springer International Publishing.

[7] C. Curino, E. Jones, Y. Zhang, and S. Madden. Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, 3(1-2):48–57, Sept. 2010.

[8] Y. S. Devi and K. Sindhura. A Survey on Partitioning Skew Diminishing Techniques in Hadoop MapReduce Environment. *International Journal of Innovative Research in Science, Engineering and Technology (An ISO*, 3297(9), 2007.

[9] D. J. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Processing.

[10] T. F. Gonzalez and T. Murayama. Algorithms for a class of min-cut and max-cut problem. In T. Ibaraki, Y. Inagaki, K. Iwama, T. Nishizeki, and M. Yamashita, editors, *Algorithms and Computation*, pages 97–105, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[11] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15(4):287–317, Dec. 1983.

[12] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi. LEEN: Locality/fairness-aware key partitioning for MapReduce in the cloud. In *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010*, pages 17–24, 2010.

[13] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, Aug. 2005.

[14] G. Karypis and V. Kumar. Metis – unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, 1995.

[15] G. Karypis and V. Kumar. Metis – unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, 1995.

[16] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[17] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. Patil, and D. Barton. Big data: the management revolution. *Harvard business review*, 90(10):60–68, 2012.

[18] S. Papadomanolakis and A. Ailamaki. Autopart: automating schema design for large scientific databases using data partitioning. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pages 383–392, June 2004.

[19] A. Pavlo, C. Curino, and S. Zdonik. Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 61–72, New York, NY, USA, 2012. ACM.

[20] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi. Ja-be-ja: A distributed algorithm for balanced graph partitioning. In *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*, pages 51–60, Sept 2013.

[21] D. SaccA CRAI Rende and C. Wiederhold. Database partitioning in a cluster of processors. Technical report.

[22] P. Sanders and C. Schulz. *Distributed Evolutionary Graph Partitioning*, pages 16–29.

[23] E.-G. Talbi. *Metaheuristics: From Design to Implementation.* Wiley Publishing, 2009.

[24] N. Tölgyesi and M. Jelasity. Adaptive peer sampling with newscast. In H. Sips, D. Epema, and H.-X. Lin, editors, *Euro-Par 2009 Parallel Processing*, pages 523–534, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[25] J. Ugander and L. Backstrom. *Balanced Label Propagation for Partitioning Massive Graphs.*

[26] L. Wu, R. J. Barker, M. A. Kim, and K. A. Ross. Navigating big data with high-throughput, energy-efficient data partitioning. *ACM SIGARCH Computer Architecture News*, 41(3):249, 2013.

[27] Y. Xu, P. Zou, W. Qu, Z. Li, K. Li, and X. Cui. Sampling-based partitioning in mapreduce for skewed data. In *Proceedings - 7th ChinaGrid Annual Conference, ChinaGrid 2012*, pages 1–8, 2012.

API-Driven Architecture and Development

Marian Kreiser
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
marian.kreiser@rwth-aachen.de

Pascal Brunner
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
pascal.brunner@rwth-aachen.de

ABSTRACT

In recent years several tools have been created to aid developers with the process of API development. API description languages like API Blueprint or the OpenAPI Specification allow developers to describe, produce & visualize APIs independent of their implementation. Older approaches like SOAP or WSDL have mostly been replaced by REST APIs which provide similar functionality and use existing infrastructure of the web like HTTP / HTTPS for communication.

Nevertheless modern media landscape requires more permeability, and increased flexibility in how we deal with information. That is why modern API designers came up with new concepts that go beyond the typical CRUD (Create, Read, Update and Delete) principal. Design trends like Event Subscriptions, Content Negotiation or On-Device APIs mirror what we actually use APIs for.

This paper will on the one hand sketch out the current state of the art in context of API development and design, and on the other hand show challenges in those design approaches. Beyond that, we want to point out some new trends and methods to address those challenges, in order to develop APIs for modern software systems like the voice-assistant Alexa.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management—*productivity, programming teams, software configuration management*

Keywords

API Development, Software Development, Software Architecture, REST, Spring Boot, GraphQL, Swagger, API Blueprint

1. INTRODUCTION

Modern software developers spend much of their time making design and architecture decisions. Designing an appropriate Application Programming Interface (API) that offers all the required functionality while providing proper documentation for its users is a challenging task for developers.

In recent years many techniques and tools have been developed to aid with API Development. However with the pure number of different standards, tools and conventions it is hard to stay on top of things. In this paper we try to give an overview over some of the concepts, tools and techniques available. We start by looking at how APIs are designed. There we talk about the so-called “API-First approach” which makes the API a ”first-class citizen” in a project. We then switch over to the implementation side of things and look at some key areas in API implementation like security, robustness and reliability. We also introduce the REST paradigm which many APIs implement today. After that we introduce some Tools and Frameworks like Spring Boot or Swagger that can help with various aspects of API development. Subsequently we look at some of the limitations present in current API development and the problems and challenges that arise from them. Finally we examine some future and upcoming trends that provide alternatives or improvements to existing technology like GraphQL. At the end we summarize our findings in a conclusion and think about some possible future areas that might be important to investigate.

2. STATE OF THE ART

To give an overview on state of the art, we will give a short introduction to design methods, best pratices and commonly used tools in API developoment. Thereby we will start with elaborating API design methods like the API-First approach. We will continue explaining some best practices in security and reliability and finally elucidate tools like Spring Boot or Swagger.

2.1 API design methods

Software developers and even whole companies are working on methods and guidelines to standardize APIs. These guidelines include more concrete approaches like “Public classes should not subclass other public classes for ease of implementation” [6], or some general behavior, like “A good API should be easy to use, even without documentation” [6].

Nevertheless, guidelines are always on a more concrete and technical level. Regarding methods in API development, many companies try to follow the so called “API-First approach”. API-First approach means that for any given development project, your APIs are treated as “first-class citizens”. That means everything about a project revolves around the idea that the end product will be consumed by other applications [31]. The API-First approach helps to organize the entire development process and enables multiple people to work on components in parallel. While older techniques forced front-end developers to wait for their back-end developer colleagues to finish the interfaces, API-First gives teams the ability to work against each other‘s public contracts without interfering with internal development processes [18]. This allows the developers to focus only on the business logic[30] instead of getting caught up in implementation details and by designing API-First, software developers are able to facilitate discussion with their stakeholders well before the projects complexity reaches a point where it is hard to make structural changes.

Another major factor in API development is the level of documentation. One thing you will observe in all enterprises who have made it big in API economy (i.e., Twitter, Expedia, etc.) is that their APIs are easy to use [30]. Those APIs have extensive documentation and are often supported with specific tools, like mentioned in 2.3.

Summarizing, API design methods are only a small part of the whole API development and architecture process, but have a huge influence on API development and quality. State of the art design methods allows the management of challenging tasks in modern software projects.

2.2 Implementation Best Practices

Just as important as the design of an API is the actual implementation. Especially when handling sensitive or private data you have to make careful decisions to keep your clients data safe.

2.2.1 API-Security

APIs are meant to be as open and visible as possible so that other applications and services can easily integrate them. However this openness also creates additional security risks as it provides new pathways for malicious entities to get into applications and systems [8]. While some of these issues, like bugs in operating systems or web servers, are out of the reach of an API developer, there are still some approaches you can take to make your API more secure.

It is important that you always assume someone wants to get access to your data [3]. Even if your API is not meant to be accessible from the outside you should still secure it as if it was. Any data that is private or sensitive must be hidden behind an Authentication and Authorization layer. A commonly used method for a client is to obtain some kind of API key beforehand that both authenticates and authorizes him. Standards like OAuth make this easier and allow users to grant third party services access to specific data. Especially there, developers should give attackers as little information as possible. When a client fails to authenticate, sending him a generic authentication error message gives out much less information than a “Wrong Username Error”.

Another fundamental security requirement is encryption, keeping authentication credentials safe during transit is essential. Keeping up to date with modern encryption algorithms is important as weaknesses in older security protocols are found all the time. In addition signing your API Requests and Responses allows both client and server to be sure that the data has not been tampered with. For HTTP based APIs using a modern version of Transport Layer Security (TLS) is an easy way to achieve this.

2.2.2 API Robustness

In computer science robustness refers to “the ability of a computer system to cope with errors during execution and cope with erroneous input.” [32]. Especially the second part is also a desired property of APIs since you generally have to deal with input that is controlled by clients.

Even if a client is properly authenticated and authorized you should still not trust the data in their requests. Always validate and sanitize the parameters and content of API Requests and discard any that do not conform to the rules of your API. Attacks like SQL Injection or Cross-Site-Scripting have long been known about but are still exploited today.

If you are logging parts of a clients request make sure that customer data is omitted from the log files and any data clients have control over is properly sanitized. Otherwise attackers might be able to coerce your server into executing arbitrary code through a so called log injection attack.

2.2.3 API Reliability

Even if your API is secure, that does not help if it is not accessible in the first place. To keep your API stable and usable even under load you can apply quotas to your APIs and throttle clients that exceed them. This also prevents clients from misusing your API. Quotas can also be used as a business model, charging users based on the amount of API requests they make is a business model that companies like Google use. Depending on the nature of your API caching can also greatly decrease the requests that actually reach your API. Having one or multiple backup APIs that traffic gets redirected to as necessary can help with traffic spikes. [3]. If applicable a container based approach like Docker allows you to easily scale your system if demand increases.

2.2.4 REST

Many Web APIs you will find today call themselves REST (Representational State Transfer) APIs. To be called Restful an API has to comply with six constraints:

1. There has to be some kind of client-server relationship. Unlike SOAP APIs where clients are tightly coupled to their specific server, REST clients and servers should be able to evolve independently of each other [10].

2. The API has to be stateless, each request from a client must contain all the information the server needs to respond to it. This keeps the session state entirely client side which allows for easy scaling. It also makes the API reliable as it is easier to recover from partial failures.

3. Data in API responses should be implicitly or explicitly marked as cacheable or non-cacheable by the client. Allowing a client to cache data can reduce or even eliminate the API requests it has to make and therefore improve efficiency and performance.

4. The API is required to have a uniform interface, REST imposes four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state [10].

   Individual resources are identified in the request, for example using the URL in web based REST APIs. The actual resource on the server is different from the representation that is returned to the client. A server might return data from their database in any kind of format like JSON or XML none of which necessarily match the actual data stored. Though if a client holds a representation of a resource, that has to be enough information for it to modify or delete it.

Data returned by the server is also required to have some kind of media type attached to it so the client can decide how to handle or display it on its own.

A REST client should also be able to discover all methods and resources a server has to offer from an initial URL. The server should respond to requests with links to additional resources and actions that are available. This will be further elaborated on in the "Hypermedia and HATEOAS" chapter of this paper.

5. The API should be layered, each component should not be able to "see" beyond the layers it directly interacts with. While this does add some overhead it further improves scalability and allows independent development of components. Since the client doesn't know whether it is talking directly to the API or some intermediary this can also be used to implement load balancing or enforce security policies.

6. The last constraint is called "code-on-demand". Servers should be able to send their clients executable code to extend or customize their functionality. This simplifies clients as less features have to be implemented in them and allows for easy extendability. Since not every API requires this kind of customizability and it decreases visibility of the API itself this constraint is optional and might even only be applied to parts of the system.

Much of the required infrastructure for a REST API is already available in the WWW in the form of web servers and HTTP clients, a server serving a static website to a client is effectively RESTful. That makes REST a great fit for Web APIs as it is easy to implement and already available technology like HTTP or SSL can be used for REST APIs.

2.3 Tools and Frameworks

As software systems get more complex and more powerful all the time, software engineers develop tools and frameworks to support API development. As there are many different disciplines in API development, there are various tools. Some of these tools will be introduced in the following chapter.

2.3.1 Spring Boot

The Spring Framework is an application framework for the Java platform. The framework's core features can be used by any Java application. Up to this the Spring Framework is open source, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform, for which the source code is not available [35]. One of the extensions build upon Spring is Spring Boot, a convention-over-configuration solution for creating stand-alone, production-grade Spring-based Web Applications [35].

As Spring Boot is a convention over configuration framework, developers using this framework are required to make few decisions, while there is nearly no loss of flexibility. Convention over configuration itself is a software design paradigm, where a developer only needs to specify unconventional aspects of their application. When the convention implemented by the tool matches the desired behavior, it behaves as expected without having to write configuration files.

Spring Boot provides a simple and fast way to set up, configure, and run both simple and complex web-based applications. It chooses dependencies, auto-configures all the features and and allows the developer to quickly create a runnable application. Furthermore, it also simplifies the deployment process of applications [21]. Spring Boot offers the option to package the whole application as a JAR-file, including all required components like for example the web server.

Listing 1: Spring Boot example

```
 1  @RestController
 2  @RequestMapping("/events")
 3  public class EventController {
 4
 5      @GetMapping(path = "/{eventId}")
 6      public Mono<EventDetails> getEventDetails(
 7          @PathVariable(value = "eventId") String eventId)
 8      {
 9          return adapter.getEventDetails(eventId);
10      }
11  }
```

Listing 1 shows a simple example for a Spring Boot class. The code is written in Java and describes an API Route to look up details about an event. All necessary imports for underlying frameworks are managed through Spring Boot itself. The class is marked as a "@RestController", in line 1, and handles the "/events" Route, as defined by the "@RequestMappings" annotation, in line 2. In the class a method is implemented that handles the sub route for event details. Through the use of curly brackets "eventId" is defined as a parameter for this request. Spring will call this method with the parameter extracted from the URL. The EventDetails returned by the method will automatically get serialized to JSON, embedded in a http response and sent back to the client.

Spring Boot is one example for a category of software frameworks, which all have the purpose of supporting APIs based on REST. There are several more frameworks, like Django, Google Web Toolkit, JSF or Node.js, which all support creating REST interfaces, with Spring Boot being one of the more popular ones.

2.3.2 Swagger

Swagger is a synonym for two different concepts related to API development. One of those concepts is the "OpenAPI Specification", originally known as the "Swagger Specification", which is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services [34]. Applications implemented based on OpenAPI interface files can automatically generate documentation of methods, parameters and models. The OpenAPI Specification is language-agnostic, therefore it is possible to adapt it into new technologies and protocols beyond HTTP [14].

The other meaning of Swagger is an open-source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful Web services [36]. This software framework has been created, to simplify writing and documenting APIs. The swagger tooling differentiates primarily between three use cases.

The main one is developing APIs, where Swagger may be used to automatically generate an Open API document based on the code itself. Alternatively, using Swagger Codegen [12], developers can decouple the source code from the Open API document, and generate client and server code directly from the design.

The second use case, Swagger is meant for, is interacting with APIs. Using the swagger-codegen [12] project, end users generate client SDKs directly from the Open API document.

The last use case is documenting APIs, whereby Swagger open source tooling may be used to interact directly with the API through the Swagger UI [13]. This project allows connections directly to live APIs through an interactive, HTML-based user interface. Requests can be made directly from the UI and the options explored by the user of the interface.

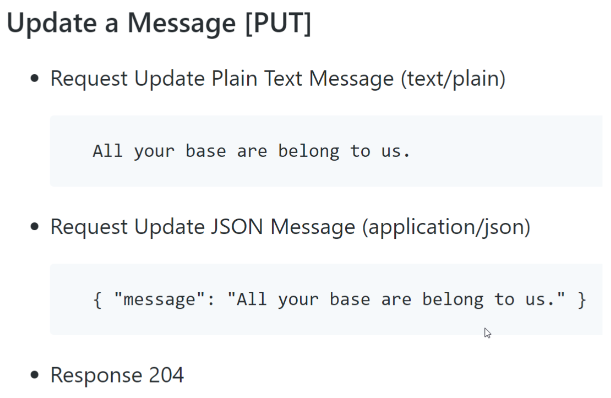
2.3.3 API Blueprint

API Blueprint is a high-level API description language for web APIs. In API Blueprint you can define API templates as ".md" files, using an extended Markdown-like syntax. Those files will later on be translated to HTML documents, that visualize the API structure in a clear way. The blueprints themselves start with a metadata section, where you can define for example the used format. Headings start with one or more "#" symbols followed by a title. Actions can for example be defined by "+". Listing 2 shows a simple example for a blueprint.

While description languages like JSON are often confusing about their structure, API Blueprint is simple and accessible to everybody involved in the API lifecycle [4]. Its format is far more human readable than JSON and can even be used to create server mocks, which are invaluable in testing service ecosystems [18]. API Blueprint is built to encourage dialogue and collaboration between project stakeholders, developers and customers at any point in the API life cycle. Furthermore it is completely open sourced under the MIT license allowing anybody to use or modify it.

A valuable design feature of API Blueprint is, that it is built to encourage better API designs through abstraction. The goal of API Blueprint is to decouple elements of the API to enable modularity while encapsulating back-end implementation behavior. It is designed to follow the API-First approach. Similar to tests in test-driven development, API Blueprint represents a contract for an API.

Listing 2 describes an example for a simple API defined by API blueprint. Line one defines the HTTP method, that is used for calling the API, in this case a PUT method. Line three and five define the possible parameters, which are in this case plain text or JSON format. At last the return value is given inside the response part. API Blueprint is able to translate this format to an HTML document. The resulting HTML document for Listing 2 can be seen in Figure 1. The example above is one simple case, but API Blueprint can also be used to represent more complex data structures. As data structures get more complex API Blueprint automatically generates the corresponding JSON structure used in the implementation. The exported HTML page will provide the visualization for this underlying structure.


Update a Message [PUT]

- Request Update Plain Text Message (text/plain)

```
All your base are belong to us.
```

- Request Update JSON Message (application/json)

```
{ "message": "All your base are belong to us." }
```

- Response 204

Figure 1: HTML result for API Blueprint example

Listing 2: Simple API example defined in API Blueprint

```
1 ### Update a Message [PUT]
2
3 + Request Update Plain Text Message (text/plain)
4         All your base are belong to us.
5 + Request Update JSON Message (application/json)
6         { "message": "All your base are belong to us." }
7 + Response 204
```

2.3.4 Apiary

Apiary is a sub-tool from Oracle, which means that it is a tool, designed to support another tool, in this case API Blueprint. One of its main features is generating server mocks out of API Blueprints. The combination of API Blueprint and Apiary allows developers to design APIs as blueprints, mock those APIs and run automated tests as well as validations. Since all of this is possible without writing any implementation code the whole development team can use the back-end immediately without waiting for it to be finished. Apiary is also bilingual, instead of API Blueprint you can also use the Swagger Specification for your server mocks.

3. CHALLENGES AND LIMITS IN CURRENT DESIGN

The world of API development is one of constant evolution. The magnitude of data we dealt with in the 1990s is not comparable to the one we deal with in 2018, and because of this, the ways APIs are designed had to change and new approaches had to be found. There are many issues in the API architecture space that we are only just now becoming aware of [26].

APIs are largely evolving, as a result, many of the issues in the modern API space derive from the fact that developers have been effectively putting a band-aid over the problems of past implementations and approaches.

In the 1990s, the concept of Distributed Object Integration (DOI) reigned supreme, allowing resources to be collated and linked to one another for users to more fully manipulate and integrate with.

Following this, in the early 2000‘s, the movement to and support for SOAP for corporation-to-corporation integration and resource sharing made DOI-style systems somewhat obsolete. SOAP reigned on high for many use cases, but had significant issues that required further systems to be developed, iterated upon, and implemented in the foreseeable future [26].

With the coming of the 2010‘s, APIs shifted away from stateful APIs designed for network stacks and into the concept of stateless design with a focus on mobile data integration. REST was a great fit since it could easily be implemented using the existing HTTP infrastructure and provided basic CRUD operations. This makes it easy to use, understand and write services against it. REST also makes efficient use of bandwidth, as it is less verbose than SOAP. Unlike SOAP, REST is designed to be stateless, and REST requests can be cached for better performance and scalability. Smartphones, once thought to be a limited field of telecommunications, are nowadays micro-supercomputers, and our data solutions had to appropriately scale to meet this increased demand.

RESTful design was quickly taken to by developers as a standard. APIs in general are thereby designed for CRUD actions (Create, Read, Update, Delete). In the regular process of an API, the user requests a resource be created, that they read a resource, that a resource is updated, or that the resource is deleted. Every single function a CRUD-centric API will do is a simple combination of those four elements, and nothing more [26]. This was fine before, but the modern media landscape requires more permeability, and increased flexibility in how information is dealt with. Because of this, the CRUD model is inefficient because it does not mirror what we actually use APIs for anymore.

As an example for the limitations of CRUD, imagine calling a voice-assistant like Amazon‘s Alexa with “Alexa, add my reminders for today into the calendar.”. Interfacing with Alexa in such a way requires collations of resources and AI-like functions to discern relevance, interest, and value - this is not something easily done with a CRUD system, which is why CRUD is not the design structure of the frontend systems that drive Alexa. It is simply not enough to make an API call, with one of the CRUD functionalities. If the user for examples says , first of all several voice recognition functionalities need to be called. Up to this the API would need to perform application comprehensive tasks. A simple CRUD functionality would only be able to perform one of the tasks, like the CREATE action, which would be adding a calendar entry in this case. A complex combination of tasks wouldn’t be possible.

The APIs available in the market all have different standards regarding stability, reliability and quality; not all are equally safe, and in some cases a few may become gateways for hackers [29]. Furthermore there is another major challenge for APIs. They can either be developed as closed “partner-only” APIs, where a company announces with some fanfare that it now has APIs, but you can not use them unless you are very important. The other situation is that a company launches a technically great API, but does so with no developer business model. The closed-API model ends up missing the big opportunities because, it relies on handcrafted partnerships that inevitably try to pick winners and losers before a line of code has been written by quickly becoming about things like exclusivity and co-marketing [5]. In many ways, having a poorly thought out API monitization model is even worse than not having one at all, because it does not provide sustainable incentives and rewards to either the API provider, the developer or the user.

Monetization has an impact on API design, because for the different business models different design approaches need to be implemented. If your API is designed open, you need to focus on robustness, because theoretically everybody can use your API. Also your servers need to be scalable, to handle possible traffic. In comparison to that, a closed API can have difficulties in its authentication, because you need to define authentication methods for the clients that should be able to access your API.

Furthermore there are some timeless challenges, that any developer has to face during API development. One of those is the problem of enhancing and versioning APIs. A version change usually indicates a milestone in the codebase of the API. It declares a change in the requirements of API consumption and implementation. Often, these new versions become whole new products. Although they share a common ancestry, new versions of legacy APIs require careful thought about their implementation.

Changes that can warrant a new version include for example removing an operation, renaming an operation and complex structural changes of data types. A version increment can also indicate significant changes to API consumption requirements and therefore require changes in the underlying resources offered by the API.

One school of thought is to focus on one unchanging URI with just one set of criteria for consumption. If the API structure is changed, resources altered, or parameter set modified, then the product is re-launched with the same URI. This pushes the obligation to refactor code to downstream developers.

Another consideration is backwards compatibility. For many providers of web resource APIs, this is the primary consideration. Maintaining multiple versions of a resource intensive API can be a serious drain on the time and focus of engineering teams. It can also introduce long term stability problems to services that have moved on to more modern architectures.

Both measures aren’t ideal to solve the problem of versioning. Also all the other mentioned challenges have no optimal solution. This is the reason, why there are several design trends, to adapt APIs to modern software systems and web standards. Based on the problem, that there is not the one ideal solution, it is onto developers to decide between and even combine those design approaches.

4. FUTURE API DESIGN TRENDS

The following chapter elaborates some trends in API development. We will thereby present some more general concepts, like HATEOAS or Event Subscriptions, and explain languages like GraphQL.

4.1 GraphQL

GraphQL is a query language for APIs, developed internally by Facebook in 2012. It provides a flexible approach for developing web APIs using a type system you define for you data. It is not made with a specific database or storage engine in mind and can instead be adapted to work with your existing code and data set. As seen in listing 3 the data returned by the API has the same shape as the query. This is essential to GraphQL, because you always get back what you expect, and the server knows exactly what fields the client is asking for [16]. It therefore preventing excessively large amounts of data from being returned, but this has implications for how effective web caching of query results can be.

Where standard, REST based, APIs have endpoints to several resources, GraphQL is kind of a layer in between, which organizes the traffic and provides a single endpoint. This endpoint can take in complex queries, and then fit the data output into whatever shape the client requires. Basically the GraphQL layer lives between the client and one or more data sources, receiving client requests and fetching the necessary data according to the developers instructions.

A misunderstanding that often occurs is the belief that GraphQL is a better REST. Comparing the two is not appropriate in the first place, because REST is an architectural concept for network-based software, has no official set of tools, has no specification and is designed to decouple an API from the client. The focus is on making APIs last for decades, instead of optimizing for performance. GraphQL is a query language, specification, and collection of tools, designed to operate over a single endpoint via HTTP, optimizing for performance and flexibility.

In order to provide this performance and flexibility, GraphQL queries access not just to the properties of one resource but also smoothly follow references between them. With typical REST APIs you might have to follow multiple URLs to get all your data, with GraphQL you might be able to get all data you want in one request.

One of the problems GraphQL tries to solve is versioning. Developers are able to add new fields and types to their GraphQL API without impacting existing queries. This is possible, because with GraphQL fields can be deprecated and hidden from tools. By using a single evolving version, GraphQL APIs give apps continuous access to new features and encourage cleaner, more maintainable server code [15].

Listing 3: GraphQL Example

```
Query:
{
  hero {
    name
  }
}

Response:
{
  ”data”: {
    ”hero”: {
      ”name”: ”R2−D2”
    }
  }
}
```

4.2 Hypermedia and HATEOAS

Hypertext extends normal text by adding references (hyperlinks) to other texts that the reader can follow. “Hypertext Markup Language” or HTML is the most prominent example of a hypertext based language. Hypermedia extends Hypertext further, linking all types of media like images, videos audio and plain text together. The whole WWW is one example of Hypermedia [33].

4.2.1 Influence on APIs

In APIs Hypermedia creates tangible links between multimedia and relevant resources [22]. Like described in the REST chapter “Hypermedia As The Engine Of Application State” (HATEOAS) is actually required for an API to be truly called RESTful. HATEOAS allows a REST client to discover the functionality of an API server from a single URL by following hypermedia links provided by the server. If the client does not know how to handle a media type it encounters, the server might even be able to send the client code that allows him to do so through the code-on-demand REST construct. Still many APIs that consider themselves RESTful do not implement Hypermedia and instead provide documentation a client has to be built upon.

4.2.2 Hypermedia Types for Web APIs

There are many different standards available for Hypermedia types, in this chapter we will talk about “Structured Interface for Representing Entities” (Siren). Unlike many other hypermedia types you can also use Siren to define actions a client can take to alter the application state [27].

Listing 4: Siren actions example

```
1  GET https://api.com/user/2314
2
3  {
4      ”class”: ”user”,
5      ”links”: [
6          {”rel”: [”self”], ”href”: ”https://api.com/user/2314”}
7      ],
8      ”actions”: [
9          {
10             ”name”: ”change−image”,
11             ”title”: ”Change image”,
12             ”method”: ”POST”,
13             ”href”: ”https://api.com/user/2314/image”,
14             ”type”: ”image/png”,
15             ”fields”: [
16                 { ”name”: ”image”, ”type”: ”file” }
17             ]
18         }
19     ],
20     ”properties”: {
21         ”id”: ”2314”,
22         ”name”: ”Max Mustermann”,
23         ”nickname”: ”mustermax”,
24         ”image”: ”https://api.com/user/2314/image.png”,
25         ”dateCreated”: ”2003−11−12”
26     }
27 }
```

First of all, each siren entity may have one or more classes (Line 4) that describe the entities content. What classes exist is implementation-dependent and should be documented.

Properties (Line 20) are a set of key-value pairs that describe the state of an entity.

Links (Line 6) represent navigational transitions. They are not used to describe entity relationships. Each link should have a “rel” property for describing the relationship of the link to its entity and a “href” property to point to the target URI. You can also specify a “type” property to define the media type of the linked resource. Every siren entity should at least have a self link pointing to itself. One example of using links is a list structure where each element has a “next” and “before” element.

Actions (Line 8) describe what behaviour an entity exposes. The only required properties are the “name”, that has to be unique, and “href” which defines the URI of the action. “Fields” are effectively the parameter the action has and “type” the way these parameter are encoded in the request. [28]

Listing 5: Siren sub-entities example [28]

```
1  {
2    ”class”: [ ”order” ],
3    ”properties”: {
4        ”orderNumber”: 42,
5        ”itemCount”: 3,
6        ”status”: ”pending”
7    },
8    ”entities”: [
9      {
10       ”class”: [ ”items”, ”collection” ],
11       ”rel”: [ ”http://x.io/rels/order−items” ],
12       ”href”: ”http://api.x.io/orders/42/items”
13     },
14     {
15       ”class”: [ ”info”, ”customer” ],
16       ”rel”: [ ”http://x.io/rels/customer” ],
17       ”properties”: {
18         ”customerId”: ”pj123”,
19         ”name”: ”Peter Joseph”
20       }
21     }
22   ]
23 }
```

Entities can also have sub-entities assigned to them, they can either be referenced via a link like in Line 10 or directly embedded in the json like in Line 15.

4.2.3 *Hypermedia API Examples*

Listing 6: FamilySearch example [28]

```
−− GET /platform/tree/persons/PPPJ−MYZ
...
<person id=”PPPJ−MYZ”>
<link rel=”ancestry” href=”...”/>
<link rel=”artifacts” href=”...”/>
<link rel=”child−relationships” href=”...”/>
<link rel=”children” href=”...”/>
...
<living>true</living>
<gender type=”http://gedcomx.org/Male”/>
<name type= ... id=”name−id”>
    ...
    <preferred>true</preferred>
    <nameForm>
    <fullText>Alex Aleksandrova</fullText>
    ...
    </nameForm>
</name>
<fact type=”http://gedcomx.org/Birth” id=”born”>
    ...
    <date>
        <original>3 Apr 1836</original>
        <formal>+1836</formal>
    </date>
    <place>
        <original>Moscow, Russia</original>
        ...
    </place>
</fact>
...
```

One example for a Hypermedia driven API is the API of FamilySearch. In the example, instead of just returning information about the person it also provides links to other types of information that is related to that person like their ancestry or their children. Linking different sources of data is important when you try to generate new information from otherwise unlinked data [22]. FamilySearch uses the atom hypermedia types to represent these links in xml.

4.3 Event Subscriptions

In this type of design approach, the usual interaction between the API and the user is reversed. In a traditional architecture, the user comes to the API, makes a request, and then voluntarily terminates that interaction. The actual relationship between the user and the provider is terminal and temporary, and as such, is expressly limited [26]. The event subscription architecture inverts this relationship. A relationship is not longer temporary, but the user subscribes to the information and its future updates.

This type of architecture design forces the user to subscribe to an entity. Every change to the entity will then trigger changes in the metadata and will be provided via the data resource. The API in this case serves two primary functions: First, the API serves the function of actually delivering the resources to the requesting user, and second, the API serves as a message broker to collate publisher additions and update those who have subscribed [26].

For this reason, event-driven architectures are very popular, and lead to improved power, bandwidth, and co-processing than other solutions and architectures such as polling and other poll-centric derivatives [23]. Because modern software systems differ in their architecture there are several protocols to use. The following five protocols are examples for event-driven APIs.

4.3.1 *WebSockets and WebHooks*

Essentially, WebSocket is a protocol that provides full-duplex communication on a single TCP connection. It was standardized by the Internet Engineering Task Force as RFC 6455, and the WebSocket API in Web IDL was later standardized under the W3C banner [23]. Another functionality, WebSockets offers is, that its protocol can be used by any case where a client-server relationship is present, not only with web browsers and servers, where it was originally meant for.

Because WebSocket is expressly designed for browser operation, it boasts extremely low overhead for what it actually does. By establishing a full-duplex conversation using a standardized methodology, connection both to and from the two entities can take place simultaneously, resulting in lower overhead and better throughput [23]. Against that WebSockets have one distinct disadvantage: While it might have support for HTTP-like functionality, it is not HTTP. This has implications, especially when considering optimizations in HTTP such as caching, proxying, etc., that haven‘t quite become apparent. Because WebSockets are relatively new, having been only officially standardized in 2011, the industry is still understanding what the side effects mean.

Perhaps the strongest argument for the use of WebSockets are the fact that they are natively supported by all major browsers [2]. This means that any web application that ties into it will be intractable within the vast majority of both browser-based and browser-independent gateways and applications.

4.3.2 REST Hooks

REST Hooks is essentially “hooking” baked into REST itself. Defined as an initiative from Zapier, hooks are collated to a single target URL as a subscription, which pings the resource requester when a change is noted. This approach is a response to the practice of polling, in which a client constantly checks for changes to a resource. Under the REST Hooks paradigm, the client instead waits for a change, and reacts to it.

REST Hooks are very powerful, being able to passively receive a resource rather than dedicating processing power to constant polling frees up a lot of the client-side cost. Perhaps the strongest argument for REST Hooks though, is the fact that it‘s so easy and intuitive to use. While WebHooks utilize HTTP and thus do not need new architecture to set up, they are also limited by the fact that they are built upon HTTP, and can thus be somewhat complex to set up properly and use effectively.

4.3.3 Pub-Sub

Pub-Sub is a slightly different approach. Referred to by its full name as publish-subscribe, the concept is where events are published to a class without knowledge of the client subscribing to the class. Basically, a user will join one or more classes, and then will receive event updates without regard or knowledge to the event publisher. The main difference here is one of conscious choice of provider, where a user consciously communicates with a given server or provider and receives events as pre-determined. Under the Pub-Sub scheme, the user only specifies which class they wish to be part of and what events they are interested in receiving. From there, they receive these events when one is pushed out [23].

A huge benefit of Pub-Sub is the fact that it‘s loosely coupled, and thus is extremely scalable and flexible. The only thing the event-provider is doing is generating the content, each other step is done through a separated middleman, and so the content is easily scaled and modulated to the architecture and design of the solution. Decoupling is also a huge disadvantage for this pattern. By being a middleman, Pub-Sub cannot effectively notify the provider that a message has been sent, and the listener is separated from the event and thus may not know if a message wasn‘t sent that should have been.

4.3.4 Server Sent Events

Server Sent Events, or SSE, is a communication protocol much like WebSockets, but with the implication of unidirectional data. In this architecture, the server is consistently sending updates to the client as an automatic process. This was standardized under HTML5 by the W3C, and is thus compatible with any solution that is likewise compatible with HTML5 [23].

SSE is not bidirectional in its communications, so the server is issuing the events in a steady, predictable method. This is hugely beneficial to applications which do not need the two-way communications baked into WebSockets or other such solutions, as this means lower bandwidth, and an allowance for the connection to be temporary rather than always-on during the duration of data transfer. That simplicity could be where SSE fails for particular use cases. SSE is a very poor solution for situations that require bidirectional communication, and while this seems obvious, it would surprise many developers to see how many systems actually depend on bidirectional communication for simple functionality.

4.4 Content Negotiation

As a definition, content negotiation is a process in which a given set of possible input variations is narrowed down to a singular likely input that is the best possible representation of that request for the requesting entity. The content negotiation mechanism allows clients to select different representation formats from the same resource URI. It‘s a mechanism defined in the HTTP protocol (RFC 7231) [9].

Listing 7: Language Negotiation example

```
Accept-Language: de; q=1.0, en; q=0.5
Accept: text/html; q=1.0, text/*; q=0.8,
image/gif;q=0.6, image/jpeg; image/*;
```

Listing 7 describes a simple example, why content negotiation is an interesting field in API development. The browser has been configured to accept German and English, but prefer German, and to accept various media types, preferring HTML over plain text or other text types, and preferring GIF or JPEG over other media types, but also allowing any other media type as a last resort. As a result of that, with correct usage of content negotiation, developers can enrich the diversity of result types. While a simple interface can only return objects and information of a single kind, content negotiation based system are more flexible.

Content negotiation is a technique that has been around for decades, but has not been adopted by many REST APIs that serve structured data or media. When file formats started to change very fast in the web, content negotiation became more popular, because it is a way to design long-lasting APIs that adapt to an ever changing world of file formats. Content negotiation allows a user to determine which media types they prefer to receive from the server.

4.4.1 *Technology in content negotiation*

Listing 8: GET Request example

```
GET /user/avatar/nordicapis.png

Accept: image/png,
 image/jpeg; q=0.8,
image/gif; q=0.8,
 image/*; q=0,5
 application/json; q=0.1
```

Listing 9: GET Request example adapted for content negotiation

```
GET /user/avatar/nordicapis

Accept: video/avi; q=0.8,
 video/mov; 8=0.5,
video/*,
 image/gif; q=0.8,
 image/*; q=0.5,
 application/json; q=0.1,
```

The following scenario describes a situation where the user wants to retrieve an avatar photo. In Listing 8 the RFC says that multiple formats may be specified. Because these are later all accepted, only requesting a PNG becomes redundant. Using an accept header, the developer can drop the PNG from the resource name, and replace it with a list of acceptable image types.

Listing 8 presents the situation. The request type is not longer necessary, as the developer can define specific accept types. When the application using the API in listing 8 releases a new feature, as also allowing video content, it would raise errors on client side. An content negotiation approach is presented in listing 9. In this example the URI remains constant, and now all the client has to do is say that they accept videos in the HTTP accept header.

Furthermore, with the help of content negotiation, developers have another approach to face the versioning challenge. With content negotiation developers can design their APIs more open and allow clients to receive a larger range of return types.

4.4.2 *Examples from the real world*

Travis CI, a distributed continuous integration service used to build and test software projects hosted at GitHub, stopped serving a PNG file. But, for backwards compatibility, the SVG version will be returned even if you request PNG or SVG. When format adoption changes, developers may end up returning a different file format from what is technically requested.

WebM and Webp are alternative file formats that have been pushed by Google. Though Google supports the formats within their own apps, they mainly took care on their own environments. However, because Google has a huge influence in the web, software developer will also need a way to evolve their APIs for those formats.

As another example of format evolution on a trusted API-first platform, in June 2014 Twitter began supporting animated GIFs. In 2015 GIFs were adopted into JSON payloads as well. The way they support it is by taking a GIF, and converting it into a video (Mp4) for compression benefits.

5. CONCLUSION AND FUTURE WORK

API development as a process has been iterated upon many times throughout the years and many different techniques and tools have arisen out of it as a result. One mayor historical milestone was the emergence of REST and as a consequence thereof a standard in designing web APIs.

A problem with this is that those REST constraints, as presented in 2.2.4, are not all covered in every REST API. In fact most of the modern APIs implement only a subset of those constraints and is therefore more Rest-like than RESTful.

With REST becoming more and more known, also several tools got developed and published to support creating APIs. Because REST supports the already described CRUD principle, many APIs are limited by it in their functionality. As a result of that, it is getting increasingly complicated to provide all the necessary information for modern client systems and to fit modern software development structures.

As a result, CRUD is no longer always the appropriate architectural approach like it used to be. The internet is evolving, and so are the needs of the average user. With this evolution of needs, API developers need to start considering additional methodologies and design approaches. In the future, failing to do so may result in the spread of stopgaps designed to make CRUD do things that it is not really meant to do.

Parallel to the development of tools and frameworks, that should help developers to write APIs, several new design trends have been initiated. Although some of them offer solutions for specific challenges, none of them are able to solve all difficulties. Approaches like content negotiation offer a possibility to define a larger range for return types, but with major releases in APIs, versioning is still a problem.

In conclusion, the presented design trends offer strong solutions for specific problems, and are promising to say the least. It will be interesting to see what these designs ultimately create, and whether or not they will be able to fully break free of the CRUD archetype. For future investigations it might be interesting to see if CRUD is still applicable for these modern use cases.

6. REFERENCES

[1] `https://www.familysearch.org/developers/docs/api/resources`.

[2] Can I use Websockets. `https://caniuse.com/#feat=websockets`. sighted 01.2019.

[3] State of API Security. `https://www.soapui.org/learn/security/state-of-api-security.html`.

[4] API Blueprint. A powerful high-level API description language for web APIs. `https://apiblueprint.org/`, 2018. sighted 11.2018.

[5] E. Anuff. Almost everyone is doing the API economy wrong. `https://techcrunch.com/2016/03/21/almost-everyone-is-doing-the-api-economy-wrong/?guccounter=1`, 2015. sighted 11.2018.

[6] J. Bloch. How to Design a Good API and Why it Matters. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/32713.pdf, -. sighted 11.2018.

[7] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web. *IEEE INTERNET COMPUTING*, pages 86–93, March 2002.

[8] T. Despoudis. 7 API Security Best Practices. https://www.twistlock.com/2018/06/04/7-api-best-practices/, 2018. sighted 11.2018.

[9] B. DOERRFELD. Content Negotiation For Web API Longevity. https://nordicapis.com/content-negotiation/, 2015. sighted 11.2018.

[10] R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000. sighted 11.2018.

[11] M. Fowler. Richardson Maturity Model. https://martinfowler.com/articles/richardsonMaturityModel.html, 2010. sighted 11.2018.

[12] Github. Swagger Code Generator. https://github.com/swagger-api/swagger-codegen/blob/master/README.md, 2018. sighted 11.2018.

[13] Github. Swagger UI. https://github.com/swagger-api/swagger-ui/blob/master/README.md, 2018. sighted 11.2018.

[14] Github. The OpenAPI Specification. https://github.com/OAI/OpenAPI-Specification/blob/master/README.md, 2018. sighted 11.2018.

[15] GraphQL. GraphQL. https://graphql.org/. sighted 12.2018.

[16] GraphQL. GraphQL Fields. https://graphql.org/learn/queries/#fields. sighted 12.2018.

[17] O. Hartig and J. PÃľrez. An Initial Analysis of FacebookâĂŹs GraphQL Language. https://liu.diva-portal.org/smash/get/diva2:1141747/FULLTEXT01.pdf, -. sighted 11.2018.

[18] K. Hoffman. An API-first approach for cloud-native app development. https://www.oreilly.com/ideas/an-api-first-approach-for-cloud-native-app-development, 2016. sighted 11.2018.

[19] Y. Y. Hung. Writing, Mocking and Testing API document with API Blueprint, aglio, drakov and dredd. https://blog.darkcl.tech/2018/03/01/Writing-Mocking-and-Testing-API-Document-with-API-Blueprint-aglio-drakov-and-dredd/, 2018. sighted 11.2018.

[20] A. Parecki. OAuth 2 Simplified. https://aaronparecki.com/oauth-2-simplified/, 2016. sighted 11.2018.

[21] Z. Raffai. What Is Spring Boot? https://dzone.com/articles/what-is-spring-boot, 2018. sighted 11.2018.

[22] K. SANDOVAL. How to Improve API Experience Using Hypermedia. https://nordicapis.com/improve-api-experience-using-hypermedia/, 2016. sighted 11.2018.

[23] K. SANDOVAL. 5 Protocols For Event-Driven API Architectures. https://nordicapis.com/5-protocols-for-event-driven-api-architectures/, 2017. sighted 11.2018.

[24] K. SANDOVAL. How Could Artificial Intelligence Improve API Design? https://nordicapis.com/could-artificial-intelligence-improve-api-design/, 2017. sighted 11.2018.

[25] K. SANDOVAL. Securing Medical IoT Devices. https://nordicapis.com/securing-medical-iot-devices/, 2017. sighted 11.2018.

[26] K. SANDOVAL. 7 Growing API Design Trends. https://nordicapis.com/7-growing-api-design-trends/, 2018. sighted 11.2018.

[27] K. Sookocheff. On choosing a hypermedia type for your API - HAL, JSON-LD, Collection+JSON, SIREN, Oh My! https://sookocheff.com/post/api/on-choosing-a-hypermedia-format/, 2014. sighted 11.2018.

[28] K. Swiber. Siren: Structured Interface for Representing Entities, super-rad hypermedia. https://github.com/kevinswiber/siren, -. sighted 11.2018.

[29] UNKOWN. What are APIs and what challenges do APIs face? https://blog.signaturit.com/en/what-are-apis-and-what-challenges-do-they-face, 2017. sighted 11.2018.

[30] K. Viswanathan. The Basics of API-Driven Development. https://dzone.com/articles/abcs-of-api-driven-development, 2017. sighted 11.2018.

[31] J. Wagner. Understanding the API-First Approach to Building Products. https://swagger.io/resources/articles/adopting-an-api-first-approach/, 2018. sighted 11.2018.

[32] Wikipedia. Robustness (computer science). https://en.wikipedia.org/wiki/Robustness_(computer_science). sighted 12.2018.

[33] Wikipedia. Hypermedia. https://de.wikipedia.org/wiki/Hypermedia, 2018. sighted 11.2018.

[34] Wikipedia. OpenAPI Specification. https://en.wikipedia.org/wiki/OpenAPI_Specification, 2018. sighted 11.2018.

[35] Wikipedia. Spring Framework. https://en.wikipedia.org/w/index.php?title=Spring_Framework&oldid=872920248, 2018. sighted 11.2018.

[36] Wikipedia. Swagger (software). https://en.wikipedia.org/wiki/Swagger_(software), 2018. sighted 11.2018.

[37] D. Winer. XML-RPC Specification. http://xmlrpc.scripting.com/spec.html, 1999. sighted 11.2018.

What is the Added Value of Enterprise Architecture?

A Systematic Literature Review

Barry-Detlef Lehmann
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
barry.lehmann@rwth-aachen.de

Felix Tomski
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
felix.tomski@rwth-aachen.de

ABSTRACT

Back in the days, Information Systems were disconnected from the actual enterprise infrastructure. Computing was only used for automating simple processes, but with the growth of technology the need of a systematic inclusion of Information Systems (IS) and Information Technology (IT) infrastructure became more and more essential. Because IS and IT were not aligned with the overall business goals, potential benefits and efficiency were not realized. Zachman et al. was one of the first who became aware of this problem in 1987 and designed a framework for IS architecture. Out of this evolved *Enterprise Architecture* (EA), which gives a rich set of principles, methods, and models to bring organizational structure, business processes, information systems, and infrastructure together.

In the last few decades the relevance of enterprise architecture for (economic) organizations increased, and so did scientific research on this topic. Most of the research focuses on EA frameworks, methodologies and models, but there is only a small amount of research on the question, what actual value EA adds to a business and what benefits EA brings to a business. Since most papers which tackle these or similar questions are already outdated, we decided to approach these questions based on a systematic literature review to give an update on this specific EA research topic.

Categories and Subject Descriptors

D.2 [**Software**]: Information Systems; D.2.9 [**Software Engineering**]: Management—*productivity, programming teams, software configuration management*

Keywords

enterprise architecture, value, benefits, systematic literature review

1. INTRODUCTION

1.1 Motivation

With the innovation and evolution of technology, Information Technology (IT) and Information System (IS) became a more and more important role in businesses [37] [10] [36]. Advanced computer technologies were mostly used for business process automation [10]. Later on, the use of IT and IS became ordinary in most organizations, not only in the economy, but also in governmental organizations [20] and agencies [12]. Today's businesses must be agile, flexible and able to react to market changes quickly, in order to survive in the modern economic market and gain competitive advantages [35] [55]. Simultaneously, technologies evolve and change rapidly, as well as get more complex from day to day [2]. However, both, IT and IS, were not well integrated into the organization. But, when both, the IT and IS department of a business and the rest of it develop independently of one another, and they are not well integrated, the agility and flexibility is lost. One of the first who addressed this issue was Zachman in 1987, by proposing "a framework for information systems architecture" [57]. The main purpose of Zachman's framework is to make a link between things of the real world of the business and their representation in the computer systems. Out of this evolved the concept of *Enterprise Architecture* (EA).

1.2 Enterprise Architecture

To understand what EA is and how it adds value to an organization, we need to introduce several terms. So we start with the basic concept of what an "architecture" is.

1.2.1 Architecture

To handle the complexity of any vast organization or system, you need an architecture. Think of the analogy of building a house [26]. When you contact an architect to design your house, you discuss how rooms, staircases, windows etc., will be put together. You agree on a common base, on which the the architect will produce a detailed plan, which then can be further used by engineers and builders [26].

But what makes it so effortless to communicate such a plan? It might be because you share a common frame of references. You both know what a 'room','window' and 'staircase' is. You know their function and their relationship to each other. You both use a mental image of a house architecture [26]. This image or model introduces its major functions and relationships , and they are structured. It gives us an abstract design, that ignores many details. Details like which materials to be used will be discussed later.

To design enterprises we need a similar frame of references [26]. To create an overview of the structure of an organization, its business processes, their application support, and the technical infrastructure, you need to express their function and their relationships to each other [26]. So for this LR we use the ISO/IEC/IEEE FDIS 42010:2011 standard (ISO/IEC/IEEE 2011) definition of architecture:

> "*Architecture: fundamental concepts or properties of a system in its environment, embodied in its elements, relationships, and in the principles of its design and evolution.*"

Another important definition from IEEE standard is 'stakeholder':

> "*Stakeholder: an individual, team, or organisation (or classes thereof) with interests in, or concerns relative to, a system.*"

Most stakeholders of the system are not interested in its architecture, but how it affects their concerns. But an architect needs to be aware of these concerns and discuss them with the stakeholders, and therefor he should be able to explain the architecture to all stakeholders involved, which often have a different background [26].

But the term architecture can be applied in many scopes. The discipline of Enterprise Engineering sees enterprises as whole designed system that can be adapted and redesigned in a systematic and controlled way. In this context 'enterprise' according to TOGAF [18]:

> "*Enterprise: any collection of organisations that has a common set of goals and/or a single bottom line.*"

Architecture at the level of an entire organization is mostly referred to as 'enterprise architecture', which leads us to the next paragraph.

1.2.2 Enterprise Architecture

Even though, the proposition of Zachman et al. is now about 30 years old and thus, is the concept of Enterprise Architecture, there is still no uniform definition of EA [45]. A first definition of enterprise architecture was given by Richardson et al. [38], a few years after Zachman's proposition, in 1990. Richardson et al. define EA as "*a dynamic information technology foundation that provides a direction for the deployment and integration of future technological and managerial developments*". However, this definition does not include the importance of the operating model or overall business strategy of the organization and that the whole organization is to consider when talking about EA. Thus, in this paper, we will use one of the most common definitions of EA, suggested by [42]:

> "*The enterprise architecture is the organizing logic for business processes and IT infrastructure, reflecting the integration and standardization requirements of the company's operating model. The enterprise architecture provides a long-term view of a company's processes, systems, and technologies so that individual projects can build capabilities - not just fulfill immediate needs.*"

We chose this definition because it emphasizes that EA not only describes the current state of the business, but also includes the business state and goals for the future. Similar definitions can be found in the majority of literature on EA. Another mentionable definition, that is often used in EA research by the IT world [21] [53], is a general definition of architecture [1] standardized by IEEE Standard 1471-2000.

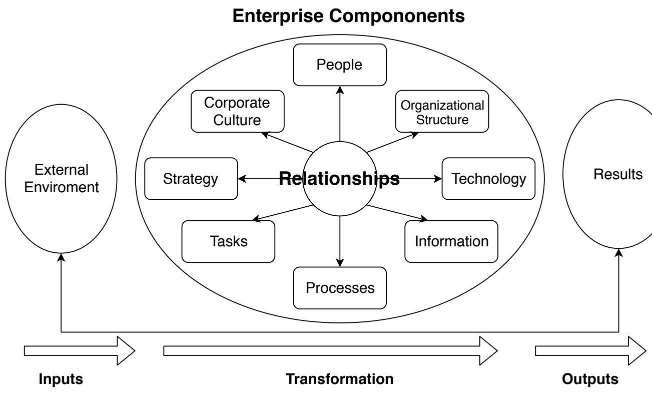


Figure 1: Generic Enterprise Architecture, adapted from [41]

Figure 1 gives a visual impression of an abstract enterprise architecture. From the definition and the figure, we can observe that EA gives an holistic view of the enterprise components and their relationships. This also includes the relationship between the IT/IS and the rest of the enterprise, which is often emphasized, when discussing EA in IS science. It is often referred to as business and IT alignment, which we will also discuss later on. Without this holistic view on the enterprise and its relationships, each department of an organization may evolve without considering the other departments and the overall business strategy [27]. So, EA can help businesses to keep their domains inline with each other and the overall enterprise strategy. But what further benefits arise from an enterprise architecture?

In research this question has been almost totally omitted for quite some time. However, a lot of research was done on EA frameworks, models and other methodologies [48]. Benefits that EA delivers to an enterprise receive only little consideration on the side in most literature, even though this plays an important role in practice, because of investments in EA must be justified, e.g. by Chief Information Officers (CIOs). This has changed, at least a little, in the last few years, but explicit research on the benefits and added value of EA is still only moderate [48].

This brings us to our research topic. We want to gather the benefits of EA from other literature through a literature review, to identify the value, EA adds to organizations. The terms value and benefit will also be delimited from each other and a more clear definition of value will be given in Section 4. We formulate our research question (RQ) as:

> *RQ: What are the values and benefits EA adds to an organization?*

To find an answer to this question, we continue our paper as follows. In the next section we describe our research method. Section 3 deals with a few related papers which have a similar research question. In Section 4 we will present the findings and results of our research. The findings are then more discussed and analyzed in Section 5. Finally, Section 6 gives a summary of the major findings and propositions for future research.

2. RESEARCH METHOD

We approached the RQ by conducting a *Systematic Literature Review* (SLR) following the guidelines proposed by Kitchenham et al. [24] combined with Webster et al. [52].

To get a rough overview of the topic EA in general and, especially, EA benefits, we searched various digital libraries mentioned by [24]. Thus, our literature consists mainly of conferences and journal articles from online libraries, despite the fact that a few books on the topic could be found, which are also considered in the review. That is, we searched via IEEE Xplore, ACM Digital Library and Google Scholar. Firstly, we only searched specifically for terms as "enterprise architecture benefits", "enterprise architecture value" and "enterprise architecture added value".

The ACM Digital Library only returned 20 results when searching for entries that include "enterprise architecture" and value or benefit. The exact research term was ""enterprise architecture" AND (value OR benefit)". A general search on EA resulted in 182 entries. In these results were still papers that do not cover the topic EA at all. So, we limited the results further by only including papers that include "enterprise architecture" in the title, which resulted in 62 papers.

On the other hand, a search with IEEE Xplore resulted in over 3.000 articles about EA in general and over 2.000 about the value and/or benefits of EA. Thus, also here, we specified that the paper's title must include "enterprise architecture" or ""enterprise architecture" AND (value OR benefit)", respectively. This resulted then in 423 articles about EA and 92 about EA value and/or benefits.

A naive search with Google Scholar for EA, with e.g. "(value OR benefits) AND "enterprise architecture"", resulted in more than 20.000 results. Consequently, we chose to only search for literature that includes those terms in the title, as we did when searching IEEE Xplore and ACM Digital Library. This specification resulted in 3.630 entries about enterprise architecture in general and 101 which also cover the value or benefits of EA.

Since the topic also includes the financial value of EA, we additionally searched in a database dedicated to economics. We did this via IDEAS/RePEc, which resulted in 631 results about enterprise architecture and 120 about the value or benefits of enterprise architecture. The 120 papers were found by searching in the abstract because only searching in the title resulted in only 3 papers and searching in the whole document again returned literature that does not deal with EA.

An overview of the results of different search queries on the search engines can be seen in table 1. Note that the results were not filtered for duplicates.

Table 1: Results from search engines

Search Engine	EA	EA Value/Benefits
ACM	62	20
IEEE Xplore	423	92
Google Scholar	3.630	101
IDEAS/RePEc	631	120
Total	4.736	333

It is obvious that we could not review all the 333 or even 4.736 articles in detail. So, the next step was to reduce the literature that we would review even more by sorting out manually. We decided to first limit the number by only choosing the 40 most cited articles from the papers that are dedicated to EA value or benefits. To reduce possible information bias we further included 20 most cited articles that do not directly deal with the topic but enterprise architecture in general. That is, e.g., EA frameworks, EA benefits and EA values, EA assessments or literature reviews about EA. We split up the literature on both authors. Each reviewer first only read the abstract of their assigned papers and marked them accordingly as relevant to search the paper in detail for EA benefits. Our exclusion criteria was simple, all literature that does not deal directly with EA will not be considered in the review. However, through the filters we applied on the search engines, there were only a few papers left that were off-topic. This resulted in 43 final papers to search in detail for EA benefits. After extracting the claims about EA benefits and value from the literature, we searched for more papers in the sources of some of the literature, e.g. [48]. In articles that deal with enterprise architecture in a general way, we additionally reviewed the source when a benefit was claimed. We did this because often authors just mention one or two benefits from another source as examples, even when the original source mentioned more benefits. However, this backward search was not carried out as systematic as the primary search with the different search engines.

3. RELATED WORK

While conducting our SLR, we only found 4 other works [50] [22] [32] [6] dealing with the question what value EA adds to an organization or business. Most academic research on EA focuses on frameworks [43] [51], methodologies [46] or other practical guidelines [13]. Even in the really comprehensive research [48] on enterprise architecture, the benefits and added value of EA found no big attention. That can also be seen by our search results in table 1. However, the papers that concentrate more directly on EA value and benefits, often try to exploit the EA value and benefit realization process. That is, how EA adds value to a business and how businesses can accomplish benefits of EA, not what the value actually is. In this section we only want to briefly mention the four papers we found, that also did a literature review on the added value and benefits of EA.

We begin with the most cited and also most extensive research, which was carried out by Tamm et al. [50] in 2011. The authors did a systematic and an exploratory literature review to first identify benefits of enterprise architecture. Tamm et al. then analyzed the benefits claimed in the reviewed literature and grouped them in categories. The main focus of the paper is, to clarify the question of how enterprise architecture leads to organizational benefits, which was done with the help of the categorization. In the process of their research, they identified four benefit enablers, through which EA leads to benefits for an organization. The benefit enablers are Organizational Alignment, Information Availability, Resource Portfolio Optimization and Resource Complementarity. These four benefit enablers themselves depend on the quality of the enterprise architecture. Their relation with EA and EA benefits is summarized in the EA Benefits Model proposed by Tamm et al. Furthermore, the authors came to the conclusion that an important factor is the size of the organization and the complexity of the IT environment of the business. So, not all organizations can expect to realize the same benefits when implementing an EA. Their findings also proof that research on EA benefits is rare, but a lot of other literature with other topics on EA claim benefits. Out of a total of 50 studies, 9 have as the main topic EA benefits, while 41 make claims of benefits.

To our concerns, the first mentionable literature review on EA benefits was conducted in 2006 [31], when there was really rare research on EA value and benefits, at least in academic research.

In this paper, benefits and value of EA were collected and categorized for the first time. The author grouped the benefits into a two-dimensional taxonomy. First, how attributable the benefit is to EA, from weakly to strongly and second, how measurable the benefit is, from non-quantifiable to quantifiable. Hard benefits are quantifiable and strongly related to EA. Indirect benefits are also quantifiable but are not directly attributable to EA. Intangible and strategic benefits are both non-quantifiable, but intangible are more directly related to EA than strategic benefits. Most benefits that were found are of the indirect type and at least are intangible benefits. The found benefits were also verified by interviews of practitioners.

We end this section with the latest literature review on benefits of Enterprise Architecture, which was done in 2017 [22]. The authors not only collected the benefits through a literature review, but also verified and expanded them by interviewing EA experts. They gathered 40 EA benefits in total through the review and the interviews and then grouped the benefits into five categories. The five categories are *Operational*, *Managerial*, *Strategic*, *IT Infrastructure* and *Organizational*. They adapted the categories from Shang and Seddon's framework about enterprise system benefits [47]. Please note that enterprise system is not the same as enterprise architecture. With 13 benefits, the strategic category is the one with the most benefits. Additionally, they tried to identify the success factors of EA to understand the process of benefit realization. The authors identified 37 EA success factors grouped in the four categories *Product Quality*, *Infrastructure Quality*, *Service Delivery Quality* and *Organizational Anchoring*.

4. FINDINGS

Before presenting our main findings, which are the EA benefits we identified, we want to discuss the value term in order to clarify what is meant by value in the context of enterprise architecture. Further, we give some main examples of the goals that can be found in various EA literature, e.g., in framework propositions. By comparing the goals with our identified benefits, we then are able to check whether the goals are actually achieved. However, such validation of the achievement of EA goals must be considered critically because of the lack by empirical data in most literature.

4.1 The Value Term

One important aspect to consider when trying to solve the research question, is the term "value". In order to answer the RQ, we first need a clearer understanding of the value concept. When speaking about the added value of EA, we can not exclusively pay attention to the direct value that arises from EA [40]. Like generally in IS, we must rather note that EA may provide indirectly value to an organization [35]. E.g., not all benefits that arise from an implemented enterprise architecture lead directly to financial outcome. We have also seen this in the last section about related work that other authors grouped the benefits according to, how quantifiable and how directly they are attributable to EA. Thus, it is not enough to just measure the direct financial returns of EA.

Rodrigues et al. discussed the value concept in the context of enterprise architecture critically and several issues of the term in [40]. One important issue the authors address is they distinction between the value of EA and EA benefits. According to Rodrigues, the difference between the two, is the following. When speaking about value, the expenditure that is necessary to implement an enterprise architecture has to be considered in addition to the benefits that arise from EA. So, in the economy value is often defined as the profit minus the input or investment. While, when speaking about EA benefits, only the benefits are considered.

One also should take under consideration that some benefits will be only perceptible after a few years, as the EA becomes more mature and well integrated to the business [9] [5]. We conclude that there are three points which make it quite hard to measure the value of EA in practice, which may be one major factor why there is not much empirical data on that topic. Those are the measurability of the benefits, the attributility towards EA and long maturing process it takes, until some benefits can be achieved. Nevertheless, a few frameworks and models to measure the outcomes of EA exist, e.g., [39] and [44], but they do not seem to be used in practice very much. Rogrigues et al. emphasize the importance of using both, finical and operational measures, when talking about the measurement of EA value. In fact, researchers more likely started to interview EA practitioners and leading positions in businesses, see e.g. [9], [22] or [50]. From these interviews we can see that most claims about the potential benefits of EA can at least be backed up by EA experts and practitioners, see e.g. [47] and [29].

4.2 Goals of EA

In this section we want to name some goals of EA and EA frameworks, so we can check, if the benefits we found include these goals. Clearly, each organization has its own goals in mind when designing an EA for their business. These goals should help to achieve the overall business goals. However, we want to list some goals that are found in general EA literature and especially in literature about frameworks. Lange and Mendling already collected some goals from various other literature in 2011 [25]. We extended their list with some goals of *The Open Group Architecture Framework* (TOGAF) [18].

- Business-IT alignment
- Decrease costs
- Reduce risks
- Improve manageability
- Agility, portability and scalability
- Standardization and interoperability
- Improve business operations
- Improve IT effectiveness

TOGAF is one of the most used EA frameworks in practice all over the world [16].

4.3 EA Value and Benefits

A list of our found benefits can be seen in table 2 with the sources they were mentioned in. Benefit lists of other SLRs, e.g. the one mentioned under Related Work, were excluded from the searched literature for EA benefits. The benefits and the categorization are discussed in the next section.

Table 2: Identified Benefits of EA

Perspective	**Benefit**	**Source**
Governance	Support of decision making	[33], [28], [32], [9], [15], [14]
	Speed up of decision making	[14]
	Support of coordination	[33], [28], [32], [15]
	Support of control	[33], [28], [14]
	Help managers and workers to analyze problems	[33], [28], [15], [14]
	Help managers and workers to visualize complex subjects	[33], [28], [14]
	Gives an integrated vision and a global perspective of informational resources	[33], [30], [4], [7]
	Discovery and elimination of redundancy in business processes	[33], [11], [9], [14]
	Helps identifying integration problems	[4]
	Support of project planning	[32], [8], [21], [15], [14]
	Helping the agency meet its transformation objectives	[9]
	Eliminating duplicate investments	[9]
	Capturing gains in business process efficiency	[9]
	Provides an holistic view of all parts of the business	[9], [14]
	Helps executives assessing their investments in operations and projects	[9]
	Can help executives carve out areas that can generate cash	[9]
	Phasing out costly and complex legacy systems	[9], [15]
	Identify the most promising projects	[14]
Operational	Brings order and structure to the business	[33], [19], [14]
	Unify and integrate business processes across the enterprise	[4], [7]
	Support of business process design	[32], [8], [21], [15]
	Support of system development	[32], [8], [21]
	Decreased operating costs	[9]
	Standardization of automated processes	[14]
	Greater overall technology effectiveness	[9]
	Flexibility, especially of IT systems	[15], [9], [3]
	Faster project initialization	[14]
Strategic	Help managers and workers to create new products in alignment with overall business strategy	[33], [28], [9]
	Contributes to having IS that reflect common goals	[33], [49]
	Business and IT alignment	[33], [56], [54], [32], [17], [9], [3], [14], [23], [34]
	Increase agility to business change	[4], [7], [9], [14]
	Helps building long-term business value	[9]
	Improved speed-to-market	[9], [15], [3]
	Reduction of risk and misaligned programs	[9], [14]
	Gaining competitive advantage on the business market	[9]
	Helps completing major IT projects earlier	[9]
	Tightening the connections between tactics and strategy	[9]
	Planning and executing customer-focused services	[9]
	Managing end-to-end customer experience	[9]
	Identifying key stakeholders	[14]
Communication, Collaboration, Compliance	Reduction of information and organizational complexity	[33], [11], [4], [7], [17], [9], [14]
	Encourage cooperation between different business departments	[33], [49], [14]
	Standardization across the IT function	[9], [15], [14]
	Improving transparency	[9], [15]
	Reuse of IT and business processes	[9], [15]
	Managing outsourcing arrangements	[14]
	Facilitate cooperation with other organizations	[14]
	More efficient communication between organization members	[14]

5. DISCUSSION/ANALYSIS

5.1 The Categorization

In order to make sense of the data we grouped them together into four categories. The categories are *Governance, Operational, Strategic* and *Communication, Collaboration and Compliance* 2. We chose this categorization because most papers used similar schemes to group their value from EA together, see e.g. [40]. So we abstracted them into the four groups we had chosen.

5.1.1 Governance

Governance is the ability to manage the involvement and support of all parties with an interest or responsibility to the endeavor with the objective of guarantee that the cooperate interests are served and the goals achieved [18]. They are clear identified stakeholders and a clear understanding of their interest in and responsibility to the project [18]. Also, a culture that positively influences participation towards corporate objectives instead of locals. A culture that has meaningful, as opposed to symbolic, participation in management processes and a commitment to project reviews, challenges and an open mind for advices from outsiders [18]. So values that are helping to reach these goals fall under this category.

5.1.2 Operational

An operational plan is a highly detailed plan that guides an organization through handling teams, sections or departments to achieve the organization's goals. So this category consists of the benefits that benefit

What	Strategies and task must be done
Who	Which people have responsibilities for strategies and task
When	In which time the strategies and task must be completed
How much	the amount of resources to complete the task or strategy

5.1.3 Strategic

A strategic plan outline which course or mission one will be taking in the far future. This category includes all benefits related to an improved overall strategy of the business, as well as the integration of different domains of the business to the enterprise strategy.

5.1.4 Communication, Collaboration, Compliance

This category includes all aspects concerning the improved communication between the various departments of the enterprise, as well as their improved collaboration.

5.2 Analysis

5.2.1 Governance Analysis

From the table 2 we concluded that a lot of papers had "Support of decision making" and "Support of project planning" as benefit from EA for the Governance category. For the Governance category most papers also named "Support of coordination", "Help managers and workers to visualize complex subjects", "Gives integrated vision and a global perspective of informational resources" and "Discovery and elimination of redundancy in business processes". What we noticed is that most papers matched the same benefits in Governance in comparison to our other categories. Especially [33], [32] , [28], [15], [14] named a lot of benefits in common.

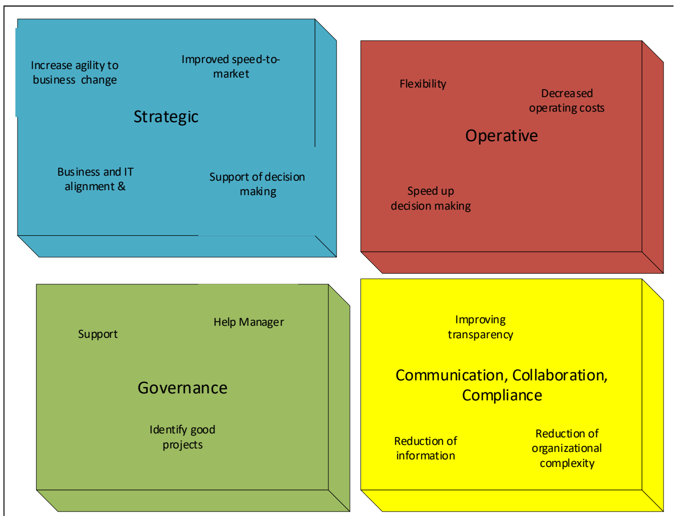

Increase agility to business change
Improved speed-to-market
Strategic
Business and IT alignment &
Support of decision making
Flexibility
Decreased operating costs
Operative
Speed up decision making
Support
Help Manager
Governance
Identify good projects
Improving transparency
Communication, Collaboration, Compliance
Reduction of information
Reduction of organizational complexity

Figure 2: Categorization of the Benefits

5.2.2 Operational Analysis

The only papers that named operational benefits are [3], [4], [7], [8], [9], [14], [15], [19], [21], [32], [33]. Those are about 50% of all our findings. A reason for that could be that the TOGAF [18] does not clearly state operational planning. But the most titled benefits (according to table 2) are "Brings order and structure to the business", "Support of business process design", "Support of system development" and "Greater overall technology effectiveness".

5.2.3 Strategic Analysis

About half of the papers name Strategic benefits. Most of them call "Business and IT alignment" as a benefit. In fact, it is the most named benefit across all categories. It only makes sense that a lot of papers call it a benefit, because one of EA goals is to align Business with IT according to TOGAF. We can also notice that "Increase agility to business change" was called about 30% from the papers that named at least one strategic benefit (see Table 2). Besides "Help managers and workers to create new products in alignment with overall business strategy" and "Improved speed-to-market" got a fair share of mentions.

5.2.4 Communication, Collaboration, Compliance

Surprisingly for this category most papers do not name a lot of benefits. But if they named one, it was most of the time "Reduction of information and organizational complexity","Encourage cooperation between different business departments" and "Standardization across the IT function".

5.3 Value Graph

Some benefits from EA are influenced by other categories or benefits. Categories alone do not capture this aspect. So we decided to model the relationship between the benefits as the following graph. We say for the values *A* and *B* that *A positively influence B* if *A* has a positive effect on *B* and visualize it with an arrow from *A* to *B*. If we look at the graph 3 below we see a subset of the value graph. For the sake of visibility we only choose this subset. Also, we took especially this one because it models the relationships between the values quite simple.

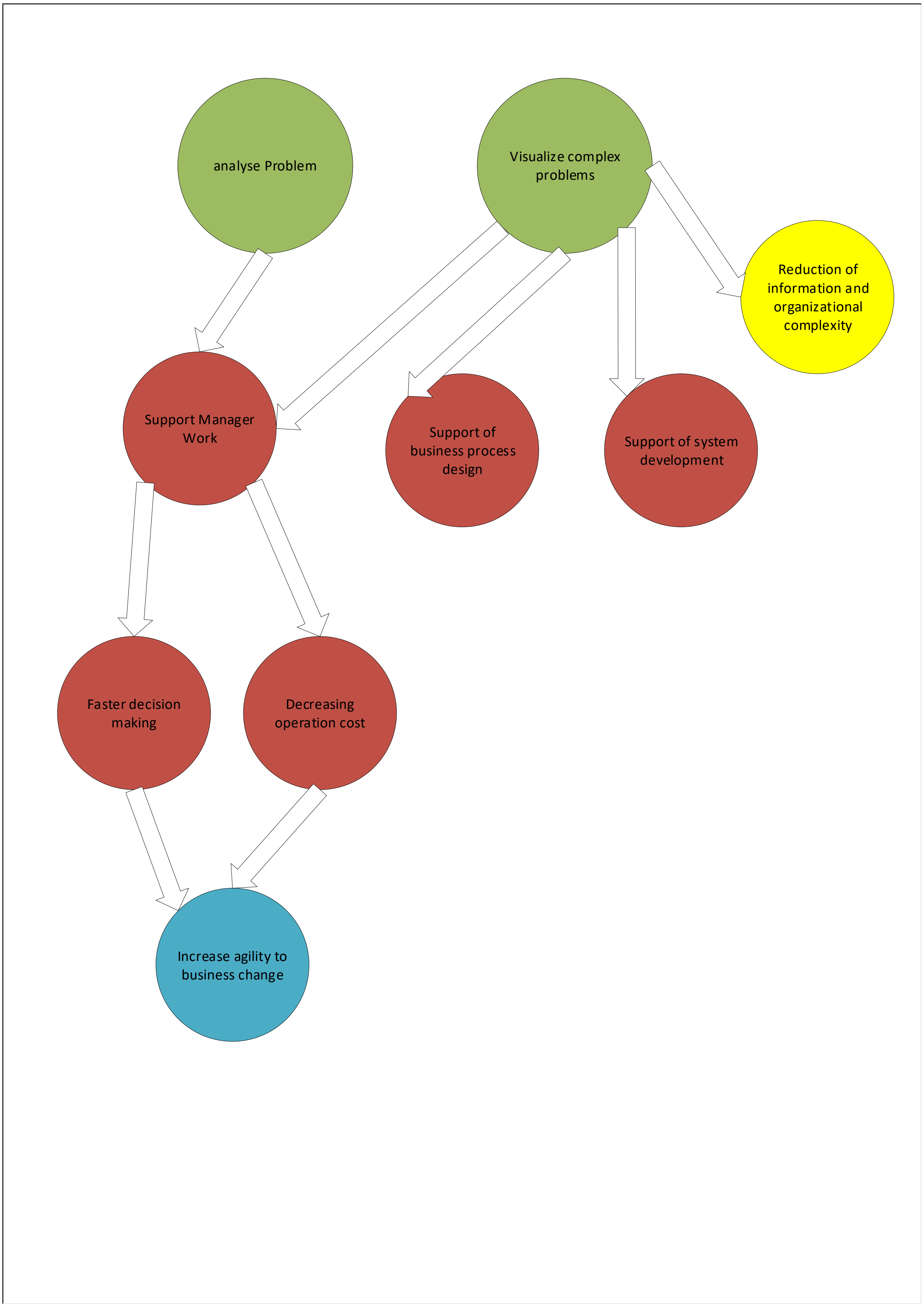

analyse Problem
Visualize complex problems
Reduction of information and organizational complexity
Support Manager Work
Support of business process design
Support of system development
Faster decision making
Decreasing operation cost
Increase agility to business change

Figure 3: Relationships between the different benefits. The colors are corresponding to the figure before. An arrow to the next node mean a positive influence

If we look closer to the figure 3 we can see the value ”analyze Problem” which positively influences "Support Manager Work". This comes due to the fact that, understanding the problem helps to do a better managing job. Also, the value ”Visualize complex problems” influences ”Support Manager Work” for the almost same reasons. Even further ”Support Manager Work” has a positive influence on ”Faster decision making” and ”Decreasing operation cost”. When the manager is supported in his work he can make faster decisions and operate more efficient which leads to decreasing operation costs. Both, ”Faster decision making” and ”Decreasing operation cost”, lead to a positive influence in ”Increase agility to business change” this could be because the business can react faster to change with fast decision making and decreasing operation costs. When we look at the graph from ”Visualize complex problems”, we have positive influence on ”Reduction of information and organizational complexity”. It is only natural that we need less information and the organizational complexity decrease, when we understand the problem better. Also, ”Support of business process design” and ”Support of system development” are positively effected by the value ”Visualize complex problems”. This comes for the same reason as ”analyze Problem” that understanding the problem better helps to develop a business process.

As we can see, values may positive influence each other which can lead to better results.

5.4 How does it fit with the goals of EA?

Earlier in this paper we mentioned the goals of EA according to TOGAF [18]. If we look back to the analysis of our categorization we can notice that some benefits, that align with the goals of EA, get a lot of mentions. And some goals of EA are not mentioned or represented as benefit at all. This is due to the fact that each organization has its own set of objectives they want to fulfill or focus on. So a generic list with goals might only give a hint in which direction they shall move with EA. But every organization needs their own set of goals they want to achieve with EA. Regardless of that we can still notice that Business-IT alignment was one of the most mentioned benefits of our LR and this fits into the goal of EA according to TOGAF "Business-IT alignment". Another benefit, that was mentioned a lot, is ” Increase manageability” which overlaps with the goal ”Agility, portability and scalability”. Also, the goal ”Improve manageability” is reflected through many benefits as we can see in table 2. An honorable mention would be from the Governance category ”Support of decision making” with a couple of mentions. Not surprisingly the goal”Standardization and interoperability” is represented by the category ”Communication, Collaboration, Compliance”, whose benefit ”Reduction of information and organizational complexity” gets mentioned in a lot of papers. The goal ”Improve business operations” is also covered with our category ”Operative” which has some mentions according to table 2. So ”Reduce risk”, ”Decrease costs” and ”Improve IT effectiveness” are rather rarely or not mentioned at all in our LR about benefits from EA. But how we can see from figure 3 a benefit that gets mentioned a lot can still positively influence a benefit that that is rarely mentioned. So in the end even, when the goals are not represented through our researched benefits, they might be still present due to the positive influence of other benefits.

5.5 Summary

At the beginning we introduced our categories and our motivation that lead to these categories. Then we give a feel in what we understand under these categories. After that we summarized our findings in each category and highlighted noteworthy mentions. When we introduced a relationship between benefits and how they are influenced by other benefits to give a connection between benefits. In the end we concluded in which way our findings fit in the goals of EA according to TOGAF.

6. CONCLUSION

In this paper, we presented our findings of a systematic literature review on the benefits of EA that was conducted in order to answer the question of what the added value of EA is. First, we searched ACM, IEEE Xplore, Google Scholar and IDEAS/RePEc for literature about EA. Next, the literature was reviewed and benefits of EA extracted from it. The benefits found in the literature were grouped in four categories: *Governance*, *Operational*, *Strategic*, and *Communication, Collaboration, Compliance*. The most benefits that were found belong to the *Governance* or *Strategic* categories, thus not bringing direct value to the business in form of money, but providing long-term benefits. Even though, one identified benefit is the reduction of operating costs. The three benefits that were claimed by the most literature are “Business and IT alignment” (strategic), “Reduction of information and organizational complexity” (communication etc.) and “Support of decision making” (governance).

We also analyzed the relations between several benefits. So even though some papers do not recall certain benefits, they are still related, present or influenced by significant higher named benefits. However, these relations must be validated by empirical data in further research.

Additionally, we took a short look on some major goals of EA frameworks and EA in general. They all could be found in the benefits list we gathered from the literature. Nevertheless, the significance of this form of validation should not be overestimated, since most papers do not give empirical proof of the claimed benefits. But, what can be said with more safety is that EA practitioners approve the value that an enterprise architecture brings to their business. This has been validated by interviews and surveys [29].

Furthermore, we noticed a lack of validation of the benefits through empirical data. This can be attributed to the difficulty of measuring the outcome value of EA. Hence, there is a need of research in a consistent theory to measure the value and benefits that arise from an enterprise architecture. One step forward could be taken by standardizing the benefits of EA, e.g., by unifying benefits that have the same meaning and delimiting the benefits better from each other.

7. REFERENCES

[1] Systems and software engineering - architecture description. 2000.

[2] Information system complexity and business value. *International Journal of Economics & Management Sciences*, 6(2):1–4, 2017.

[3] M. Alaeddini, H. Asgari, A. Gharibi, and M. R. Rad. Leveraging business-it alignment through enterprise architectureâĂŤan empirical study to estimate the extents. *Information Technology and Management*, 18(1):55–82, 2017.

[4] V. Anaya and A. Ortiz. How enterprise architectures can support integration. In *Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pages 25–30. ACM, 2005.

[5] A. Bachoo. Enterprise architecture practices to achieve business value. In *2018 IEEE 20th Conference on Business Informatics (CBI)*, volume 01, pages 1–9, July 2018.

[6] V. Boucharas, M. van Steenbergen, S. Jansen, and S. Brinkkemper. The contribution of enterprise architecture to the achievement of organizational goals: a review of the evidence. In *International Workshop on Trends in Enterprise Architecture Research*, pages 1–15. Springer, 2010.

[7] T. Brown. The value of enterprise architecture. *Zachman Institute for Framework Advancement (ZIFA), www. zifa. com*, 2004.

[8] T. Bucher, R. Fischer, S. Kurpjuweit, and R. Winter. Enterprise architecture analysis and application-an exploratory study. *Journal of Enterprise Architecture*, 3(3):33–43, 2007.

[9] P. Burns, M. Neutens, D. Newman, and T. Power. Building value through enterprise architecture a global study. *Source168) Cited by*, 4, 2009.

[10] M. Castells. *Power of identity: The information age: Economy, society, and culture*. Blackwell Publishers, Inc., 1997.

[11] M. A. Cook and M. A. Cook. *Building enterprise information architectures: reengineering information systems*, volume 7. Prentice Hall Upper Saddle River, NJ, 1996.

[12] C. Council. Federal enterprise architecture framework version 1.1. *Retrieved from*, 80:3–1, 1999.

[13] F. A. Cummins. *Enterprise integration: an architecture for enterprise application and systems integration*. John Wiley and Sons, Inc., 2002.

[14] R. Foorthuis, M. Van Steenbergen, S. Brinkkemper, and W. A. Bruls. A theory building study of enterprise architecture practices and benefits. *Information Systems Frontiers*, 18(3):541–564, 2016.

[15] A. Goikoetxea. *Enterprise architectures and digital administration: Planning, design and assessment*. World Scientific, 2007.

[16] T. O. Group. Members of the open group. `https://www.opengroup.org/our-members`. [Online; accessed Dec. 20, 2018].

[17] S. Hacks, M. Brosius, and S. Aier. A case study of stakeholder concerns on eam. In *Enterprise Distributed Object Computing Workshop (EDOCW), 2017 IEEE 21st International*, pages 50–56. IEEE, 2017.

[18] V. Haren. Togaf version 9.1. 2011.

[19] W. H. Inmon, J. A. Zachman, and J. G. Geiger. *Data stores, data warehousing and the Zachman framework: managing enterprise knowledge*. McGraw-Hill, Inc., 1997.

[20] M. Janssen and K. Hjort-Madsen. Analyzing enterprise architecture in national governments: The cases of denmark and the netherlands. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 218a–218a. IEEE, 2007.

[21] H. Jonkers, M. M. Lankhorst, H. W. ter Doest, F. Arbab, H. Bosma, and R. J. Wieringa. Enterprise architecture: Management tool and blueprint for the organisation. *Information systems frontiers*, 8(2):63–66, 2006.

[22] M. B. Jusuf and S. Kurnia. Understanding the benefits and success factors of enterprise architecture. In *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017.

[23] L. Kappelman, T. McGinnis, A. Pettite, and A. Sidorova. Enterprise architecture: Charting the territory for academic research. *AMCIS 2008 Proceedings*, page 162, 2008.

[24] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering, 2007.

[25] M. Lange and J. Mendling. An experts' perspective on enterprise architecture goals, framework adoption and benefit assessment. In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International*, pages 304–313. IEEE, 2011.

[26] M. Lankhorst. *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer Publishing Company, Incorporated, 2nd edition, 2009.

[27] M. Lankhorst. *Enterprise architecture at work: Modelling, communication and analysis*. Springer, 2009.

[28] K. C. Laudon and J. P. Laudon. *Management information system: organization and technology in the networked enterprise*. Pearson Custom Publishing, 2000.

[29] Å. Lindström, P. Johnson, E. Johansson, M. Ekstedt, and M. Simonsson. A survey on cio concerns-do enterprise architecture frameworks support them? *Information Systems Frontiers*, 8(2):81–90, 2006.

[30] F. Niederman, J. C. Brancheau, and J. C. Wetherbe. Information systems management issues for the 1990s. *MIS quarterly*, pages 475–500, 1991.

[31] E. Niemi. Enterprise architecture benefits: Perceptions from literature and practice. *Tietotekniikan tutkimusinstituutin julkaisuja, 1236-1615; 18*, 2008.

[32] E. Niemi and S. Pekkola. Adapting the delone and mclean model for the enterprise architecture benefit realization process. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–10. IEEE, 2009.

[33] C. M. Pereira and P. Sousa. A method to define an enterprise architecture using the zachman framework. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 1366–1371, New York, NY, USA, 2004. ACM.

[34] C. M. Pereira and P. Sousa. Enterprise architecture: business and it alignment. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1344–1345. ACM, 2005.

[35] S. Petter, W. DeLone, and E. McLean. Measuring information systems success: models, dimensions, measures, and interrelationships. *European journal of information systems*, 17(3):236–263, 2008.

[36] M. E. Porter, V. E. Millar, et al. How information gives you competitive advantage, 1985.

[37] T. C. Powell and A. Dent-Micallef. Information technology as competitive advantage: The role of human, business, and technology resources. *Strategic management journal*, 18(5):375–405, 1997.

[38] G. L. Richardson, B. M. Jackson, and G. W. Dickson. A principles based enterprise architecture: Lessons from texaco and star enterprise. *MIS quarterly*, pages 385–403, 1990.

[39] D. F. Rico. A framework for measuring roi of enterprise architecture. *Journal of Organizational and End User Computing*, 18(2):i, 2006.

[40] L. S. Rodrigues and L. Amaral. Issues in enterprise architecture value. *Journal of Enterprise Architecture*, 6(4):27–32, 2010.

[41] M. A. Rood. Enterprise architecture: definition, content,

and utility. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1994. Proceedings., Third Workshop on*, pages 106–111. IEEE, 1994.

[42] J. W. Ross, P. Weill, and D. Robertson. *Enterprise architecture as strategy: Creating a foundation for business execution*. Harvard Business Press, 2006.

[43] J. Schekkerman. *How to survive in the jungle of enterprise architecture frameworks: Creating or choosing an enterprise architecture framework*. Trafford Publishing, 2004.

[44] J. Schelp and M. Stutz. A balanced scorecard approach to measure the value of enterprise architecture. 2007.

[45] M. Schöenherr. Towards a common terminology in the discipline of enterprise architecture. In *International Conference on Service-Oriented Computing*, pages 400–413. Springer, 2008.

[46] R. Sessions. A comparison of the top four enterprise-architecture methodologies. *Houston: ObjectWatch Inc*, 2007.

[47] S. Shang and P. B. Seddon. Assessing and managing the benefits of enterprise systems: the business manager's perspective. *Information systems journal*, 12(4):271–299, 2002.

[48] D. Simon, K. Fischbach, and D. Schoder. An exploration of enterprise architecture research. *CAIS*, 32:1, 2013.

[49] R. Stata and P. Almond. Organizational learning: The key to management innovation. *The training and development sourcebook*, 2:31–42, 1989.

[50] T. Tamm, P. B. Seddon, G. G. Shanks, and P. Reynolds. How does enterprise architecture add value to organisations? *CAIS*, 28:10, 2011.

[51] L. Urbaczewski and S. Mrdalj. A comparison of enterprise architecture frameworks. *Issues in Information Systems*, 7(2):18–23, 2006.

[52] J. Webster and R. T. Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly*, pages xiii–xxiii, 2002.

[53] R. Winter and R. Fischer. Essential layers, artifacts, and dependencies of enterprise architecture. In *Enterprise Distributed Object Computing Conference Workshops, 2006. EDOCW'06. 10th IEEE International*, pages 30–30. IEEE, 2006.

[54] R. Winter and J. Schelp. Enterprise architecture governance: the need for a business-to-it approach. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 548–552. ACM, 2008.

[55] W. Xia and G. Lee. Complexity of information systems development projects: conceptualization and measurement development. *Journal of management information systems*, 22(1):45–83, 2005.

[56] C. Young. The unexpected case for enterprise it architectures. *Gartner Group Strategy, Trends & Tactics*, 9, 2001.

[57] J. A. Zachman. A framework for information systems architecture. *IBM systems journal*, 26(3):276–292, 1987.

State of the Art in Combinatorial Security Testing

Harish Balaji Shanmuga Sundaram
RWTH Aachen University
HirschbergerStr 58-64
53119 Bonn, Germany
harish.balaji.shanmuga.sundaram@rwth-aachen.de

ABSTRACT

Combinatorial testing is an effective black-box testing approach which has also been applied to security testing. Combinatorial methods can make software security testing much more efficient and effective than conventional approaches. Combinatorial methods are ideally suited for the *Internet of Things* environment, where testing can involve a very large number of nodes and combinations.

However, so far there is no systematic study that discusses the approaches in security testing and their differences. There is no single ultimate approach that fits to all company requirements and attacker models. But, combinations of different approaches can help the testers to identify and address the security vulnerabilities or weaknesses of the software applications.

Thus, the objective of this paper is to identify different approaches of combinatorial testing for security and to understand their differences.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Testing—*Blackbox Testing, Test Suites, Approaches*

Keywords

Combinatorial Testing, Security Testing, Protocol Testing, Vulnerabilities

1. INTRODUCTION

In most of the big applications that deal with a big set of data manipulation, some undetected errors arise even after many months of implementation. This means there exist some untested test cases.

Combinatorial testing is a black box testing method that can help detect such problems early in the testing life cycle. The idea of the t-way combinatorial testing is that not every single parameter contributes to every fault and most faults are caused by interactions between a relatively small number of parameters [23]. It is found that the maximum of 6-way combinations are potential enough of creating a fault in any system [23]. As the number of combinations of test parameters increases, more error resistant the application becomes.

On the other hand, *Security Testing* is a process with an intention to reveal flaws in the security mechanisms and finding the vulnerabilities of an application.The primary goal of security testing is to find how vulnerable a system may be and to determine whether its data and resources are protected from the potential intruders. Security testing is more effective in identifying the vulnerabilities when performed regularly. Performing regular security testing can avoid several inconveniences like loss of customer trust, system downtime, cost associated with securing online resources against future attacks, cost associated with legal issues for having negligent security measures.

SBA Research and National Institute of Standards and Technology(NIST) have developed a research program to bridge the gap between combinatorial testing and security testing. This resulted in the establishment of a new research field called Combinatorial Security Testing.

Identifying vulnerabilities and ensuring security functionality by security testing is a widely applied measure to evaluate and improve the security of software, which is also an inevitable part of quality assurance. Many software security exploitation result from ordinary coding flaws, rather than design or configuration errors. One study found that 64 percent of vulnerabilities are the result of such common bugs as missing or incorrect parameter checking, which leaves applications open to common vulnerabilities including buffer overflows or SQL injection. Although this statistic might be discouraging, it also means that better functionality testing can also significantly improve security.

In the last 50 years, combinatorial methods have had profound applications in coding theory, cryptology, networking and computer science with software testing being one of the most recent ones [22]. Covering arrays are discrete mathematical structures which, with the aid of proper software engineering techniques, have been utilised in very effective test sets in order to provide strong assurance. Yet, the application of combinatorial methods to applied computer science continues to arise and it comes as no surprise that the field of software security, in particular, provides a rich source of problems that seek solutions from mathematical methods. There has been ample evidence over the last few years to support this observation.

2. BACKGROUND AND RELATED WORK

Combinatorial security testing has rapidly gained favour among software testers in the past decade as improved algorithms have become available, and practical success has been demonstrated.

An example of a single-value fault might be a buffer overflow that occurs when the length of an input string exceeds a particular limit. Only a single condition must be true to trigger the fault: input length > buffer size. A two-way fault is more complex, because two particular input values are needed to trigger the fault. One example is a search/replace function that only fails if both the search string and the replacement string are single characters. If one of the strings is longer than one character, the code doesn't fail; thus we refer to this as a two-way fault. More generally, a t-way fault involves t such conditions.

The effectiveness of any software testing technique depends on whether test settings corresponding to the actual faults are included in the test sets. When such settings aren't included, the faults won't be detected. Conversely, we can be confident that the software works correctly for t-way combinations contained in passing tests. For security evaluations, it isn't enough that failures are unlikely to occur in ordinary usage, because attackers seek out even complex flaws. Testing only to verify requirements coverage is insufficient for security, or even for assuring critical functionality.

One of the most used solutions to access and protect information is by using a password. But, an easy to remember password is very easy to crack. One best way to test whether the passwords are strong is by using a password meter. Password meters are used widely to help users create better passwords, yet they often provide ratings of password strength that are, at best, only weakly correlated to actual password strength. Furthermore, current meters provide minimal feedback to users. They may tell a user that his or her password is "weak" or "fair", but they do not explain what the user is doing wrong in making a password, nor do they guide the user towards a better password. Data-driven password meter [25] is a recent development in the security domain that promises hard to crack passwords.

Bachelor students Elias Alesand and Hanna Sterneling have proposed a new idea to avoid shoulder surfing, one of the most common security related vulnerabilities. Shoulder surfing is the practice of spying on the user of any electronic device in order to obtain their personal information. They have proposed a shoulder-surfing resistant graphical password system [4]. The intention of their system is to make passwords easier to remember as well as making them harder to crack through shoulder-surfing and guessing attacks while still keeping the system quick and user friendly.

In 2016, a research on penetration testing proposed a new Security Goal Model (SGM), a model-driven penetration test case generation method[8], which can describe the regularity of current SQL injection attacks. The experiment showed that the test cases generated by the proposed method can more effectively find the SQL injection vulnerability hidden behind the inadequate defense mechanism, and can reduce the omissive report of SQL injection.

The importance of web penetration testing can be felt by looking at the case study where SBA Research developed an input parameter model to test the API of Koha integrated library management system (https:// koha-community.org) The source code of Koha is under open source license. The NIST ACTS tool was used to produce test suites that covered all possible 2-way, 3-way, 4-way and 5-way parameter combinations. More than 50 cross site scripting (often denoted as XSS) vulnerabilities in Koha were reported to the developers. Two such security vulnerabilities CVE-2015-4630 and CVE-2015-4631 were assigned to MITRE's vulnerabilities and exposures list [1, 2].

Dimitris E. Simos, a key researcher at SBA Research who is currently leading the combinatorial security testing research team of SBA Research tested the World Wide Web Consortium (W3C) tidy service against a test suite using a prototype cross-site scripting (XSS) injection tool and succeeded in discovering a previously unknown remote XSS vulnerability of this popular service [23].

3. METHODOLOGY

Dimitris E. Simos is one of the active researchers who works on the combinatorial security testing. I have used his publishes to develop this paper. Based on his research papers and the references he used, I managed to come up with my discussion of the possible combinatorial testing methods.

Dr. Johannes Dahse exploits the security vulnerabilities in PHP code for 10 years. He is an active speaker at academic and industry conferences and a pioneer in the field of code analysis. I have used his works for my references to develop this paper [9].

With blackbox testing, quality assurance professionals put themselves in the shoes of the hacker and attempt to break the app through various attack vectors. This methodology can yield a lot of information and help better secure the program from actual threats. A white paper by Security Innovation noted that software testers first analyze the system's architecture and business model to identify any security vulnerabilities. Looking over the software logic in this way can uncover subtle security and privacy issues that may not have been noticed otherwise, such as defects in design, input, system dependency, authentication, cryptography and information disclosure. Test cases are built around specifications and requirements. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily functional in nature, non-functional tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output, often with the help of a previous result that is known to be good, without any knowledge of the test object's internal structure.

Technically there are two approaches in combinatorial security testing.

3.1 Manual Testing Method

Mostly, the web applications are tested from the outside without the source code. The goal is to simulate an attack and to get an overview of how successful an attacker could be. A team of penetration testers is hired that attacks a production or test setup of the application in a realistic scenario: only with access to the URL/IP and without further knowledge about the internals.

A crucial factor is how much time is given to the testers. The final report can only list what was found in the limited time frame. This time frame should reflect the resources of a real attacker which varies from a few days for an entry level hacker to several weeks for a motivated expert level hacker.

It is recommended to hire a small company with a strong specialized team, preferably with a recommendation or a list of renowned experts. There is a huge difference in what a team of skilled security experts can find manually in the web application than a team that only uses automated blackbox tools.

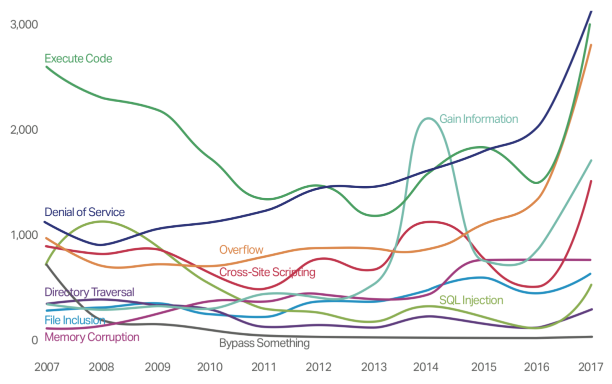
3.2 Tools Based Testing Method

Dynamic Application Security Testing (DAST) or blackbox tools perform a lightweight scan from the client-side of a given web application that is deployed and running. Multiple malicious input patterns for common web attacks are automatically send to the URL of the application while its responses are evaluated for abnormal behavior that could indicate a vulnerability.

It is recommended to use an additional test setup to prevent interference with real user data. This fuzzing approach is very slow and only scratches on the surface of an application without crawling all features deep enough. For example, vulnerabilities are missed that require a specific combination of actions (e.g. login first, activate mode 1, use feature 5). As a result, blackbox tools have a limited code coverage, a lack of support for many vulnerability types, and miss many security issues. The DAST tools are often used for assistance in manual penetration tests.

4. RESULTS

Cyber criminals use a wide range of methods to lure the legal users and gain access to the confidential resources like data and infrastructures [17]. The figure shows the trends in vulnerability disclosures in 2017 .



Source: CVE Details, https://www.cvedetails.com/vulnerabilities-by-types.php
Data for selected years and top vulnerability categories.

4.1 Identified Security Threats

There are different types of threats that can take advantage of the security related vulnerabilities. Some of those threats are discussed briefly here.

4.1.1 Privilege Elevation

Privilege Elevation is a security threat where the attacker has an account in the system and tries to increase their privileges to a higher level than what is allowed for that specific account [19].

4.1.2 SQL Injection

SQL Injection is a very common *application layer level* attack where the attacker inserts a harmful SQL statement in the input fields for execution. After a successful SQL Injection, the hacker can get all the critical information from the database server [18].

4.1.3 Unauthorized Data Access

Unauthorized Data Access may include illegal access to the reusable client authentication information in addition to the normal data. This can be achieved through data fetching operations and by monitoring the access of other people on a server or on a network [7].

4.1.4 URL Manipulation

URL Manipulation is an attack where the attacker gets the user information by manipulating the web application's URL query strings. The HTTP GET method passes the user specific data between the client and server systems. This information is passed as parameters in the query string which can be easily manipulated by the attacker [20].

4.1.5 Denial of Service

Denial of Service is an attack where the attacker makes the entire system or some parts unusable for its legal users. This may cause lack of customer trust [24].

4.1.6 Data Manipulation

Data Manipulation is a kind of attack where the attacker gets control over the HTML pages and changes the valid data used by the website [6].

4.1.7 Identity Spoofing

Identity Spoofing can be achieved by the attacker by using the credentials of a valid user to launch the attacks. This includes data theft and security compromises [5].

4.1.8 Cross-Site Scripting (XSS)

XSS enables the attacker to insert client side script on the frequently visited web pages and make the user click the malicious URL. Once executed on the user's browser, the entire system can be compromised [14].

4.2 Identified Security Testing Approaches

To avoid the possible security related vulnerabilities, it is required to have a deep knowledge in HTTP protocol and the understanding of client-server communication using this protocol. Also, some basic ideas about verification and validation becomes unavoidable. Several approaches for combinatorial security testing are discussed in this section.

4.2.1 Buffer Overflow Testing

A generic buffer overflow occurs when a buffer that has been allocated a specific storage space has more data copied to it than it can handle. The possible preventive mechanisms are code reviewing, developer training, compiler tools, developing safe functions and periodical scanning [13].

To prevent buffer overflow, developers should avoid standard library functions that are not bounds-checked. In addition, secure development practices should include regular testing to detect and fix buffer overflows. The most reliable way to avoid or prevent buffer overflows is to use automatic protection at the language level. Another fix is bounds-checking enforced at run-time, which prevents buffer overrun by automatically checking that data written to a buffer is within acceptable boundaries.

4.2.2 Cross Site Scripting

Developers have to ensure that they escape all untrusted data based on the HTML context such as body, attribute, JavaScript, CSS, or URL that the data is placed into.

For those applications that need special characters as input, there should be robust validation mechanisms in place before accepting them as valid inputs.

Cross Site Scripting Testing can be done for special characters like Apostrophe, Greater-Than Sign, Less-Than Sign. Test suites should be produced that take extra care while handing these characters [23].

4.2.3 Ethical Hacking

An ethical hacker attempts to bypass the system security and search for any vulnerability that could be exploited by malicious hackers. Ethical Hacking also known as Internet Security is very different from traditional Security. Internet security is more on a proactive basis as compared to traditional security. While traditional security is based on catching the criminals, internet security has Ethical Hackers that try to hack into a company or organization before an 'attack' so they are able to find any weak links. Ethical Hackers are hired by companies to hack their own respective company and be able to identify any loopholes where an ill-intentional hacker could create damage so that the company can buff its security and cover the cracks. They use their creativity and skills to make the internet world of a company a foolproof and safe place for both the owners and the clients [16].

4.2.4 Denial of Service Testing

The possible preventive mechanisms for Denial of Services (DoS) are performing thorough input validations, avoiding highly CPU consuming operations and separating data disks from system disks.

An effective defense against an HTTP flood can be the deployment of a reverse proxies spread across multiple hosting locations. By deploying many reverse proxies in different locations, the crush of incoming traffic is split into fractions, lessening the possibility of the network becoming overwhelmed. Deploying this type of architecture can be done in the scramble after an attack has begun, or baked into the network architecture of a web site as a preventative defense [24].

4.2.5 Web Penetration Testing

A penetration test is an attack on a computer system with the intention of finding security loopholes, potentially gaining access to it, its functionality and data.

For performing the automatic web penetration test, web scanners are used. They first crawl the target, then attack to the results of the previous phase and finally report vulnerabilities in the target [15].

4.2.6 Operating System- Call Testing

The NIST ACTS tool was used to produce test suites that covered numerous t-way combinations of system-call arguments in the kernel of the LINUX operating system. The testing experiments revealed various security related vulnerabilities that were reported for further analysis [21].

4.2.7 Password Strength Testing

Strong test suites should be developed to test the strength of passwords. A strong password must contain at least one upper case letter, one lower case letter, one number, one special character and minimum 8 characters. Another way of cracking the password is if user-name/password is to target cookies if cookies are stored without encryption. Frequent changing of passwords is also an effective way of reducing security threats [12] [25]. Checking for the accuracy of the passwords will also reduce the threats [10].

4.2.8 SQL Injection Testing

SQL Injection Testing can be done for the special characters like Apostrophes, Brackets, Commas and Quotation marks. Test suites should be designed carefully for checking these characters. Also, finding out the code from the code base where direct SQL queries are executed on the database by accepting some user inputs. Continuously monitoring SQL statements from database-connected applications and limiting the use of dynamic SQL will also reduce the SQL injection attacks. The SQL injection attack prevention methods include improving the safety of the server configuration and strengthening the code to user input information filtering check [18].

4.2.9 Posture Assessment

This describes the overall security posture of an organization; it is a combination of ethical hacking, security scanning and risk assessment.

A series or combination of these techniques can be employed at code level and configurations level to identify and avoid the security vulnerabilities of the system.

5. DISCUSSION

We suggest the companies to have periodic sessions and workshops on the latest security threats and the best practices. The passwords should be checked for its endurance whenever the user tries to change them and this can be achieved by installing a password meter [25].

Cookies are often used in web applications to identify users and their authenticated session. Stealing a cookie from a web application will lead to hijacking the authenticated user's session. The cookie value string ensures that the strings do not contain any commas, semicolons, or white-space (which are disallowed in cookie values). Some user agent implementations support cookie prefix signals to the browser, and cookie request should be transmitted over a secure channel. Cookies must be restricted and traced to a secure origin. This prevents the cookie from being sent to other domains.

The web applications that work more on documents should be tested in the grounds of file system. Releasing information about sensitive documents that may be contained on a web-page is a good spot for attackers to manipulate unguarded web applications. Also, the URLs of the documents should be encoded.

CERT-UK recommends the use of an appropriate firewall and anti-virus software (which should be updated regularly). New vulnerabilities are constantly being discovered and so a system's defences are only as good as the day they were updated. A culture of safety and good housekeeping should be encouraged as per the UK government's 10 Steps to Cyber Security [3]. Security vulnerabilties can be caused as a result of expired softwares. So, it is important to include the version and validity information of crucial software like anti-virus in the test suites.

Several other security related precautions should also be considered while designing the test suites. A regular audit should be practiced to track the list of legitimate users of the system. Test Cases should be developed to check against each user's validity. Because, an old user who is no longer working for a company but with unrevoked permissions is a potential attacker. An efficient test suite developed for this functionality can help companies to minimize the attacks and compromises at a very early stage. This should be checked at all the grounds including testing, education and production environments.

To check whether the user has typed in the correct password, systems must have a reference to check against. An attacker who can obtain a copy of this reference file can run cracking programs against it and will almost inevitably succeed in discovering the passwords for several user accounts. Password files should therefore be among the best protected information the organization holds, held on well-secured machines with limited access and, unless this is impossible, holding only salted hashes rather than the actual passwords. The choice of hashing algorithm can significantly affect the time to crack a password file. If the passwords are maintained in a flat file format, it becomes very easy for the attackers to gain access to the company's confidential data.

The National Institute of Standards and Technology has declared that the Out of Band authentication methods should be retired in the recently completed SP 800-63 digital identity guidelines [11]. Because, the knowledge of full name, phone number and email address can unlock a potential user account. Information like this is readily available in Facebook, LinkedIn and other online services where the user has signed up.

Also, the system should be developed in a way to help users cope with 'password overload'. Only use passwords where they are really needed. Use technical solutions to reduce the burden on users. Allow users to securely record and store their passwords. UK National Cyber Security Centre advice the developers to only ask the users to change their passwords on indication of suspicion of compromise. Because, frequent requirement to change passwords runs a significant risk of encouraging users to adopt sequences of passwords (e.g. by changing a digit) that increase the likelihood of a successful password guessing attack.

Encouraging users to never reuse passwords between work and home. And when the password needs to be changed, the new password should not be the one which has been already used by the user. Train staff to help them avoid creating passwords that are easy to guess. And, beware of the limitations of password strength meters. Put technical defences in place so that simpler but efficient passwords can be used.

Steer users away from predictable passwords and ban the most common. NIST Special Publication 800-63B emphasis the rule that the testers shall compare the prospective secrets against a list that contains values known to be expected or compromised.

- Passwords obtained from previous breach corpuses
- Dictionary words
- Repetitive-sequential characters like 'aaa123'
- Context-specific words, such as the name of the service, the user-name, and derivatives thereof

Technically, it is a combinatorial security test practice suggested by NIST Special Publication 800-63B. What's left is just to flag all bad passwords and maybe to send these users an email that strongly suggests they change their password.

However, there is no guarantee that the chosen password is good just because its not obviously bad.

The system should be tested for its ability to lock the account when repeated failed login attempts occur. An alternative method is increasing the delay between login attempts when there have been repeated failures. A combination of these methods makes an efficient defence against brute-force attacks. This can be effective in slowing down the attacks and letting the legitimate users buy response time to react to the alarm. I propose a combinatorial testing suite that checks both the ability of a password meter to measure the strength of the supplied password and the ability to serve an alternative strong password as a suggestion.

Checking whether all the computer monitors in the workstation are having screen privacy filters installed will reduce the shoulder-surfing to a greater extent.

6. CONCLUSION

Different approaches for combinatorial security testing were addressed in this paper. Each approach has its own merit and demerit. Undoubtedly, there is no single approach that fits to all company requirements and attacker models. However, it is still helpful to find a combination of different approaches that serves best to a company's setup, attacker model, and budget. To ensure that no security related issue occurs after the software is released, software testers perform rigorous testing. And if a software engineers want to get best testing results, they should execute combinatorial security testing at an early stage of software development life cycle. The aim of the researchers in this field is to reduce the security related vulnerabilities by introducing different approaches of testing. However, it is impossible to restrict the income of new threats. Afterall, every hacker is a developer himself.

7. REFERENCES

[1] Common Vulnerabilities and Exposures - CVE-2015-4630. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-4630`, 2015. Date Entry Created - 20150616.

[2] Common Vulnerabilities and Exposures - CVE-2015-4631. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-4631`, 2015. Date Entry Created - 20150616.

[3] 10 steps to cyber security. *National Cyber Security Centre.*, page 1, August 2016.

[4] E. Alesand and H. Sterneling. A shoulder-surfing resistant graphical password system, 2017.

[5] J. Bi, P. Hu, and P. Li. Study on classification and characteristics of source address spoofing attacks in the internet. *International Conference on Networking*, 0:226–230, 04 2010.

[6] F. Breda, H. Barbosa, and T. Morais. Social engineering and cyber security. pages 4204–4211, 03 2017.

[7] Chawki, Darwish, Khan, and Tyagi. *Cybercrime, Digital Forensics and Jurisdiction.* Springer, 2015.

[8] P. Chen. Research on penetration test of the sql injection based on the formalization model. In *2nd International Conference on Electronics, Network and Computer Engineering (ICENCE 2016).* Atlantis Press, 2016.

[9] J. Dahse and T. Holz. Static detection of second-order vulnerabilities in web applications. In *USENIX Security Symposium*, 2014.

[10] M. Golla and M. Durmuth. On the accuracy of password strength meters. *CCS '18, Toronto, ON, Canada*, October 2018.

[11] P. A. Grassi, M. E. Garcia, and J. L.Fenton. Digital identity guidelines. *NIST Special Publication 800-63-3.*, pages 1–74, June 2017.

[12] S. Houshmand. Analyzing password strength and efficient password cracking. 12 2018.

[13] James.C.Foster, V. Osipov, N. Bhalla, and N. Heinen. *Buffer Overflow Attacks-Detect,Exploit,Prevent.* Syngress, New York, 2004.

[14] A. W. Marashdih and Z. Fitri. Cross site scripting: Detection approaches in web application. *International Journal of Advanced Computer Science and Applications*, 7, 10 2016.

[15] M. Mirjalili, A. Nowroozi, and M. Alidoosti. A survey on web penetration test. *ACSIJ Advances in Computer Science: an International Journal*, 3, 11 2014.

[16] B. Pandey, A. Singh, and L. Balani. Ethical hacking (tools, techniques and approaches). 01 2015.

[17] S. Pareek, A. Gautam, and R. Dey. Different type network security threats and solutions, a review. *IPASJ International Journal of Computer Science (IIJCS) ISSN 2321-5992*, 5:001–010, 04 2017.

[18] S. M. S. Sajjadi and B. T. Pour. Study of sql injection attacks and countermeasures. *International Journal of Computer and Communication Engineering.*, 02(05), September 2013.

[19] O. Segal. Automated testing of privilege escalation in web applications. *A whitepaper from Watchfire.*, pages 1–6, 2006.

[20] P. Sharma and B. Nagpal. A study on url manipulation attack methods and their countermeasures. *International Journal of Emerging Technology in Computer Science Electronics.*, 15(01):116–119, May 2015.

[21] D. E. Simos and B. Garn. Eris: A tool for combinatorial testing of the linux system call interface. *IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops.*, 49(10):58–67, April 2014.

[22] D. E. Simos, R. Kuhn, Y. Lei, and R. Kacker. Combinatorial security testing course. *HoTSoS '18, Raleigh, NC, USA*, April 2018.

[23] D. E. Simos, R. Kuhn, A. G. Voyiatzis, and R. Kacker. Combinatorial methods in security testing. *IEEE Computer Society.*, 49(10):80–83, October 2016.

[24] K. Thakur. Analysis of denial of services (dos) attacks and prevention techniques. 07 2015.

[25] B. Ur, F. Alfieri, M. Aung, L. Bauer, N. Christin, J. Colnago, L. F. Cranor, H. Dixon, P. E. Naeini, H. Habib, N. Johnson, and W. Melicher. Design and evaluation of a data-driven password meter. *CHI 1́7 Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3775–3786, May 2017.

Theories, Methods, and Tools in Domain Model Discovery

Past, Present, and Future

Marius Molz
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
marius.molz@rwth-aachen.de

Annika Rüll
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
annika.ruell@rwth-aachen.de

ABSTRACT

Programs are full of domain knowledge that we can use to refine the quality of the program. Different approaches have been used to discover such domain knowledge. Two main techniques emerged: Static and dynamic analysis. Static analysis focuses on the static information in the source code such as identifier names and comments, whereas dynamic analysis uses the knowledge contained in in-run-time memory and execution flow.

In this paper, we present an overview on the different approaches in domain model discovery. We introduce the general concept of static and dynamic analysis, followed by the different tools for extracting domain specific knowledge. Furthermore, we compare different approaches and evaluate their past improvements. In addition we give a short view over future development.

1. INTRODUCTION

Programs are used in business to solve specific domain problems. When business evolves, the software has to be able to evolve to the same extent as the business does. The use of domain-driven design (DDD) gives a skeleton which can easily be expanded and refined as required. Alas, code is often not written after the DDD principles, so restructuring and rewriting are difficult. Therefore, there is a demand for means discovering the domains implemented in the software. Since source code is usually written by domain experts, there is a lot of domain-specific knowledge in it. This raises the question of how to extract that knowledge.

The skeleton of source code consists of several classes, methods, and attributes connected with instructions. Running a program generates output data as well as new objects or other instances. Both types of data, which means the static ones in source code and the dynamic ones created during run-time, contain information about the corresponding domain.

The class, method and attribute identifiers have names which explain their role, hence they can contain domain knowledge. However, only in rare cases, an identifier name is one word. One challenge in analyzing identifiers is to find the correct natural language words for names consisting of multiple abbreviations. For domain models, not only the topics but also the connections between each other is important. Another challenge is to analyze indirect and ambiguously identifier relations. Apart from this, developers comment their code or display information to the user of the program. Comments can contain domain knowledge as well. They are more unstructured than identifiers, hence analyzing them needs more effort. These challenges are combined as static analysis. Based on the source code and other (static) files, the static analysis has all information of the domain. To interpret these pieces of information the semantic of the extracted strings has to be analyzed. That means, there is the need for algorithms understanding the semantic meaning of sentences.

The behavior of a program can be analyzed by looking at output data during run-time, which is called dynamic analysis. To gain run-time data, e.g., the program has to run with example input data and tests respectively. The selection which scenario to run is one of the challenges in dynamic analysis. However, running a program in a test environment can cause unexpected effects. Such observer effects should be avoided while analyzing a program. Another challenge in the dynamic analysis is that there is too much output data than a human can cope with.

In this paper, we present an overview of different domain model discovery approaches and focus on static and dynamic analysis. The first section introduce the main principle of identifier analysis and states techniques for splitting and expanding identifier names to natural language terms. Furthermore, we compare tools regarding correct discovered words. In the next part, we consider techniques analyzing relations between identifiers. The last part of static analysis deals with comment analysis. The main technique we explain is *Information retrieval*(*IR*), thereby we focus on *probabilistic models* in combination with different methods for domain discovery.

The second section describes dynamic analysis. We present various techniques to set up test cases. Relevant parts of source code can be found by analyzing how often they are executed. Otherwise, tests can be generated automatically, which give a good supplement for other techniques. To avoid false data which occurs by incorrect input, special heuristics are used. In the end, we state different tools using these techniques.

SWC Seminar 2018/19 RWTH Aachen University, Germany.

2. STATIC ANALYSIS

There are two main sources of domain model knowledge in source code: The identifier names and the unstructured data like comments or information that is displayed to the user. Identifier names contain much information about the problem domain. The challenge is to find the most useful meaning of the identifier name as well as to analyze the connections of the identifier compared to the real world relations.

As a result of their typically unstructured and arbitrary nature, comments are yet more difficult to analyze. Moreover, the algorithm has to recognize the meaning of the sentence and put it in the right context of the domain.

2.1 Identifier Analysis

To analyze the identifiers, the compiler creates a list of identifiers and the associated contexts. A context could be a scope, role or class. Based on this information, ontologies can be built and merged to create a domain model [21].

Analyzing the identifier name is the first part we focus on. As the names can be concatenations of abbreviations they have to be split and expanded to complete words. These words can be analyzed by their synonyms and translations to discover similarities with the problem domain. Using the associated context, we deduce the identifier-level relations. Matching those relations with their corresponding real-world counterparts, result in ontologies which form domain models.

2.1.1 Techniques For Examine Identifier Names

Because identifiers represent complex real-world problems, developers concatenate several words, which produces long identifier names. Therefore, identifier names are often abbreviations or acronyms. The challenge with domain model discovery is to find the real meaning of the identifier. An approach to replace identifiers is proposed by CAPRILE ET AL. [5]. However, it is not always useful to restructure the whole program. We state techniques of identifier analysis, followed by tools one can use.

Split names: Identifier names are often concatenations of different words like calcAge and first_name. Among others, CamelCase and underscores are used to denote the different words. The first step in analyzing identifier names is to split them on the basis of such separators.

```
Input  : Identifier $Id$
Output: List with split identifiers $S$
L $\leftarrow$ [$Id$];
if match ($Id$, ) then
    L $\leftarrow$ split ($Id$, );
end
foreach element $s$ in L do
    if match ($s$, CamelCase) then
        $S \leftarrow S \cup$ split ($s$, CamelCase);
    end
    else $S \leftarrow S \cup s$;
end
```

Figure 1: Hard split algorithm. Adapted from [6]

As the figure 1 shows, the identifier is first split at the special character (e.g. "_", "::", etc.), and second with the CamelCase. The match function returns true if the identifier contains a special character and CamelCase respectively. The identifier is split accordingly. As output we receive a list with the split identifier, e.g., the identifier *user_getStr* would be split into {*user*, *get*, *str*}.

Because not all concatenated identifier names mark the different words with such separators, some approaches also have a soft split function. This splits a name according to a given dictionary, e.g., *firstname* would be split into {*first*, *name*}. The approach from CARVALHO ET AL. [6] splits the name into different parts and scores them according to the dictionary entry. The division with the highest score would be chosen. FEILD ET AL. analyzes three different algorithms for soft splitting [14].

Expand names: As we see above *str* is not very expressive. Therefore, the next step is to expand the split identifier. Some identifier only contain program intern abbreviations, e.g., *cmpStr* should be expand to {*compare*, *string*}. Otherwise programmers also use domain specific vocabulary, e.g., *numOfChldn* abbreviation of {*number*, *of*, *children*}. Such expansions are probably not found with general purpose dictionaries.

Dictionaries dealing with (general) programming abbreviations and acronyms are proposed GUERROUJ ET AL.. Abbreviations will usually expand, but acronyms will only be recognized [17].

To expand specific names, a so called custom corpus-based dictionary must be created. In this we include

1. The identifier, extracted from source code
2. The general purpose dictionary, mentioned above
3. The documentation corpus

The documentation corpus is informally a collection of text, with domain specific content. To create a corpus the following sources can be used:

- documentation files, such as JavaDoc or simply plain text
- `README` files or `INSTALL` files, usually containing also domain specific knowledge
- other content of plain text files

The collected data is stored in a plain text file [6].
For the decision, which expansion of the identifier name is correct, a scoring function is used. On the basis of the dictionaries, the scoring function checks the occurrence of the expansions and ranks them. The highest ranked expression will be chosen. Figure 2 depicts the splitting and expanding procedure, with its soft and hard splits.

2.1.2 Tools For Examine Identifier Names

In the last twenty years the concept of identifier names analysis has not significantly changed. CAPRILE ET AL [5] states in 1999 how to split and expand identifier names using dictionaries. Published nine years later, the tool `Samurai` [12] uses dictionaries as well. However, they also implement the mentioned scoring function. For that reason the resulting expansions are more accurate than in the first approach. `TIDIER` [16] in 2010 not only makes use of dictionaries, but use a quite similar custom corpus-based dictionary as mentioned above too. In addition, the technique of

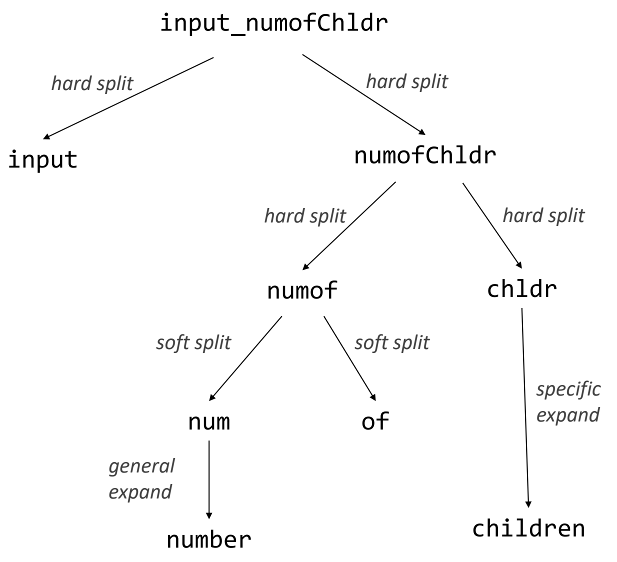


Figure 2: An example of identifier examination.

Table 1: Precision of different approaches by analyzing `Lynx`. Adapted from [6]

Metric	Approach	Mean	Median
precision	Samurai	0.47	0.50
	TIDIER	0.86	1.00
	TRIS	0.93	1.00
	LIdS	0.85	1.00
recall	Samurai	0.45	0.33
	TIDIER	0.84	1.00
	TRIS	0.91	1.00
	LIdS	0.86	1.00

splitting identifier names and finding expansions is based on computing distances between the identifier and words found in dictionaries. The algorithm is inspired by speech recognition. The same authors published two years later `TRIS` [17]. This approach uses a set of dictionaries too. The splitting and expansion process is handled as an optimization problem: First a set of word transformations including costs, is created. Then in the next step `TRIS` searches for the optimal path in the expansion graph. `LinguaId::Splitter(LIdS)` [6] in 2014 is one of the newest tools in identifier analysis. It uses the same principle as `TIDIER`, but is able to split one word abbreviations as well. Table 1 shows a comparison between the capabilities of correct splitting and expanding of the different tools [6, 17]. The row *precision* means the number of correct split identifier names divided by the absolute number of computed splits, while *recall* has been calculated as the amount of correct splits divided by the number of all correct splits. As we see in table 1, `TRIS` achieved with an average of 93% the best precision. The approach `TRIS` uses is slightly different, as it uses path optimization. Although `LIdS` is a newer tool, it has a precision of 85%.

2.1.3 Techniques For Examine Identifier Relations

When the source code is parsed, we can not only collect the identifier name, but its context and relation as well. Therefore we are able to gain significant information, using the identifiers context. There are two main methods in using this information: First, we examine the relations between the identifier (e.g. isA, hasA relations). This gives us ontologies, and collected a domain. The use of dictionaries like `WordNet` can help to validate and complete the domain concept [22].

Otherwise, we use the type of the identifier, e.g., class, function, variable), to find the grammatically corresponding class, e.g., verb, noun). Using Natural language parsing, sentences were created and analyzed to discover the concept of the program [1, 2].

Matching of relations: Especially in object-oriented programming languages, we can distinguish between module system induced relations and type system induced ones. In module systems there is, among others, the *memberOfPackage*, which is used to describe connections between the classes and interfaces. The attributes belonging to a class, are described with the *memberOfClass* relation.

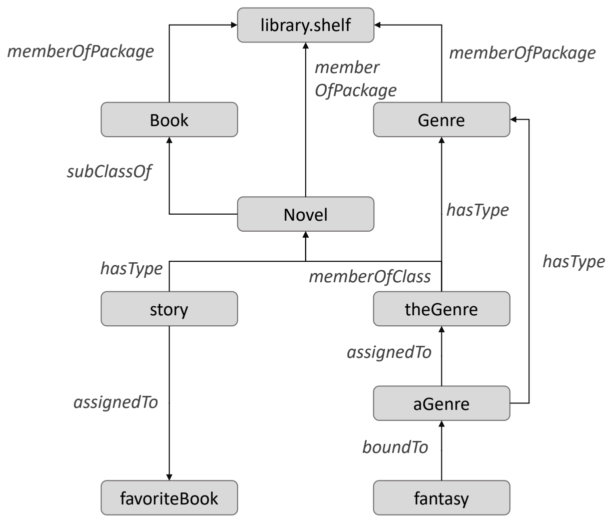
The most important relations in type system are the *supTypeOf* relation, stating the relation between two types and *hasType*, holding between a variable and its type. Relations between variables are *assignedTo* relations and, quite similar, *boundWith* relations if the former parameter is assigned to the actual parameter [22]. Figure 3 depicts the relations from a sample code snippet. To determine what the appropriate real world relation is, `WordNet` is a useful dictionary. `WordNet` groups the words by synonyms and contains additionally lexical concepts. `WordNet` distinguished between hypernym and meronym, e.g. wing is meronym of bird. The example in figure 3 gives among others the following relations:
Book is hypernym of *Novel* and *story* meronym of *Novel.*

Natural Language Parsing: Another approach makes use of Natural Language Parsing (NLP). In contrast to the previous techniques, the identifier names were divided into their types and from this, sentences can be generated. The proceeding is as follows:

1. Creating a term list
2. Generating the corresponding sentence
3. Selecting the appropriate sentence
4. Extracting the concepts and relations

Creating the term list can be realized as stated in the previous subsection. To generate the sentence, there are several rules (cf. table 2) to apply. Identifiers are distinguished by class (C), method (M) and attribute (A) identifiers. After generating the sentences, there are at most two sentences per term. In step three a sentence is chosen as follows:

- If only one sentence contains $U(Unknown)$, then select the other
- Select *MR2*, when there is a match with the attributes of the enclosing class
- Select highest frequency in term list
- Select highest user defined priority



```
package library.shelf;
class Genre {…}
class Novel extends Book {
      Genre theGenre;
      Novel (Genre aGenre) {
            the Genre = aGenre;
      }
}

Genre fantasy = new Genre();
Novel story = new Novel(fantasy);

…

favoriteBook = story;
```

Figure 3: An example of identifier concept identification. Adapted from [22]

In the last step, the relations between identifier were taken into consideration. The ontologies are obtained by mapping the linguistic relations to ontological relations. The target ontological relations and the corresponding natural language dependency relations to which they are mapped are described below:

- isA: Connection between general to more specific concepts
- <verb>: Context specific relation between concept and the object connected to the verb
- hasProperty: Relationship between concept and its properties

With this, the ontologies can be created and merged to domain models [1, 2].

Table 2: Rules for sentences generation. Reprinted from [1]

Rule	Identifier	Generated sentence	Constraint
CR1	$C =< T_1 >$	T_1 "is a thing"	T_1 is a noun
CR2	$C =< T_1 >$	T_1er "is a thing"	T_1 is a verb
CR3	$C =< T_1, < T_2 >, ... >$	T_1T_2... "is a thing"	T_1 is a noun
CR4	$C =< T_1 >$	T_1ingT_2... "is a thing"	T_1 is a verb
MR1	$M =< T_1 >$	"subjects" T_1 "object"	T_1 is a verb
MR2	$M =< T_1 >$	"subjects get" T_1	T_1 is a noun
MR3	$M =< T_1, < T_2 >, ... >$	"subjects" T_1T_2...	T_1 is a verb
MR4	$M =< T_1 >$	"subjects get"T_1T_2	T_1 is a noun
AR1	$A =< T_1 >$	T_1 "is a thing"	T_1 is a noun
AR2	$A =< T_1 >$	T_1er "is a thing"	T_1 is not a past tense verb or T_1 is a past tense verb and A is not a boolean
AR3	$A =< T_1 >$	T_1 "subjects are things"	T_1 is a past tense verb and A is a boolean
AR4	$A =< T_1, < T_2 >, ... >$	T_1T_2... "is a thing"	T_1 is a noun
AR5	$A =< T_1 >$	T_1ingT_2... "is a thing"	T_1 is a verb

Filter domain specific knowledge: A Disadvantage of this approach is that the *NLP* does not distinguish between implementation details and domain concepts. For that reason, five years later, ABEBE ET AL. propose an extension with filter techniques, dealing with concept filtering. The added filter techniques are *non-interactive* and *interactive keyword based* filtering, as well as *topic based* filtering. The *keyword based* filtering approach detects keywords in the documentation by using term frequency. This considers terms that have high frequency as keywords. *Interactive keyword based* filtering does a closer examination, as it excludes GUI and user explanation keywords [2].

A concept in the ontology is therefore represented by the keywords of a given topic. The concept is kept in the filtered ontology if all the keywords match to each other. The result is two sets of terms corresponding to the domain and the implementation.

2.2 Analysis of unstructured data

At this point, we have analyzed the identifier names and their relations. There is still more information in the source code. The code in figure 4 contains domain knowledge in attribute and method identifier names, but there are a comment and a string with specific knowledge as well. Other than the identifiers, the comment and the exception description are unstructured. There is no clue where and how many comments appear in the source code. Additionally, it is more effort to analyze comments and other strings because they are usually sentences which have to be analyzed semantically.

```
// Is it morning?
public boolean isMorning(int hours, int minutes, int seconds){
   if (!isDate(hours, minutes, seconds)){
     throw Exception(”Invalid input: not a time value.”);
   }
   return hours<12 && minutes<60 && seconds<60;
}
```

Figure 4: Example code. Reprinted from [19])

2.2.1 *Techniques For Examine Comments*

Comments are scattered all over the source code. Usually, they are very helpful for domain model discovery, but very unstructured and hence difficult to analyze. The feasibility of comment analysis depends strongly on the quantity and content of comments in source code. Another challenge is the definition of the scope of comments. Chen et al. propose an approach from 2018, dealing with this problem [7]. In older approaches, the context of comments is usually the code in the lines around.

Primary methods in comment analysis make use of information retrieval techniques (*IR*). We state the general *IR* approach, followed by two tools `DARIUS` and `PATRicia` using this technique.

Information retrieval technique: The first step is named *indexing*. Gathering the information of the document a logical view of the document can be created. The *retrieval* is the second step, where the query is processed and returned as a ranked list. In detail, the procedure is as follows

1. Preprocessing of the source code (i.e. the comments) and the documentation
2. Creating a logical view (*indexing*)
3. Execution of queries
4. Retrieving and analyzing the results
5. Returning of the results as a ranked list

There are three mathematical models for this process: The *Set-theoretic*, *Algebraic* and *Probabilistic Models*. Especially the last model group is used in comment analysis [15, 20].

2.2.2 *Tools For Examine Comments*

Although the following tools both implement *IR* techniques, the underlying analysis of identifier names is different. `PATRicia` uses *NLP* techniques, while `DARIUS` uses dictionaries.

PATRicia System: The **P**rogram **A**nalysis **T**ool for **R**euse analyzes comments and identifiers with natural language processing techniques. The system uses a sentence parser and a semantic processor. The sentence parser has information of standard comment syntactic structures and identifier formats. With a special, for this purpose created knowledge base, the extracted comments can be analyzed semantically. `PATRicia` treats comment understanding and code understanding individually. To understand comments, occurrences of words are identified and the relationships between them are concluded to detect concepts. With the analyze of the code structure and the order and nature of the statements on program, `PATRicia` infers further concepts. If concepts appear in both, comment and code understanding, then a domain is found [13, 15].

DARIUS: `DARIUS` finds the information about the domain by using a pdf file, in which the programmer has to define the problem domain. With two IR models, `DARIUS` then searches through the program's comments and returns the corresponding data.

Because of the various quality of code comments, `DARIUS` preprocesses the comments as follows [15]:

- Comment Extractor: extracts the comments keeping their context
- Statistics Calculator: provides quantitative results regarding comments in a program
- Comment Word Analyzer: computes the frequency of words on comments
- Graphical Interface: can be used to visualize all the information provided

2.2.3 *Further Techniques and Tools*

As seen in figure 4, there is more information in the source code than identifier names and comments. The comment *//Is it morning?* states the meaning of the function and is relevant for the domain, while *"Invalid input: not a time value"* is not. In this last section of the static analysis, we state approaches to analyze these unstructured data. The use of *IR* techniques is central, as mainly all techniques based on *probabilistic models*, namely *Latent Dirichlet Allocation*(*LDA*) [3, 24] and *Latent Semantic Indexing* (*LSI*) [19], are used.

LDA and TopicXP: The tool `TopicXP` uses *LDA* for analyzing all natural language terms in source code:

1. Pre-processing words
2. Extracting LDA-Topics
3. Assigning documents to topics
4. Extracting dependencies
5. Computing maximal weighted entropy

The first step is similar to the identifier analysis in section 2.1. Instead of expanding the words, they get stemmed, e.g., by `Snowball` a language for stemming algorithms) to more easily recognize individual words. Using these words, we compute the topics with *LDA*. After assigning the documents to the topics, the linguistic dependencies are computed and visualized (e.g. by `X-Ray`). Finally the maximal weighted entropy for each of the classes in the system is analyzed. This is because a class usually contains only a few topics [24].

LSI and semantic clustering: As with most information retrieval techniques, *LSI* is based on the vector space model approach. This approach models documents as bag-of-words and arranges them in a term-document matrix A such that a_{ij} equals the number of times term t_i occurs in document d_j [19]. *Semantic clustering* is a approach that uses the results of *LSI* to create the domain model. After applying *LSI* the clustering steps are as follows:

1. Identifying topics
2. Describing topics with labels and apply *LSI* again
3. Compare the topics to the structure

The clustering results in linguistic topics. By using an algorithm which groups similar elements together, clusters are aggregated. The source documents are consequently partitioned in their vocabulary, but there is no indication about the concrete connections between the documents. Therefore, the topics are labeled, accordingly to their relevance. We use *LSI* for this step again. Finally we have to compare the program structure to the discovered topics, by using a Distributed Map. This visualizes the correlate linguistic information with structural information [19].

3. DYNAMIC ANALYSIS

A rather new kind of analysis, the *dynamic analysis*, does not only take the source code into consideration but also analyzes the actual run-time behavior. This information is more valuable in specific situations because especially for those, someone can create tests that can be run in a test environment. Custom scenarios are the prerequisite of the analysis, and scenarios can be processed using different theories. Because of that, there are several techniques one can choose from, each one having its strengths at another kind of scenario [4]. *Dynamic analysis* aids the programmer to more detailed information regarding run-time behavior, because a much smaller, well-defined scope is analyzed [8]. Hence the *dynamic analysis* has a few advantages, but also drawbacks:

3.1 Features

Other than the *static analysis* (where the analysis must stay as abstract as possible), the *dynamic analysis* can test execution traces on its own, giving it the ability to provide much more detailed (and precise) output to each scenario / test case. Because of that, the results of the analysis have a much higher usability, containing highly-accurate information over a broad set of test-cases. This refers to the *precision* of the analysis [8, 4].

Because the *dynamic analysis* is not limited in scope (but in run-time), output regarding wide-spread semantic dependencies can be generated. The *static analysis* cannot achieve this, because it is limited to a specific *scope*. In literature, this is also referred as *goal-oriented strategy* meaning "[...] that only the parts of interest of the software system are analyzed" [8]. Therefore, *dynamic analysis* of a program must be clearly organized, to really obtain all information needed [4].

3.2 Drawbacks

The analysis of one specific scenario naturally only captures the run-time behavior (execution traces) in the current scope, but only one out of the typically infinite set of possible execution traces of the program's source code. Because of that, the results of the analysis regarding the whole program can be *incomplete* [8, 4].

Besides, analyzing different scenarios creates a large amount of output that needs to be processed. The more specific the scenario is, the more information one has to understand in the entire program's scope. Because of that, the user cannot simply use these techniques, but also must prioritize because otherwise the person could not process all data generated. The User would have difficulties *scaling* the output [8].

The *observer effect*: Sometimes the execution of the program in a test environment leads to different results or output than under normal (real-world) conditions, even if the scenarios are the same. This is related to a change of behavior of e.g., memory or execution traces. Sharing the same run-time environment with e.g., monitoring software or memory analyzers also influences the program flow [8].

3.3 Techniques

Because of the same demands as stated above, the *dynamic analysis* provides multiple theories of analysis, where each one has its strengths at another style of processing a specific type of scenario. Many different techniques can be derived from them. Therefore, a few techniques are presented in the following section, each having its main focus on other approach of analyzing the program dynamically.

3.3.1 Frequency Spectrum Analysis

Developed in 1999 and based on program profiling, the *Frequency Spectrum Analysis* (FSA) stores information about how frequently executed some parts of the program are. Someone then can retrieve differently clustered information, such as a partitioning of the program by levels of abstraction, a set of related computations (that are based on the same specific program's output or input) [4].

The FSA itself consists out of three rudiments which can be analyzed on its own [4]:

1. Atomic (regarding the program's functionality) functions or actions are called a lot more frequently than one-time functions such as the program's initializing methods. Therefore, a hierarchy of software abstractions can be derived simply by sorting low frequencies against high frequencies.

2. One can retrieve dynamic relationships between parts of the program which does not need to be visible in the static analysis (e.g., source code). If some methods or variables are called at the same frequency, it means that (with high probability) they are in a *Frequency Cluster*, meaning they both share that same frequency of program calls.

3. The analysis also detects specific Frequency Patterns. As the complexity of run-time, there surely are parts of a program correlating e.g., with the length of the input. These dependencies then can outline code fragments which directly influence the program's output or input processing.

3.3.2 Coverage Concept Analysis

Developed in 1940 [26], the *Coverage Concept Analysis* identifies groups of similar parts of the program. A set of relations is used an input. A set is represented by a tuple of objects and their corresponding attributes, whereas the objects are tests and the attributes their results. From that, *Concepts* are generated; a *Concept* "[...] is the pair *(T,E)*, where T is a set of tests and E a set of program entities [...] if every test in T covers all entities in E [...]" [4]. Hence *Concepts* group all sets of tests which are covering identical objects.

Based on this set of relations, the user then can easily detect which tests are redundant, and even detect which entities of the program are not covered by at least one test. Additionally, tests can be run automatically [25]. As shown in [26], high-level tables containing the relations can be derived, while only having a few limitations such as relations between an objects instance and its subobject's multiple instances. In this case, the detected relations can not be linked to a specific subobject, but only the main object. SIFF ET AL. [25]proposes that *Concept Analysis* can also be used for Module Detection, showing multiple possible modularizations. From there, so-called *Concept Partitions* can be derived, whereas a partition corresponds to a collection of modules "[...] such that every function in the program is associated with exactly one module" [25].

Combined with FSA, this provides much more useful information for the Discovery of previously unknown (e.g., unchecked) code [4].

3.3.3 *Software Reconnaissance*

This technique from 1996 takes a slightly different approach as the ones mentioned earlier: Other than mapping sets of test to their set of test results, *Software Reconnaissance* compares whole traces of different test cases. For this purpose, traces are generated out of the components of the specific test first. Then, test cases both having or missing the corresponding feature in their scope are run (like in the *Concept Analysis*). Based on this results, the correct trace can be received (containing the feature), simply by looking for components of the program that are executed in the first (inclusive), but not in the second (exclusive) set of tests [27].

This is most useful for detecting *execution branches* occurring in an execution trace for a specific feature, which is also called a *masker code*. For that, the user must only take "[...] the set of branches that are executed in some test case [...] removing any branches[...]" [27] where that specific feature does not occur in the testing scope.

Since the feature detection works on all tests provided, the *Software Reconnaissance* technique has its strength at detecting features correctly which are wide spread across large parts of the program's source code (e.g., still maintained legacy code). Nevertheless, this technique (like *Concept Analysis* still requires both test cases.

3.3.4 *Dynamic Feature Traces*

In terms of scoping the feature these techniques only build up binary relationships, that heavily rely on the correctness of the input [11].

By applying special heuristics, also non-binary relationships can be used to detect features. These so-called *Dynamic Feature Traces*(2005) are artifacts that rank the detected code-fragments - implementing that feature by their relevance in the source code. Automated feature location is possible, because these DFTs are also compatible to Test Driven Development, whereas binary analysis forces the programmer to create enough additional tests, not containing the specific feature to be checked against [11].

A DFT consists of *ranks* (which is an ordered set of methods called during the execution) and the *call* set which contains some information about the context of the method call [11].

Because of the automatic creation of the DFT, the user must only provide feature mappings to run against the given test suite. If tests are not mapped to a feature, they then are grouped to similar tests (by similar structure) automatically [11].

After that, there must only be a test suite "[...] where all relevant features are tested in at least one case" [11]which consists out of one containing- and many non-containing test sets for each feature to be tested [11].

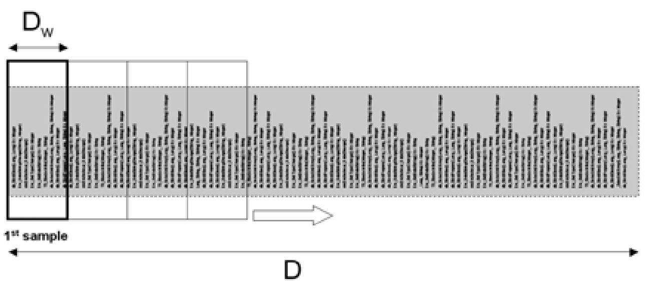


Figure 5: Sampling Window (reprinted from [10])

3.4 Tools

Based on these techniques, tools are developed, whilst their main purposes differ to each other. To give an broad overview, some tools, using different approaches of implementing some technique, are presented.

3.4.1 *Dynamic Correlation*

A tool derived from modified versions of the FSA, the application from DUGERDIL [10] tries to minimize the unneeded output of the FSA much more. For that, the output needs to be reduced to the critical data the user shows interest in. This is achieved by filtering out *noise*, that are elements executed in the period of time while the feature was in the scope. This elements then are mostly called (in terms of absolute calls) outside of the feature scope. This is a major difference to the FSA, where only frequencies are analyzed, but not the distribution of frequencies at the time of execution [10]. By also taking the Distribution of the frequencies into consideration, DUGERDIL claims to achieve an easier analysis of legacy systems, mapping high-level to low-level functions, and an easier recovery of the high-level architecture. For that, so-called *Samples* are created during the analysis. These are the time slices (cf. figure 5) the feature could be detected in. A so-called sampling window of a given time size (depicted as D_w in figure 5) moves over the execution trace (with run-time D). Inside the scope set by the moving sampling window, an algorithm can distinguish between noise and elements that really refer to the feature and remove unnecessary information from the output which otherwise would have been needed to be processed [10]. After sampling, dynamic correlated elements can be detected. These are elements that are occurring in the same samples. By measuring the number of the shared samples, one can then define their grade of correlation. Afterwards, the clustering of elements can be computed based on the results of the dynamic correlation analysis [10].

3.4.2 *Visualizing Recurrent Code*

Another attempt of making the huge amount of output data processable by human readers is the visualization of duplicate code fragments in the execution trace. To detect these code fragments a two-dimensional matrix is used, whereas entries in the matrix represent string matches in the lines of code (cf. figure 6). If diagonal lines in this matrix are present, the program's source code contains duplicated code fragments. Based on this matrix, multiple visualizations, each designed for a different purpose, can be created:

- *Recurrent pattern visualization* shows reoccurring code fragments as clear patterns. From that visualization,

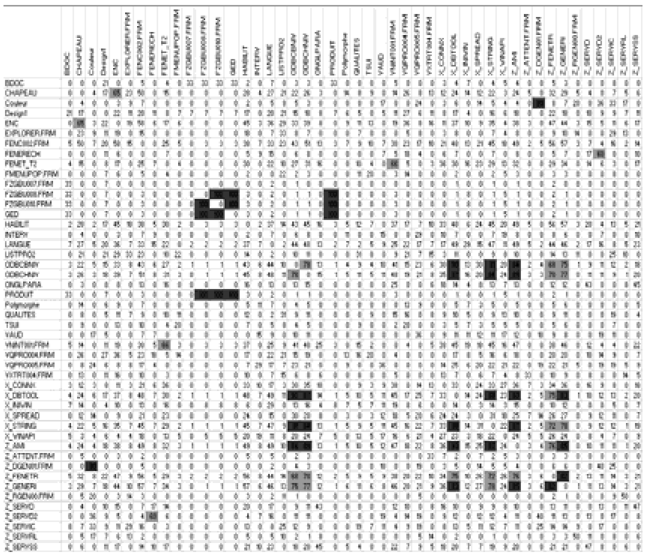


Figure 6: Feature Trace Correlation Matrix (reprinted from [10])

new approaches of solving this redundancy can be derived.

- The *Execution phase visualization* makes it possible to detect so-called execution phases from trace fragments. With that knowledge a more focused investigation is possible.
- The *Polymorphism visualization* detects "[...] recurrent call traces of which the calls differ only slightly[...]" [10] by tweaking the comparison algorithm of two events.

3.4.3 GCOV

With the tool `GCOV`(*GNU Test Coverage Program*), it is possible to analyze the coverage of the code. `GCOV` can be run using two different modes:

- In the first mode, called *Statement Coverage Analysis*, the code is broken down into basic blocks "[...] which are the blocks of code that exist between branches." [18] Because they only appear in one branch, all the content of the basic block is executed at the same frequency. The analysis then checks the number of calls from each statement.
- The other method, called *Branch Coverage Analysis*, just overviews how often the branches themselves are executed. To do this, each branch is analyzed to derive which branches will be executed in what possible combination and order; from there, also *Statement Coverage* related data can be generated. By creating tests for each possible combination of executed branches, even complex branch patterns can be coverage checked using this method.

`GCOV` then runs a pre-prepared code file and annotates the Statements with their number of total executions [18].

3.4.4 PANDA

With `PANDA`(*Platform for Architecture-Neutral Dynamic Analysis*), a `QEMU`-based system, the user has many possibilities how to analyze the program's binary code and how the output is parsed afterwards [23]. Commonly used for reverse engineering of legacy code, the system relies on the dynamic analysis of execution traces. For that, `PANDA` has the ability to replay specific samples of the execution trace: Users can iterate over the code that may contain the feature and pinpoint the exact location. This is possible because the system's replay consists out of snapshots of the guest's system memory and CPU state [9].

In contrast to common analysis tools, `PANDA` can also analyze the program backwards. That means, the debugger holds information about its previous states, which can be reloaded such that the program can be processed backwards through its execution trace. This is not common, often the user must rerun the debugger to debug an earlier program state [9].

Because the system of `PANDA` is based on `QEMU`, the whole analysis itself stays architecture-neutral, leading to a dynamic analysis which even scopes different run-time behavior of the program based on different architectures [23]-

Overall, this tool has a few advantages compared to most other tools [9, 23]:

- Open Plugin System: Plugins can be easily written and (un)loaded live, while the code is analyzed
- Architecture-Neutrality: Because multiple Architectures can be emulated, even different behavior based on multiple architectures can be detected
- No normal debug system: Because of the unique debug system, execution traces can also be analyzed backwards.
- Because of the replay system, one can close in on a specific feature, by reducing the size of the scope in each iteration of analysis.

4. FUTURE DEVELOPMENT

With the growth of software systems and their regular refinement in practice, domain model discovery has become an increasingly important part of software development. Identifiers are the predominant knowledge source in domain model discovery. For this reason, the identifier analysis should be precise. Existing approaches achieve a precision up to an average of 93% which is highly accurate. The precision depends mainly on the underlying dictionaries. Neuronal networks, trained with these dictionaries, could improve the accuracy of identifier analysis. Avoiding noise is another point that is not solved satisfactorily. Not all parts of programs contain domain knowledge. Therefore, there is a need for techniques which filter results so that only domain knowledge, and not program intern knowledge, is collected. Currently criteria for non-domain knowledge are still missing.

Regarding the future development in the field of dynamic analysis, we imagine new tools and techniques will be developed that make real-time analysis in software development possible. Because computers will become even more powerful, the ability to analyze (and reverse engineering) obfuscated code will also grow. Taking the recent achievements in the field of cloud computing and virtualization into consideration, we expect that future tools will be capable of distributing e.g., their internal tasks of analysis, workload across multiple processor units. Based on that, even the analysis of multi-threaded programs could be improved, when multiple execution traces can occur simultaneously. This is also the case for the earlier mentioned problem of analyzing multiple subobject's instances of the same object, because nowadays, retrieved data can not be connected to the related code fragment out of one of these objects.

Finally, neither static nor dynamic analysis is able to deduce the whole domain knowledge included in a program. A significant improvement can be achieved by combining both types of analysis. Unfortunately, this approach is not common in literature.

5. CONCLUSION

In the first section of the paper, we described how static analysis works and how it retrieves the domain model of the program's code. We started with simple identifier names analysis over to more complex unstructured data analysis. Using dictionaries we are able to split and expand identifier, which ensures to analyze meaningful names. Combined with the analysis of relations and identifier types, we have already a strong knowledge base. The tools we have introduced have a precision up to average 93%. Interesting in this context is, that within time the techniques changed and may improve, but the precision does not improve to the same extent. Furthermore, the use of dictionaries containing real-world relations, enable the mapping of the program entities counterparts. To avoid ambiguous identifier, they are transformed into natural language sentences. Than *Natural Language Parsing* clearly analyze the sentences. Together with filter techniques noise and other domain irrelevant content do not appear in the domain model. Afterward, we explained how information that is not related to identifiers is processed. To mention here are comments and other strings in the source code. The main technique to deal with this is *Information Retrieval*. We finally had a closer look at two probabilistic models from IR and how to extract domain-specific knowledge with them.

In the second part of the paper, we explained the purpose and usage of dynamic analysis. For that, we give an overview of techniques, which one can use to find test scenarios. After describing techniques like *Frequency Spectrum Analysis*, *Coverage Concept Analysis*, *Software Reconnaissance* and the concept of *Dynamic Feature Traces*, we listed some tools used for dynamic analysis. The *Visualization of recurrent code* is a solution for scalability. By visualizing results, the human reader can cope with the data. Lastly, we described how tools like `GCOV` and `PANDA` can be used to analyze the program's source code (or even the program's binary, through reverse engineering) to detect the domain model dynamically.

There are several already powerful methods in domain model discovery. However, combining different approaches, especially static and dynamic methods, would produce better results. Alas, such approaches are very rare in literature.

References

[1] S. L. Abebe and P. Tonella. Natural language parsing of program element names for concept extraction. In *IEEE International Conference on Program Comprehension*, 2010.

[2] S. L. Abebe and P. Tonella. Extraction of domain concepts from the source code. *Science of Computer Programming*, 98:680–706, 2015.

[3] M. Alenezi. Extracting high-level concepts from open-source systems. *International Journal of Software Engineering and its Applications*, 9(1):183–190, 2015.

[4] T. Bell. The concept of dynamic analysis. *ACM SIGSOFT Software Engineering Notes*, 1999.

[5] B. Caprile and P. Tonella. Restructuring program identifier names. In *Proceedings 2000 International Conference on Software Maintenance*, pages 97–107, 2000.

[6] N. R. Carvalho, J. J. Almeida, P. R. Henriques, and M. J. Varanda. From source code identifiers to natural language terms. *Journal of Systems and Software*, 100:117–128, 2015.

[7] H. Chen, Z. Liu, X. Chen, F. Zhou, and X. Luo. Automatically Detecting the Scopes of Source Code Comments. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, pages 164–173. IEEE, 7 2018.

[8] B. Cornelissen. Evaluating Dynamic Analysis Techniques for Program Comprehension. *PHD Dissertation*, 2009.

[9] B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, and R. Whelan. Repeatable Reverse Engineering with PANDA. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop on - PPREW-5*, 2015.

[10] P. Dugerdil. Using trace sampling techniques to identify dynamic clusters of classes. In *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research - CASCON '07*, 2007.

[11] A. D. Eisenberg and K. De Volder. Dynamic feature traces: Finding features in unfamiliar code. In *IEEE International Conference on Software Maintenance, ICSM*, 2005.

[12] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 71–80, 2009.

[13] L. H. Etzkorn, L. H. Etzkorn, L. L. Bowen, and C. G. Davis. An Approach to Program Understanding by Natural Language Understanding. *NATURAL LANGUAGE ENGINEERING*, 5:1–18, 1999.

[14] H. Feild, D. Binkley, and D. Lawrie. An empirical comparison of techniques for extracting concept abbreviations from identifiers. In *Proceedings of IASTED International Conference on Software Engineering and Applications (SEAï£¡06)*, 2006.

[15] J. L. Freitas, D. d. Cruz, and P. R. Henriques. A Comment Analysis Approach for Program Comprehension. In *2012 35th Annual IEEE Software Engineering Workshop*, pages 11–20, 2012.

[16] L. Guerrouj, M. Di Penta, G. Antoniol, and Y.-G. Guéhéneuc. TIDIER: an identifier splitting approach using speech recognition techniques. *Journal of Software: Evolution and Process*, 25(6):575–599, 2000.

[17] L. Guerrouj, P. Galinier, Y. Guéhéneuc, G. Antoniol, and M. D. Penta. TRIS: A Fast and Accurate Identifiers Splitting and Expansion Algorithm. In *2012 19th Working Conference on Reverse Engineering*, pages 103–112, 2012.

[18] N. Hinds, P. Larson, H. Franke, and M. Ridgeway. Using Code Coverage Tools in the Linux Kernel.

[19] A. Kuhn, S. Ducasse, and T. Gîrba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243, 3 2007.

[20] A. Marcus, V. Rajlich, J. Buchta, M. Petrenko, and A. Sergeyev. Static Techniques for Concept Location in Object-Oriented Code. In *13th International Workshop on Program Comprehension (IWPC'05)*, pages 33–42. IEEE, 2005.

[21] M. J. Pereira, M. Berón, D. Cruz, N. Oliveira, and P. Henriques. Problem domain oriented approach for program comprehension. In A. SimÃţes, R. Queirós, and D. da Cruz, editors, *Symposium on Languages, Applications and Technologies (SLATEâĂŹ12)*, pages 91–105, Wadern, 2012. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[22] D. Ratiu and F. Deissenboeck. Programs are knowledge bases. *IEEE International Conference on Program Comprehension*, 2006:79–83, 2006.

[23] Ryan Whelan, Tim Leek, and David Kaeli. Architecture-Independent Dynamic Information Flow Tracking. Technical report, European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, 2013.

[24] T. Savage, B. Dit, M. Gethers, and D. Poshyvanyk. TopicXP: Exploring topics in source code using Latent Dirichlet Allocation. In *2010 IEEE International Conference on Software Maintenance*, pages 1–6. IEEE, 9 2010.

[25] M. Siff and T. Reps. Identifying Modules via Concept Analysis. Technical Report 6, University of Wisconsin, Wisconsin, 1999.

[26] G. Snelting and F. Tip. Reengineering Class Hierarchies Using Concept Analysis. In Anonymous, editor, *SIGSOFT *98*, pages 99–110, Braunschweig, 1998. ACM.

[27] N. Wilde and C. Casey. Early Field Experience with the Software Reconnaissance Comprehension Technique for Program Comprehension. In N. Wilde and C. Casey, editors, *Proceedings of WCRE '96*, pages 270–276, Pensacola, FL 32514, 1996. University of West Florida, IEEE.

Towards a Catalog of Design Patterns for Domain Modeling

Vitalii Isaenko
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
vitalii.isaenko@rwth-aachen.de

Erik Pinders
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
erik.pinders@rwth-aachen.de

ABSTRACT

The domain model captures our understanding of real world entities, their relationships and responsibilities in a given problem domain. This artifact can be used to communicate domain knowledge to all involved parties. Furthermore, the quality of the domain model influences the degree to which an application satisfies business requirements.

The importance of a high-quality domain model can be observed in complex software solutions, which tend to miss their business requirements if the underlying domain model is poorly designed, leading to various problems such as lost business value and technical debt impacting the evolvability of the business. To support domain experts in building a high-quality domain models, design patterns may help them to better understand the core requirements of the domain that is being modeled.

But most of the existing design patterns are tackling system-level complexity, and their applicability in domain modeling is yet unknown. Despite the emergence of the domain driven design approach there is a lack of patterns for domain modeling. To solve this problem, this study explores existing design patterns and assesses their applicability in domain modeling.

Keywords

Design Patterns, Domain Modeling, Domain Model, Factory Method Pattern, Publish-Subscribe Pattern, Composite Pattern, Mediator Pattern, Prototype Patter, Strategy Pattern, Iterator Pattern, Object Pool Pattern, Chain of Responsibility Pattern, Decorator Pattern, Software Development

1. INTRODUCTION

Building high quality software solutions is a hard and complex process, which requires knowledge about domain specific topics and technical implementation details.

Over the last decades software design patterns have evolved to a staple of most software developers to tackle complexity on a technical level. Due to their acceptance in software development, software design patterns are well understood and researched and there is a wealth of information summarizing and describing each pattern and its applicability.

However, solving implementation problems is only one part of a successful software solution. Providing a solution for an underlying business problem is in a lot of cases much more difficult.

There are many ways to help participants of a software project to understand the underlying business problems. One of the first activities in a new project is called domain modeling. During domain modeling participants try to build the so called domain model which should capture the understanding of real world entities, their responsibilities and relationship among them.

Analogous to software design patterns, one could expect the existence of such patterns for domain modeling, as it is quite a complex process. However, too few of design patterns for domain modeling are discovered so far.

Therefore, initiating a catalog of patterns for domain modeling is the target of this research. There are several quality factors of a domain model that we will try to tackle in this paper. Most important are evolvability, understandability and descriptiveness of the resulting model. The three quality factors were chosen as the most valuable ones for a good domain model.

The approach that is chosen in the research for building a catalog of design patterns for domain modeling consists of analyzing existing design patterns in software development and adapting them to domain modeling. Many patterns from different fields were analysed in advance but the final catalog contains only adaptations of object-oriented design patterns as the most relevant ones.

2. BACKGROUND

Before discussing adaptations of existing design patterns for domain modeling we will have present the background of our research. Especially some background knowledge regarding design patterns and domain modeling will be beneficial over the course of this paper. Design patterns and domain modeling will be two main topics of the following sections.

2.1 Design Patterns

A software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design, and using design patterns is often considered to be a best practice for solving common problems. But design patterns are not finished designs that can be implemented without additional work, its rather an abstract description for solving a problem in different situations.

After Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published ”Design Patterns: Elements of Reusable Object-Oriented Software” in 1994, design patterns gained popularity. Since then design patterns have become a staple of every professional software developer.

Following that book, there were many different design patterns discovered not only in the object-oriented design, but also for enterprise integration patterns, patterns of enterprise application architecture and functional programming patterns, which help to develop software in different settings.

However, a collection of tested and proven design patterns in domain modeling has not been created.

2.2 Domain modeling

Domain modeling is a way to capture our understanding of real world entities, their relationships and responsibilities that collectively describe the problem domain space. The complex process of building a domain model is one of the first stages in software development and heavily influences the degree to which an application satisfies business requirements.

Identifying domain entities and their relationships can provide an effective tool for building a common understanding across stakeholders and can help during the design of the maintainable and testable solution.

Domain modeling requires a sound knowledge of the described domain, common understanding among business experts and processes that promote that common understanding. There are supportive process guides like domain driven design, feature driven development and object oriented analysis which come with new design patterns and help to build useful models. However, a common catalog of proven design patterns for domain modeling has not yet been created.

3. PROBLEMS WITH DOMAIN MODELING

The main difficulty in creating a good domain model is due to the fact that it is required to think about software that will be built based on the model. Not only does domain modeling require domain knowledge but also understanding of software development.

Thus, domain modeling is rarely performed by a single person. Usually, domain experts and developers collaborate on that task that presents even more challenges such as lack of collaboration among them, divergent focus or misunderstanding relationships between concepts of separate domains. Thus, we chose desciptiveness and undestandability among main quality factors to tackle with proposed patterns.

Another problem that is related to the main artifact of domain modeling - domain model is keeping it up-to-date. It degrades with time as requirements change continuously. Therefore, evolvability of the model is among our main considered quality factors.

Some of the existing problems are already solved, or partially solved, by proposed and widely used process guidelines. However, there is still lack of established and documented best practices. We believe that domain modeling field can also benefit from comparison of patterns in domain modeling and in software development while software solution is based on the domain model. To address these gaps, we try to take advantage of existing design patterns in software development and apply them to domain modeling.

4. PATTERN DESCRIPTION TEMPLATE

When building a catalog of design patterns it is crucial to follow a well-defined description template. As patterns provide solutions for some repetitive problems, the description of the patterns should be unified for better understanding. It is also meant to enhance further communication between domain modeling experts and software developers by having the description template. Thus, we developed a template with 4 sections. These sections and their purpose are described below.

- Pattern name.
- Summary: In the section brief summary of the pattern and its intention is given.
- Original pattern structure: In the section a diagram of existing software development pattern is presented for reader to see the relation to its adaptation for domain modeling.
- Applicability: In this section two questions - ’why to use the pattern’ and ’how to use it’ are answered. It incorporates motivation to use the pattern and its intent. Quality factors of resulting domain model that are influenced by proposed design pattern application are also discussed in this section.
- Pattern structure: This section provides a diagram of the pattern application using the example. The visual representation is meant to enhance reader understanding of the pattern and its real use in domain model.
- Example use: This section demonstrates an example of pattern application. It has references the pattern structure to describe it better. It was decided to use e-commerce domain as an example for all the patterns. The domain is general enough to be familiar to many experts. Use of one domain for all described patterns also facilitates reader’s understanding.

The description of the patterns in section 5 will follow this template.

5. SELECTED DESIGN PATTERNS

The design patterns we selected for the catalog are well known object oriented patterns that have been applied in software engineering for more then 20 years. A lot of these patterns were described in a seminal book ”Design Patterns: Elements of Reusable Object-Oriented Software” by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Most of the patterns were selected due to the reason they arise in software solutions. Specifically, the patterns that are described in the catalog arise from the business needs. Therefore, the original software design patterns and their adaptations for domain modeling mostly have the same reason to be applied. Depicting suggested patterns in domain model will hint developers on their use in developed software. Additionally, due to paper size limitations, we focused on patterns that should be known to every computer science student from various lectures.

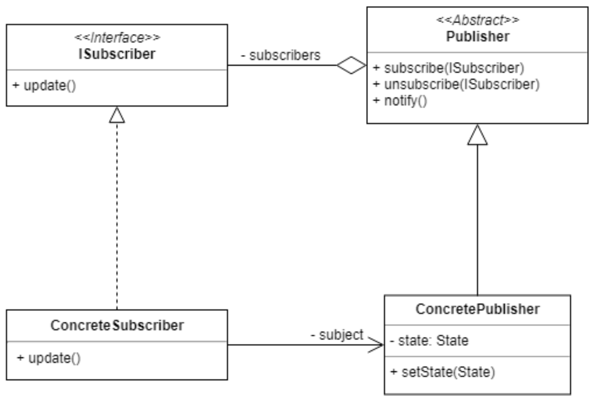


Figure 1: Publish-subscribe pattern

5.1 Publish-Subscribe Pattern

The publish-subscribe pattern, also known as the observer, is a behavioral design pattern that is commonly used in software engineering; potentially many objects publish information that is broadcasted to all subscribing objects. [3]

Figure 1 depicts the fundamental structure of the original publish-subscribe pattern: Objects can be publishers that manage a list of subscribers and notify them about changes, while subscribers subscribe to events of publishers and react to broadcasted changes. The exact behavior of both the publishers and subscribers is determined by concrete implementations. Furthermore, it is common to see implementations that diverge from the depicted structure. Sometimes a third object is introduced which acts as a message broker between publishers and subscribers, facilitating a better separation of concerns.

This pattern provides greater scalability and a more dynamic network topology but comes at an increased design complexity. The two main benefits that the publish-subscribe pattern brings to the system are loose coupling and scalability. The pattern is not restricted to be used only in OO-system but can be useful on architectural level. Loose coupling that it brings is especially helpful with the rise of microservice based architectures. Typically services depend on each other to perform a given operation using a request-response pattern that can lead to cascading failures if one service goes down. The publish-subscribe pattern can help to mitigate this problem by isolating different services into a publishing and a subscribing services for concrete events. This way the services operate independently from each other and failures do not cascade. Scalability can be improved by using the publish-subscribe pattern due to the possibility of parallel message processing. The message published by a publisher can be read and processed independently by multiple subscribers without them knowing about each other. [11]

The publish-subscribe pattern can be used in e-commerce in many use cases. One of them being the completion of the checkout process. Once a customer completes the checkout process, a number of different other actions like billing and shipping have to be triggered. Without the publish-subscribe pattern it would be the checkout process' responsibility to trigger all of the following processes, which becomes inconvenient as the number of subsequent processes

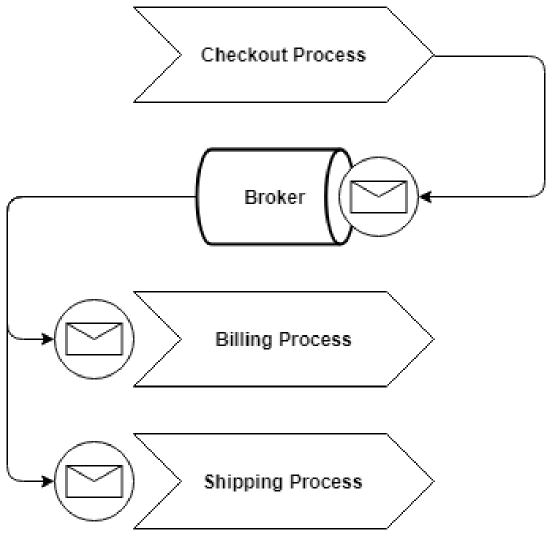


Figure 2: Practical example of publish-subscribe pattern

increases. Using the publish-subscribe pattern we end up with the following model, which scales really well while the number of subsequent processes increases.

Figure 2 shows how the problem could be solved using the publish-subscribe pattern. The checkout process is a publisher that once completed publishes a message like "Checkout Completed" to a message broker. The message broker then forwards the message, which triggers both the billing and shipping process. If we want to add another process following the checkout process we can trigger it based on the "Checkout Completed" message that is broadcasted by the broker leaving the checkout process untouched. By using the publish-subscribe pattern we ended up with a domain model that is open for modification and more robust against changes.

5.2 Composite Pattern

The composite pattern is a structural design pattern, which composes zero-or-more similar objects so that they can be manipulated as one object. This is done by composing objects into tree structures that represent whole-part hierarchies. [3]

Figure 3 depicts the fundamental structure of the composite pattern: The modeled component has functionality, in this case the method "operation()", which is present on all objects that can be part of the group of objects that the pattern describes. If the component is not split into parts we call it a leaf. If it is broken down further, we call it a composite. [6] The pattern simplifies the work with tree-structured data by removing the need to distinguish between leaf-nodes and a branch, which makes the model less complex and therefore less error-prone. [7]

In the e-commerce domain the composite pattern could be used to describe a hierarchy of categories, where each category could have zero-or-more sub-categories. Without using the composite pattern our domain model would include a lot of edge cases and would be tightly coupled to

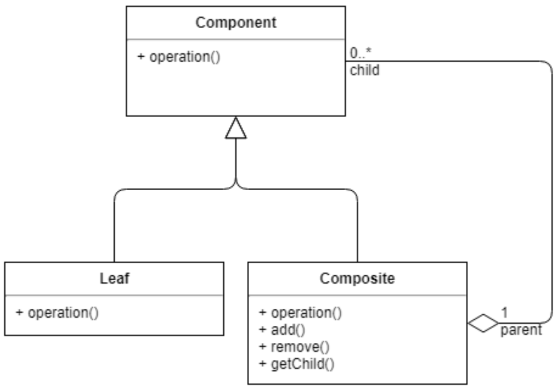


Figure 3: Composite pattern

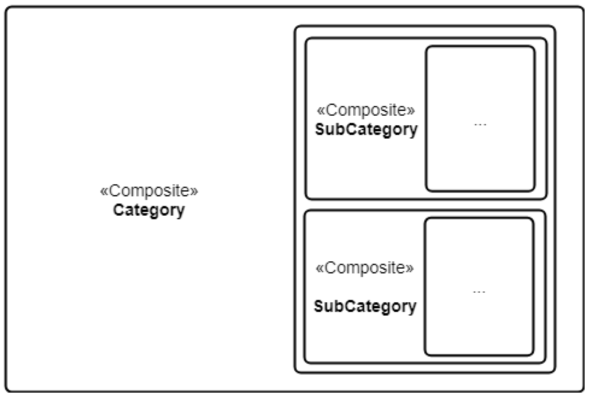


Figure 4: Practical example of composite pattern

the current number and types of categories, making it hard to evolve as the existing categories change. But using the composite pattern we end up with the following model, that will be easy to evolve as the existing categories change.

Figure 4 shows how the problem could be solved using the composite pattern. Each category is modeled as a composite with a certain set of functionality and sub-categories that are composites themselves. By using the composite pattern we ended up with a domain model where the existing categories can easily be changed without affecting the overall structure of the solution.

5.3 Mediator Pattern

The mediator pattern is a behavioral pattern which aims to reduce the coupling between a set of interacting objects and makes it possible to change the interaction between these objects by introducing a mediator which encapsulates the communication between objects. [3]

Figure 5 depicts the fundamental structure of the mediator pattern: A mediator manages a number of colleagues and handles new colleagues that register with the mediator. Instead of sending messages directly to one of the mediator's colleagues messages are distributed by the mediator decoupling the sender of the message from its recipient. [10]

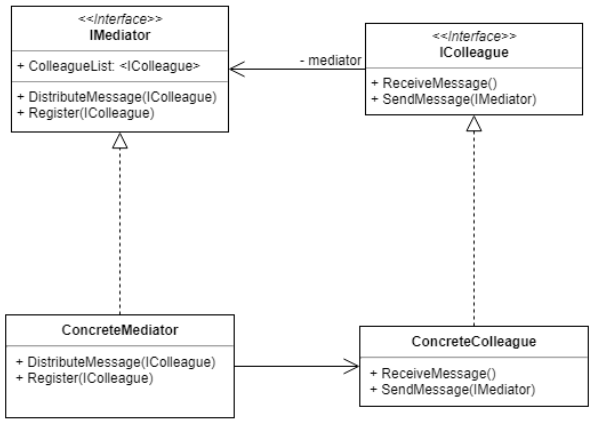


Figure 5: Mediator pattern

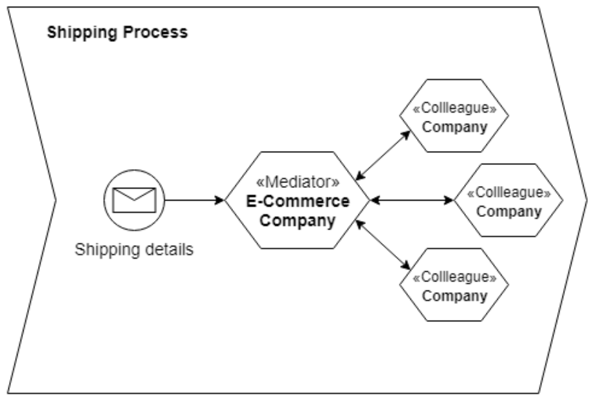


Figure 6: Practical example of mediator pattern

This pattern makes it easier to build loosely coupled applications by encapsulating communication between objects. This prevents us from building applications where objects directly depend on other objects that reduces the overall system's modifyability and testability. The resulting model is more descriptive and hints an appropriate solution for the business problem to developers [9]

In the e-commerce domain the mediator pattern could be used to model the shipping process, where the delivery of ordered goods has to be arranged. If we did not use a mediator in our domain model we would end up with a complex solution where a given customer has to talk to multiple companies until his ordered goods arrive. A lot of things could break in this model when processes at other companies change. Using the mediator pattern we end up with the following model that encapsulates communication between different companies.

Figure 6 shows how the shipping process could be modeled using the mediator pattern. All the customer has to do in this model is to provide the required shipping details, for example, address. From this point forward the e-commerce company acts as the mediator between different companies that are involved in the delivery process of

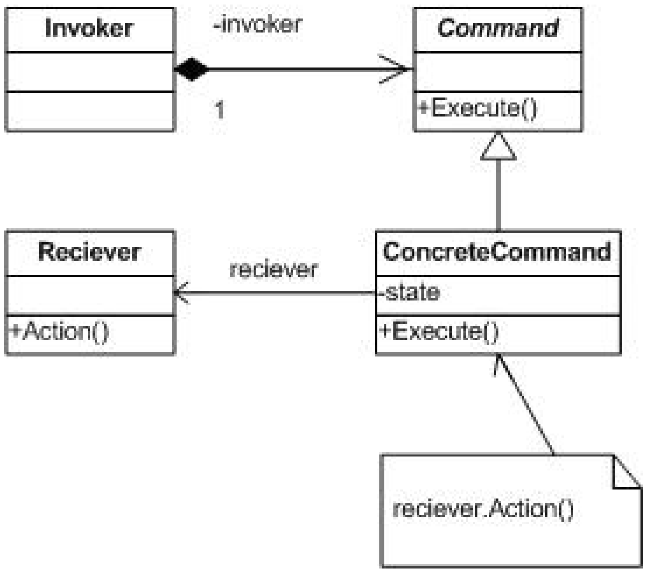


Figure 7: Command pattern

the ordered goods. If the actual delivery process changes only the e-commerce company is affected. The customer will not notice anything because the communication is encapsulated. By using the mediator pattern we ended up with a domain model that clearly separates different concerns and allows future changes without affecting the customers end-experience.

5.4 Command Pattern

The Command pattern is a behavioral pattern that is used with intent to encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations. [3] The diagram of the original pattern is depicted on figure 7.

In domain modeling, the pattern can be used to represent complex domain entities that consist of a set of instructions and require additional but closely related data as one separate encapsulated object.

It helps to make the model clearer, as only one entity will be demonstrated. The detailed view of the entity with instructions, their order specification and command data can be shown separately. It also improves understandability and descriptiveness of the model. Besides simplification of the model representation, the pattern allows experts to keep the elements of the command entity together during the modeling not spreading them across the system. It facilitates the quality of the future system design. Proposed pattern also eases communication among domain experts, as they will address the closely related set of instructions and their data as one entity.

A product gift wrapping can be an example of the pattern application in e-commerce domain (figure 8). The is a precise procedure to follow while choosing an appropriate gift box for a product ordered. Besides this procedure there is wrapping. The algorithm that consists of several steps for box selection and wrapping is our set of instructions that we will encapsulate. Information about the order of instructions execution is added to the detailed view. Some of the steps require additional information. Information about the boxes may be external to the modeled entity but the wrapping process may require gift-mode specific information that can be encapsulated together with above mentioned instructions.

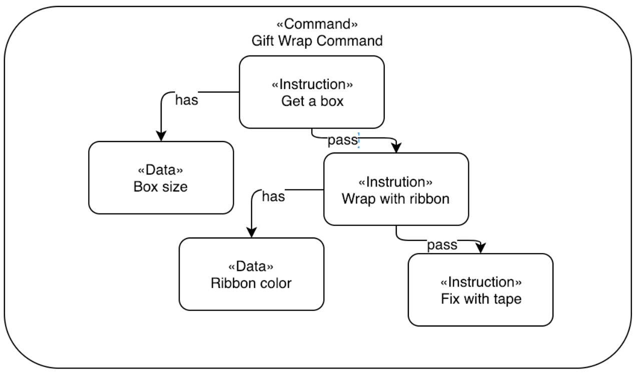


Figure 8: Practical example of command pattern

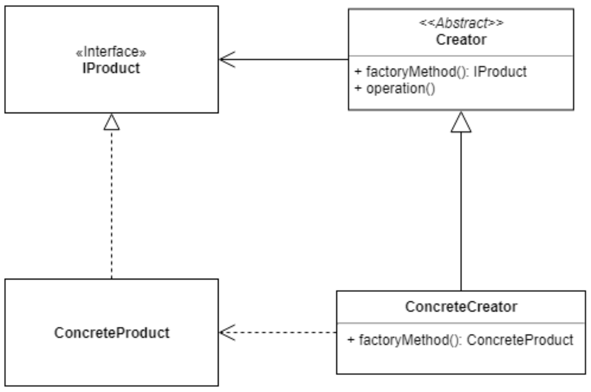


Figure 9: Factory method pattern

In the demonstrated example there are three instructions for the command. They pass control to each other after execution. Two of them, namely 'Get a box' and 'Wrap with ribbon' that require additional data - box size and ribbon color respectively. Without the pattern application, the instructions and the data they need would be sprawled in the model or, at least, not organized well. The order of the instructions would not be determined in the model, therefore additional document would be required for this that brings inconvenience.

5.5 Factory method Pattern

The factory pattern is a creational design pattern which uses factory methods instead of constructors to create new objects. [3]

Figure 9 depicts the fundamental structure of the factory method pattern: Products are constructed by a creator using a factory method, which returns the product. In case when we have different products that have to be created we can further specify the creator. The concrete creator's factory method then return concrete products. [2]

In most non-trivial cases the creation of an object requires

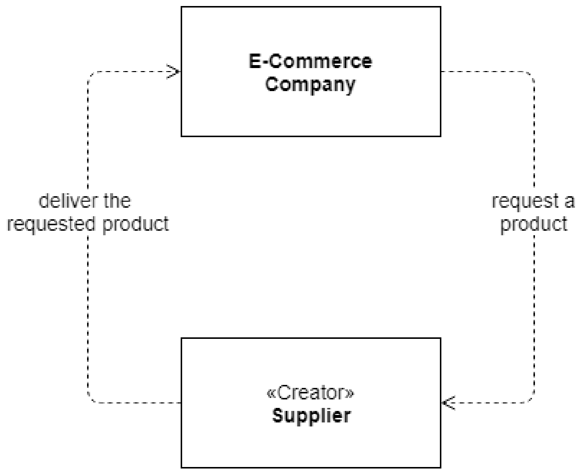


Figure 10: Practical example of factory method pattern

complex processes, which do not naturally fit into the composing object. This may lead to duplication of code, not to provide a sufficient level of abstraction or couple tightly the objects of the application together. By using the factory method pattern the composing class can defer instantiation to another class, called a creator. That way we have a clear separation of concerns and can change objects without affecting the whole application. [8] In the e-commerce domain sold products are usually not manufactured by the e-commerce company itself but ordered by one of its suppliers, which can be modeled using the factory method pattern.

Figure 10 shows how this situation could be modeled using the factory method pattern, which is a good fit as the e-commerce company does not need to know how the actual products are manufactured. By using the factory method pattern we achieved a clear separation of concerns, which allows us to model the e-commerce company's domain independently of the supplier's domain as long as both domains agreed upon a communication standard between them. It increases descriptiveness and understandability of the model.

5.6 Prototype Pattern

The prototype pattern is a creational design pattern which specifies the type of objects to create using a prototypical instance. Instead of calling the 'new' operator to create an instance of a class, the client calls the 'clone()' method on the prototype. [3]

Figure 11 depicts the fundamental structure of the prototype pattern: A client performs an operation, but instead of creating objects using the 'new' operator, depends on a prototype that can be cloned. The exact implementation of cloning an object can differ. [2]

This pattern can be used to avoid two potential problems; The existence of subclasses of an object creator that can exist when using the factory method pattern and the cost of creating new objects in a standard way. Avoiding subclasses can help to make a solution more concise and therefore avoid potential errors.

In e-commerce we can consider objects that are reasonable

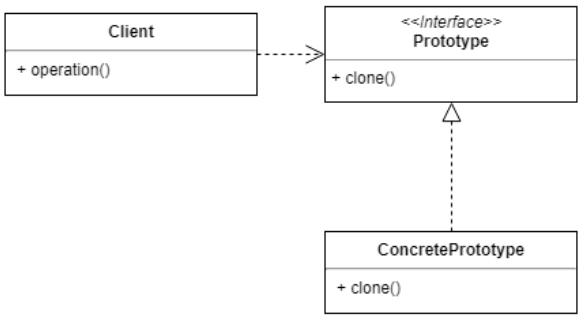


Figure 11: Prototype pattern

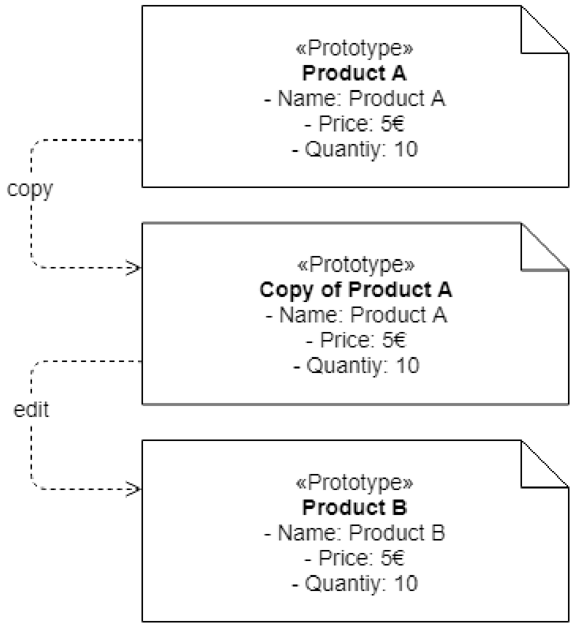


Figure 12: Practical example of prototype pattern

to clone instead of creating from scratch, because creating them every time from scratch would be too expensive. For example, addition of a new product that is similar to a product that is already offered in a shop can be unnecessary in our case. If we were not allowed to use the prototype pattern in our domain model, we could not take advantage of information that is already embedded in existing products. Instead we would have to add the product from scratch, which would require significantly more time. The pattern improves evolvability of the resulting model. Using the prototype pattern we end up with a domain model that allows us to take advantage of products that were previously added and save time later on.

Figure 12 shows how the addition of a new product can be done using the prototype pattern. Instead of adding "Product B" from scratch we can take advantage of its similarity to "Product A" and copy "Product A" and change only the attributes that differ. Depending on the number of attributes

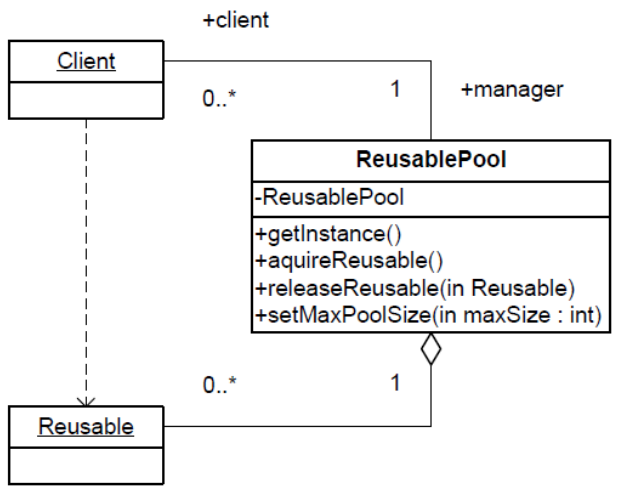


Figure 13: Object pool pattern

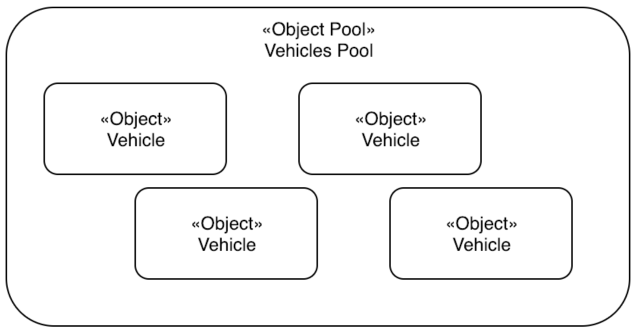


Figure 14: Practical example of object pool pattern

that are similar between products that process could save a significant amount of time.

5.7 Object Pool Pattern

The Object Pool Pattern is a creational design pattern. Its main intention is to reuse and share objects that are expensive to create. [4] Its original diagram is shown on figure 13.

The pattern application in domain modeling is meant to incorporate a set of objects that should be shared among other entities in the model. Only entities of the same type can form a pool so they are not differentiated by user.

The pattern contributes expressiveness property of the domain model. Using the pattern the resulting model will communicate a property of objects organized in the pool of being sharable. It also promotes an idea of sharing these objects. These two aspects adding descriptiveness to the domain model might help software developers in making decisions about design of the system.

There may be different reasons to share objects among entities instead of creating new ones. One example from e-commerce domain is the transportation of the products (figure 14). When goods are ordered by people, they are to be delivered with van or any other transport. The vehicles are located in a garage that is our object pool. Each vehicle is one shareable object. One vehicle is used to deliver multiple products.

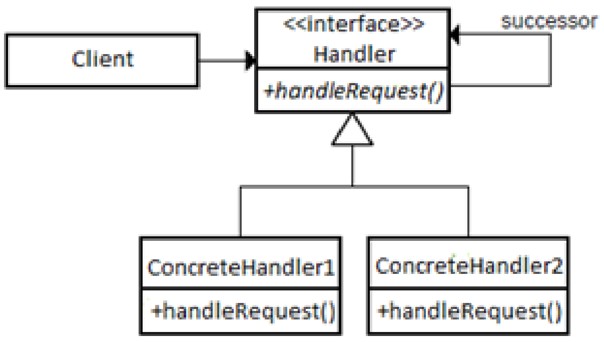


Figure 15: Chain of responsibility pattern

In our example, having several same vehicles in the pool hints that any of them can be chosen interchangeably. For this reason, we intentionally not presenting any information that could help user to distinguish between them. If we did not do so, developers might integrate support of different vehicles in the system following the diagram while these different types are not required. Moreover, by specification one vehicle can deliver many products. If domain model does not highlight this idea, developers could design a solution the way that it will create a new object for each product. Such dissension with real-world can lead to problems in the designed system.

5.8 Chain of Responsibility Pattern

The main intention of this pattern is to avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it. [3] The basic structure is demonstrated on figure 15.

The purpose of the pattern adaptation for domain modeling is to organize command entities discussed in Command pattern section. When the commands are forming a higher level algorithm of execution, it happens that some of the commands have the same general responsibilities but exactly one should be chosen due to some additional criteria. The chain of responsibility pattern in domain modeling helps to organize these criteria connected to the commands in the execution flow. The resulting chain can be considered as an meta command with instructions regulating execution flow as well as choosing appropriate commands at each step.

Domain model benefits from the pattern application in several aspects. The main reason to use the pattern is to document knowledge about responsibility of each command in the process that construct a chain. Designers of the system will be able to separate responsibilities better following the model. Another motivation to use the pattern is to extract information about order of execution from commands themselves. It will make it easier to understand the command entities from the model and increase descriptiveness of the domain model. As the commands may be reusable out of the chain context, it will also help designers to separate chain-related information from commands themselves

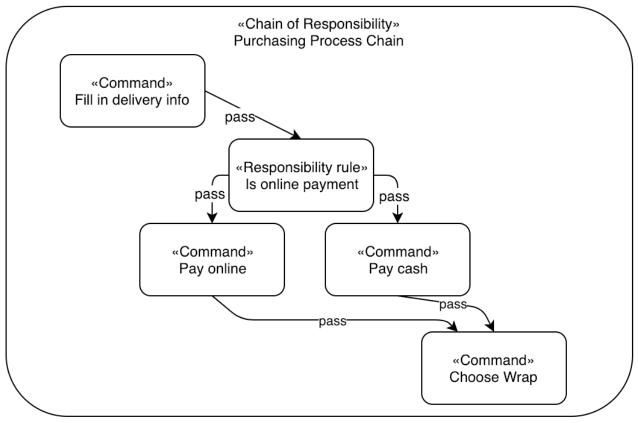


Figure 16: Practical example of chain of responsibility pattern

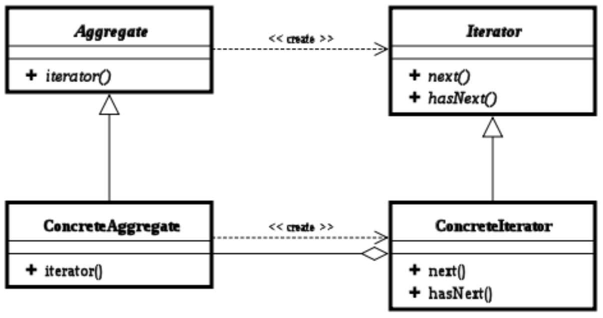


Figure 17: Iterator pattern

during the design and development phases.

One example from e-commerce of the pattern application is the purchasing process (figure 16). Each step can be designed as a separate command, such as filling delivery information containing several simple steps. The payment step, however, often offers different options - payment methods. Each of the method can be represented as a command. These commands are interchangeable in context of the whole purchase process (chain). The information about choosing appropriate command is stored in the chain.

If domain model did not combine the commands in a chain with demonstrating their order as well as the interchangeable parts, all this information would have to be written in specification documents. The visual representation gives better understanding and is easier to memorize.

5.9 Iterator Pattern

The iterator pattern helps to provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation. [3] The basic structure in UML is depicted on figure 17.

In domain modeling the pattern can help to encapsulate information about ordering in the execution of steps in some process. It contains all the information necessary about iterating through a set of steps to achieve intended results.

One application of the pattern come from the idea that in some cases it is beneficial to separate steps and information

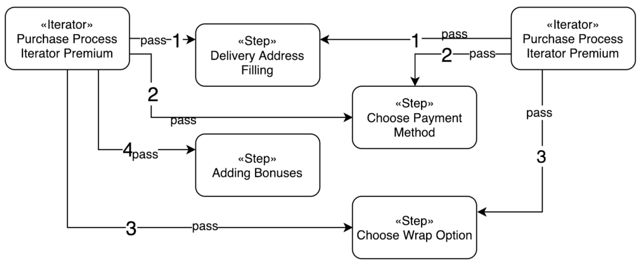


Figure 18: Practical example of iterator pattern

about their order of execution. It also often the case that the same set of steps executed in different order can lead to different results. Applying the iterator pattern, we will need only one set of steps and an entity representing the rules to iterate them.

The pattern increases generality of the domain model that improves its evolvability. Model that is more general is easier to extend in the future. To have new instructions that share the same set of steps previously defined for other purposes, only one more entity will have to be added without any other changes to the model.

The pattern also makes the resulting domain model easier to understand. Looking at many entities that have much in common but lead to different results is difficult. The pattern helps to keep commonalities together.

In e-commerce domain there is a possibility to combine command and iterator design patterns for purchasing process (figure 18). Steps in the purchasing can be executed in different orders. Questions about address information can precede questions about gift packaging or other way around. The order of the commands can be encapsulated in an iterator entity to replace it later or use different order for different clients (as in our case, for 'premium' and 'basic' client subscriptions). On the system design stage developers will treat this iterator as a separate entity that will help them to make it configurable for each user if required.

The domain model without iterator pattern would need additional specification document that would encapsulate rules on instructions order. Disconnecting the visual representation of instruction entities and the rules on their execution brings much cognitive load.

5.10 Decorator Pattern

The decorator pattern allows to attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality. [3] Its formal structure is demonstrated on figure 19.

Decorator pattern in domain modeling helps to label entities that are supplementary to base entities. Instead of representing several variations of an entity it is suggested to have one entity with many decorators. The decorators can be added to the model as separate entities if required. Decorated entity is located inside of all its decorators in the model.

Application of the pattern makes the domain model more concise that improves the understandability quality factor. The multiple variations of an entity that can be represented as different ones spoil domain model. The suggested solution leaves only one entity that can have multiple decorators.

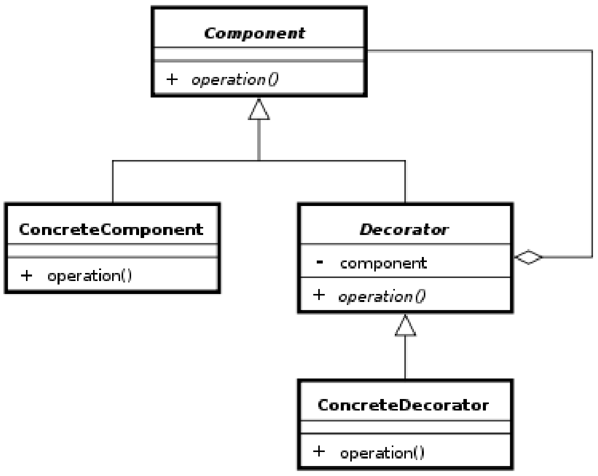


Figure 19: Decorator pattern

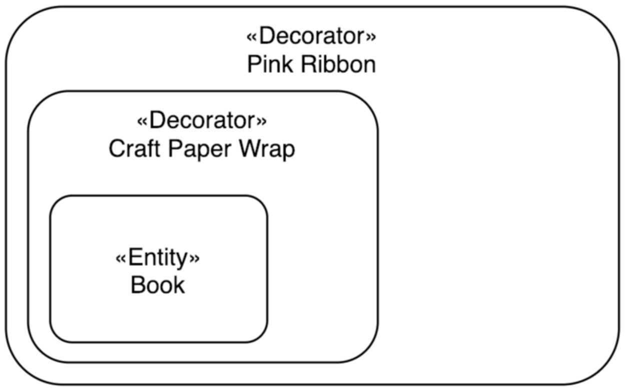


Figure 20: Practical example of decorator pattern

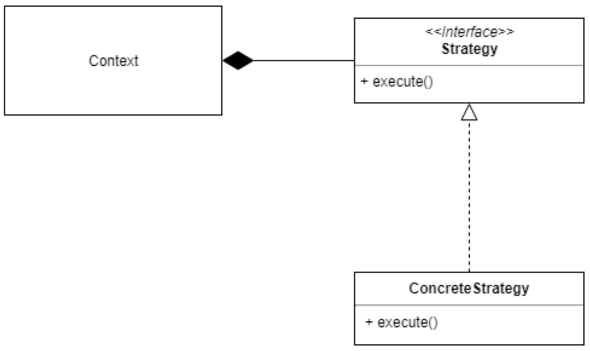


Figure 21: Strategy pattern

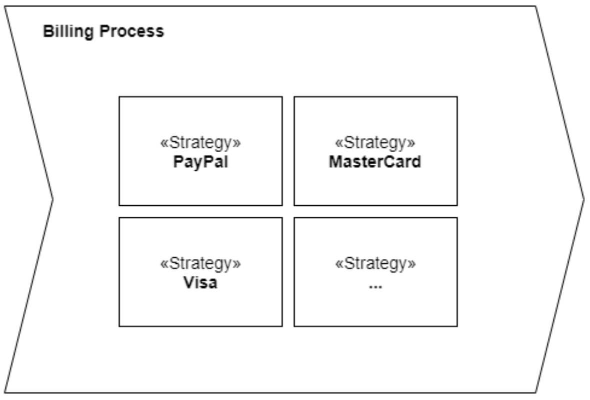


Figure 22: Practical example of strategy pattern

The pattern helps to communicate the intention of decorators meaning increased descriptiveness that can help on the software design phase. Developers will know that there is only one entity while other objects that bring variations simply decorate this entity. In case of having one entity for each variation on the model, it would be more difficult to recognize. The similar code that is required to be written to represent the entity would be spread among several objects. Such system is more difficult to maintain.

An example from e-commerce is a gift wrapping options for a purchased product (figure 20). Our entity would be the product and the wrapping is a decorator. As there are several different wraps for a product, there would be one decorator for each of them, still having only one entity, namely the product, that we decorate.

Disconnecting an entity and its decorators worsens the developers' understanding of decorators main responsibility. It also makes it difficult to learn all the possible options that entity can have without having decorators close to it in the model.

5.11 Strategy Pattern

The strategy pattern, also known as the policy pattern, is a behavioral design pattern which allows selecting an algorithm at runtime instead of making the algorithm statically. This way the algorithm can be specified on a per-client basis. [2]

Figure 21 depicts the fundamental structure of the strategy pattern: In a given context we rely on an abstract notion of a strategy that can be executed and is implemented by multiple concrete strategies. We can exchange the concrete at any point in time without breaking objects that rely on the strategy. This way we gain a lot of flexibility and can extend our application as the number of strategies grows.

In many there are different algorithms, which solve the same problem that make the algorithms interchangeable. Instead of deciding on a single algorithm we can define a family of algorithms, called a strategy, that encapsulates each specific algorithm. [3] In e-commerce we can look at the billing process, which shows potential to be modeled using the strategy pattern.

Application of the pattern in domain modeling make the model more descriptive helping developers to make the right decisions on development phases. Moreover, using the pattern we increase model's evolvability not requiring much changes when requirements change and more different algorithms arise.

Figure 22 shows a potential application of the strategy pattern to our domain. The billing process does not rely on a concrete billing implementation. Instead, concrete billing implementations (like PayPal, MasterCard or Visa) are treated as strategies with the same outcome of billing the customer. If we want to allow customers to pay with another provider later on, we can simply add this provider as a new strategy without affecting existing strategies.

6. EVALUATION

In the previous section we described well known object oriented patterns and looked at their applicability to domain modeling providing the adapted version of the pattern using the e-commerce domain as an example. We have shown by example that every pattern we looked at is applicable to the e-commerce domain and that the resulting domain model benefits from their application. For this reason, we discussed the quality factors affected by the patterns and made a comparison of model with and without the suggested pattern to demonstrate how exactly the quality is increasing.

However, the patterns we looked at are only representative for a small number of existing design patterns, as all of them focus on static properties of a model. There are other design patterns that focus on processes or organizational aspects. While we demonstrated that the selected object oriented design patterns were applicable in domain modeling, it is not clear whether the result would hold as we evaluate process, organizational or other object oriented design patterns that do not focus on the static properties of a model. Therefore, they require further investigation and assessment of their applicability in domain modeling.

7. RELATED WORK

There is not much work done on the topic so far. This research direction in domain modeling field is still unexplored. However, there are some articles on the internet as well as master thesises. The most remarkable book on the topic is Patterns, Principles, and Practices of Domain-Driven Design [5]. In the book author describes several useful patterns and discusses their applicability. The patterns follow best practices in the domain modeling field that are collected in the book, too. Many of the patterns are based on the existing ones, collected by Eric Evans and described in book "Domain-Driven Design" [1]. However, none of the found researches had the same intention of exploring applicability of software engineering design pattern in domain modeling, as in this paper.

8. CONCLUSION & FUTURE WORK

This paper set out to assess the applicability of existing design patterns to domain modeling. After looking at design patterns, domain modeling and the problems with domain modeling in general, we described a total of eleven well known object oriented design patterns and showed how their adaptations could be applied to modeling of the e-commerce domain cases.

We found that all of them are applicable and provide benefits to the resulting domain model. However, as mentioned earlier, the selected design patterns focused only on static properties of a solution, so the question whether design patterns in general are applicable to domain modeling is still unanswered. Therefore, whether design patterns that do not focus on static properties, like process or organizational design patterns are still applicable to domain modeling will be the big question of future work. The patterns we described can be seen as a first step towards a catalogue of design patterns for domain modeling that can be expanded with further design patterns after their assessment.

9. REFERENCES

[1] E. Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley, 2004.

[2] E. K. S. B. B. H. M. L. M. e. Freeman, Eric; Freeman. *Head First Design Patterns.* O'REILLY: 162., 2004.

[3] R. J. R. V. J. Gamma, Erich; Helm. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley Publishing Company, 1994.

[4] https://www.oodesign.com/. Object pool. `https://www.oodesign.com/object-pool-pattern.html`. Retrieved December 2, 2018.

[5] S. Millett. *Patterns, Principles and Practices of Domain-Driven Design.* Wiley, 2015.

[6] w3sDesign.com. "the composite design pattern - implementation". `http://w3sdesign.com/?gr=s03&ugr=implem`. Retrieved November 28, 2018.

[7] w3sDesign.com. "the composite design pattern - problem, solution, and applicability". `http://w3sdesign.com/?gr=s03&ugr=proble`. Retrieved November 28, 2018.

[8] w3sDesign.com. "the factory method design pattern - problem, solution, and applicability". `http://w3sdesign.com/?gr=c03&ugr=proble`. Retrieved December 3, 2018.

[9] w3sDesign.com. "the mediator design pattern - problem, solution, and applicability". `http://w3sdesign.com/?gr=b05&ugr=proble`. Retrieved December 2, 2018.

[10] w3sDesign.com. "the mediator design pattern - structure and collaboration". `http://w3sdesign.com/?gr=b05&ugr=struct`. Retrieved December 2, 2018.

[11] w3sDesign.com. "the observer design pattern - problem, solution, and applicability". `http://w3sdesign.com/?gr=b07&ugr=proble`. Retrieved November 25, 2018.

Optimizing Enterprise Architectures Considering Different Budgets

Niklas Dohmen
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
niklas.dohmen@rwth-aachen.de

Kevin Koopmann
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
kevin.koopmann@rwth-aachen.de

ABSTRACT

Enterprise Architectures (EA) are used to define the structure and operation of an organization and commonly find usage in the realization and modification of IT business strategies. We propose a technique to optimize the costs incurred between two layers of the EA, especially considering differing departmental budgets. This is achieved through consideration of a flow problem aiming to optimize a graph consisting of budget-, upper layer-, capability- and lower layer nodes allowing budgets to be used by different departments. Additionally, we implement techniques previously published to allow operational and transitioning costs to be taken into consideration, in an effort to better reflect the organizational problems found in reality.

Categories and Subject Descriptors

Applied Computing [**Enterprise Computing**]: Enterprise Architecture modeling—*Optimizing*

Keywords

Enterprise Architecture, Enterprise Architecture Management, Linear integer programming, Optimization, Department budgets, Minimum-cost flow problem

1. INTRODUCTION

Mostly all IT-Projects in large companies realize technical requirements which are requested by different business branches. To adjust IT-Projects to the overall strategy of the company it is essential to develop and manage enterprise architectures [1]. The most common definition of the term Architecture is found in the ISO norm ISO/IEC/IEEE 42010:2011. There it is described as the "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". Therefore an EA is a holistic view over all underlying structures, elements and their relations. It gives large enterprises a centralized framework, to consolidate their strategic plans and business mission. According to IT, it includes all IT-relevant components in the business, like information resource management, life cycle planning, systems or software reengineering. Furthermore, it is considered to be the backbone for coordination of actual and further development of the business IT systems and data. [9]

A real-life example where an EA needs to be used is the following: The Dutch government did a massive redesign of their entire chain of organizations involved in their social security system. In this context, the collection of employees' social security premiums is transferred from the central social security organization to the tax administration. For sure, this sounds logical, since the collection of taxes is pretty similar to collecting social security premiums. This seemingly simple change needs a major redesign of organizational structures, business processes, IT applications, and technical infrastructure. Massive data flows need to be redirected within the different organizations.[7]

As in the revolution of information technology (especially in businesses), IT landscapes like the above described, become increasingly larger (Vodanovich et al. [11]). As a result of that, the complexity of IT-systems is continuously growing, so that manual maintenance is unfeasible. Nevertheless, to keep systems preferably facile, it is a common practice trying to reduce the complexity of such systems e.g. eliminating elements to reduce complexity. There are several options on which parameters a system should be optimized e.g. minimal quantity of elements or minimum operating costs. Aspects which are important for an optimal solution are depending on the usage, but often a hybrid solution is necessary. In this paper, we integrate the approach from Hacks and Lichter [5], in which they find an optimal solution considering transition costs for adding and removing applications to be economically reasonable. In larger businesses departments commonly have department budgets to get an easier financial overview for managers. Additionally, budgets are often shared by different departments or one department is using more than one budget. This will be the case if e.g. two departments benefit from the same software in the EA. To achieve a more realistic optimization model we integrate such department budgets with this approach.

As the goal of this paper is to improve modelling of EAs in an optimization sense we will not focus on suggestions for management decisions that could be derived from this data. We will represent EAs in the form of graphs, where elements are detailed as nodes and relations as edges. Interpretation of what constitutes elements and edges will be specific to the particular usecase.

In section 2 of our paper we introduce related work according to mathematically described EAs and their previous optimization models. Here we focus on two previous approaches on which this paper is based. The formal definition of EAs budgets and our approach to optimizing considering different budgets is introduced in section 3. Considering this formal definition we also developed a technological implementation which is described in section 4. To evaluate and prove our findings we give an insight on how our approach is scaling in reality according to our implementation which we will introduce in section 5. In section 6 we summarize our results and give a short perspective on further regarding topics.

2. RELATED WORK

Different work has been done to describe and optimize EAs in a mathematical programming formulation fitted to their field of use. As mentioned above, the most important related work, on which this paper is mainly based on, is worked-out from Hacks and Lichter [5]. Here the authors optimize relations between two adjacent layers of EAs considering operational costs and transition costs to change systems from the as-is state to the optimal state. In this graph, optimization constraints are interpreted between two layers as triangles. These triangles consist of the connection between a needed capability of an upper layer element which is realized by a lower element. The optimization model is then solved using a linear integer program which is even appropriable for realistic scenarios. In our approach, we extend this optimization problem from Hacks and Lichter regarding department budgets which also can be not mutually exclusive.

A similar approach from Giakoumakis et al. focused on replacing existing services with new services without disrupting the enterprise which they call the "information system architecture evolution management problem" [3]. They formalized this problem into a graph and solved it using the multi-objective optimization problem which considers departments, existing services, new services and IT-components. To apply changes in the EA they link services and modules, department and services and attach transition costs for changes. In this paper, old services are replaced by new services one-to-one, which is not a common practice. Although they considered budgets to fund the changes, each department has only one budget, so that budgets are mutually exclusive.

Another optimization approach is made by Franke et al. [2] where they use a binary integer programming model to find an optimal mapping between IT systems and processes based on needed functionalities. These functionalities are connected with processes they use and with certain fulfilling IT systems. In a next step, connections for the as-is state will be made through connecting processes and IT systems directly, which has to be optimized by applying change costs and operation costs. Therefore redundant connections will be dissolved.

One fundamental different approach concerning transition costs is made from Lagerstrom et al. [6] by using a probabilistic relation model with a meta-model based on literature research and evaluated through interviews and workshops. This is used to supply transitions costs estimation for large software projects. By providing a prediction and not optimizing the as-is state this approach differs from our formulation.

3. MODELLING A BUDGETED ENTERPRISE ARCHITECTURE OPTIMIZATION

Before presenting our approach we first give a quick overview in subsection 3.1 of how EAs are modeled in literature. Later on, we especially take care of the foundations which are already acquired by Hacks and Lichter. Based on these foundations we provide our optimizing approach.

3.1 Towards an Optimization Model

According to Winter and Fisher [12] enterprise architectures are most commonly composed of hierarchical layers. Each of these layers consists of various architectural artifacts which can be connected through different relations with elements of the same layer. Additionally, artifacts on a layer can be explicitly influenced by elements on the superseding layer, reflecting a priority of importance of decisions undertaken on higher levels of the architecture. [8]

For instance, an enterprise architecture could consist of business entities such as "Human Resources", "Enterprise Resource Planning" and "Liquidity Management", each requiring a certain set of capabilities. Such capabilities could be functions such as "Hiring" or "Stock Trading". This is modeled by directed edges from the business entities to the capability nodes. Furthermore, applications selectively implement certain capabilities, represented through edges from the capabilities to implementing applications. Figure 1 shows such an exemplary architecture where the layers are named as *Cost Centers* and *Applications*.

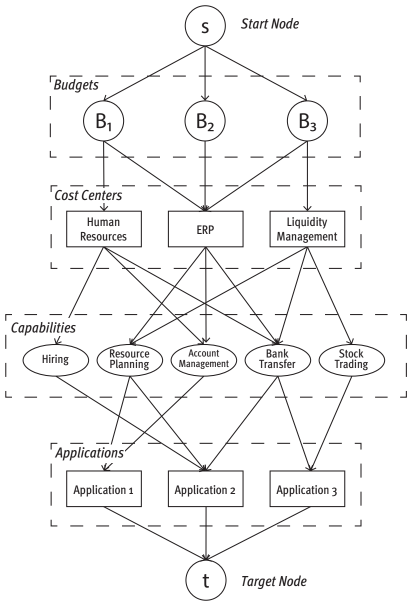


Figure 1: Concrete example for two layers in an enterprise architecture (reproduced from Hacks and Lichter [5])

An example of an application of this concept could be the simplification of an IT business infrastructure in order to reduce costs. A business resource may require certain capabilities (most times functionalities) fulfilled by applications on the application layer. An optimization model, in this case, would be suitable to reduce the operational cost of the architecture by selecting a set of applications that implements all required capabilities at the smallest possible cost to the enterprise.

In order to be able to apply common graph algorithms to the solution of problems related to the enterprise architecture, we will model these architectures as directed graphs, as suggested by Hacks and Lichter [5]. An element of the architecture describing an aggregation hierarchy will be represented by vertices, and the edges between these vertices mirror dependencies between elements on different layers of the architecture.

In our subsequent analysis of the optimization problem for enterprise architectures we will consider only the interactions between two directly adjacent layers of an overall architecture. This is permissible because decisions on higher layers of the architecture reduce the degree of freedom on subsequent layers [8], and greatly simplify understanding of the optimization process. To perform an analysis of the entire architecture, simply repeat the analysis in top-down for each interface between two layers.

Hacks and Lichter [4] present different objectives that could be used to optimize the model, such as minimal coupling, amount of lower layer elements or operational costs. In real-world applications, we often find certain business entities, such as departments, to be restricted to a specific budget. These often yearly scheduled budgets are important for business managers to estimate the project and department costs. It is common that various departments are sharing the same IT infrastructure so that each department has its own scot, which needs a specific budget. We will extend the model by considering these budget restrictions, while also incorporating transitional costs for the introduction or decommissioning of currently active applications in the architecture. This seems to be warranted as moving away from old applications or implementing new ones often arises substantial costs necessary for example to educate users, transfer data or resolving other dependencies.

3.2 Foundations

Building on this initial understanding of enterprise architecture modeling and objectives in its optimization we will now introduce a more formal representation of these architectures.

As described by Hacks and Lichter [4] we want to represent the enterprise architecture as a quadruple $EA = (\mathcal{L}, \mathcal{C}, E, R)$ where $\mathcal{L}$ is an ordered set of architectural layers, $\mathcal{C}$ describes a set of sets of capabilities, E is a set of architectural elements, and R represents a set of relations between these elements and capabilities.

We assume each layer $L \in \mathcal{L}$ to consist of architectural elements so that $L \subset E$, and to be disjunct as prescribed by eq. 1.

$$L_i \cap L_j = \emptyset \qquad \forall L_i, L_j \in \mathcal{L} \quad i \neq j \tag{1}$$

For each layer element $ul_i \in L_j$ of the upper layer there may be an associated budget $b_i \in \mathbb{N}_0$. For each element on the lower layer $ll_i \in L_{j+1}$ there is an associated operational cost p_i and a transitional cost tp_i.

Furthermore we place capabilities on interfaces between two adjacent layers. For each such interface between L_i and L_{i+1} we assign the symbol $\mathcal{C}_i$. As each capability is associated with a specific business requirement we assume all capabilities to be unique to an interface between two adjacent architectural layers.

$$\mathcal{C}_i \cap \mathcal{C}_j = \emptyset \qquad \forall \mathcal{C}_i, \mathcal{C}_j \in \mathcal{C} \quad i \neq j \tag{2}$$

The set of relations consists of tuples of architectural elements and capabilities, which couples upper layer elements with capabilities and capabilities with lower layer elements.

$$R \subset \{(e, c) : e \in L_i, c \in \mathcal{C}_i\} \cup \{(c, e) : c \in \mathcal{C}_i, e \in L_{i+1}\} \tag{3}$$

This object EA is the subject of optimization.

3.3 Modeling Cash Flows in Enterprise Architectures

Hacks and Lichter [4] present a solution to the optimization problem for enterprise architectures by introducing intermediate relations between all elements of an adjacent layer constituting a bipartite graph. In order to be able to incorporate budget constraints into our model in a meaningful manner and benefit from graph algorithms we chose a different approach by modeling cost flows originating from budget nodes.

In order to obtain the desired solution we apply a modified maximum flow problem to a graph we construct from input parameters of the problem. The maximum flow is concerned with obtaining a maximum flow through a single-source single-sink flow network by assigning a feasible flow $f(e) \in \mathbb{R}_+$ to all edges $e \in \mathcal{E}$, such that the total flow $\sum F_{i,t}, (i,t) \in \mathcal{E}$ into the sink is maximal. Additionally, the maximum flow problem imposes an additional constraint by limiting the flow across an edge to a certain capacity limit c. Naturally, in a flow network, the flow out of a node must be equivalent to the flow entering the node, excepting source and sink nodes. Instead of considering a maximum flow across the network, we want to minimize cost flow, and we limit the flow across certain budget-related arcs to a specific budget value. In linear programming terms, we formulate this objective in eq. 4, and formulate these initial constraints in 5, and 10. [10]

We begin to construct a graph with vertices $\mathcal{V}$ consisting of elements from two adjacent layers L_i and L_{i+1}, as well as capabilities $\mathcal{C}_i$. We also start with a set of edges $\mathcal{E}$ equivalent to the set of relations R for these adjacent layers.

We also add a start node s as an entry point for a flow algorithm. Furthermore, for each budget b_i it is necessary to construct a node and add relations to all upper layer elements associated with this budget. For additional relations, we introduce for each budget between s and the budget node we then apply constraints to these relations to ensure a maximum cost flow equivalent to the budget.

From each of the lower layer elements, we construct additional relations to the sink node t. The activation of a lower layer element bears operational costs and may additionally arise transitional costs. The deactivation of a lower layer application may also arise transitional costs, but no operational costs. To incorporate this requirement into our cash flow model, we must also enforce an additional constraint to force the flow on these edges to be equivalent to exactly the cost arised by (de-)activation of the element.

This model represents the flow of monetary resources originating from operational budgets through the architecture. To optimize towards minimal costs, we can now simply attempt to minimize the flow of cash through the network.

3.4 Final Modelling

To summarize our findings we will now combine the previous results to formulate the following selection problem for two adjacent layers of the architecture.

1. All upper layer elements ul_i are to be implemented. We also refer to these elements as *cost centers.*

2. The implementation of a distinct upper layer element ul_i requires fulfilling a known subset $c_i \subset C$ with $C = \biguplus_{\mathcal{C}_i \in \mathcal{C}} \mathcal{C}_i$ of capabilities. We refer to those subsets indicating the necessary capabilities for the operation of a cost center as the cost center's *capability set.*

3. Fulfilling a capability C_i requires the activation of at least one element of a known subset S_i of lower layer elements ll_i. We refer to these lower layer elements as *applications.* Furthermore, we refer to the subset S_i which indicates the applications suitable for implementing a capability as the capability's *implementing set.* These implementing sets are not mutually exclusive. If an application belongs to multiple implementing sets, then selecting the application for the solution would simultaneously satisfy all implementing sets it is contained in, and thereby all corresponding capabilities.

4. An application which has been activated arises an operational cost p_i.

5. Implementing an application which has previously been inactive arises an additional transitioning cost tp_i, and vice versa.

6. All cost centers are subject to budget constraints. These budgets are distributed among multiple cost centers and are not mutually exclusive. That is, a cost center can receive funds from multiple designated budgets, and a budget can be designated for multiple upper layer elements.

The problem we consider is to determine which applications should be implemented in order to minimize operational and transitional costs, as well as how the budgets are consumed by the cost centers in order to realize all capabilities. This combines all the requirements that we have previously formulated.

In order to accomplish this, we derive a network flow minimization problem from the selection problem and formulate the following integer program. We first construct a graph from our problem description. Let $\mathcal{V}$ denote the set of vertices, and $\mathcal{E}$ the set of edges.

Let s be a source, and t be a sink node, and let $\mathcal{V}_-$ be the set of vertices $\mathcal{V}$ excluding these nodes s and t. Construct a node B_i for each budget b_i and edges (B_i, ul_j) for each cost center ul_j assigned to this budget. Construct one edge (s, B_i) for each budget node B_i.

Construct a capability node C_i for each capability $\mathcal{C}_i$, and let (ul_j, C_i) be the edges from each cost center ul_j to the capabilities it requires. Then let (C_i, ll_j) be the edges from the capability C_i to all applications ll_j it is implemented by.

Finally, construct one edge (ll_i, t) for each application ll_i.

Figure 2 shows an example architecture modelled accordingly.

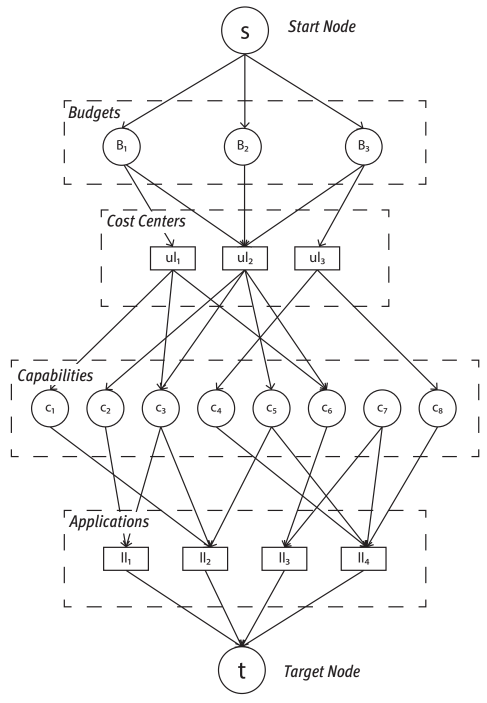


Figure 2: Abstract example for two layers in an enterprise architecture

Let $\mathcal{Z}$ be the set of applications already implemented (as to be considered for transitional analysis), and $z_i \in \mathcal{Z}$ be the implementation state of the application A_i. $F_{i,j}$ designates the flow between two nodes (i, j) and A_i is a boolean determining whether application ll_i is to be implemented. Let tn_i be a boolean variable indicating whether a transition is necessary for application ll_i. These are the variables of the integer program.

The integer program describing the problem we have specified can now be defined as follows.

$$\text{Minimize} \quad \sum_{(i,t)\in\mathcal{E}} F_{i,t} \tag{4}$$

$$\text{subject to} \quad F_{i,j} \leq b_j \quad \forall(i,j) \in \mathcal{E}, j \in \mathcal{B} \tag{5}$$

$$\text{and} \quad \sum_{(i,j)\in\mathcal{E}} A_j \geq 1 \quad \forall i \in \mathcal{C} \tag{6}$$

$$\text{and} \quad tn_i \geq A_i - z_i \quad \forall(i,t) \in \mathcal{E} \tag{7}$$

$$\text{and} \quad tn_i \geq z_i - A_i \quad \forall(i,t) \in \mathcal{E} \tag{8}$$

$$\text{and} \quad F_{i,t} = (A_i \cdot p_i) + \left(tp_i \cdot (z_i - A_i)^2\right) \quad \forall(i,t) \in \mathcal{E} \tag{9}$$

$$\text{and} \quad \sum_{(i,k)\in\mathcal{E}} F_{i,j} = \sum_{(k,j)\in\mathcal{E}} F_{i,j} \quad \forall k \in \mathcal{V}_- \tag{10}$$

4 describes the objective function. As we are attempting to minimize accruing costs, our objective is the minimization

of flow across all incoming arcs at the sink node t.

5 enforces budget limits on the edges connecting the source s to the budget nodes.

6 ensures that at least one of the applications connected to a capability is activated.

7 and 8 set the variable tn_i that indicates whether a transition of application A_i is necessary to the correct value.

9 enforces a valid flow on the edges connecting the applications to the sink t; that is, the flow is either 0 if the application is disabled, or exactly the sum of operational and transitional costs. This is accomplished by adding up the product of the boolean variable A_i defining whether the application is to be activated and the corresponding cost p_i, and the transitional cost function.

10 is a constraint common to all flow problems and forces the amount of flow entering a node to be equal to the amount of flow leaving it.

Thereby we have implemented all requirements originally specified in the definition of the selection problem for budgeted enterprise architecture optimization.

The variables A_i will contain the implementation state for application ll_i in the optimal solution, and $F_{i,j}$ will describe the amount of cash flow between the two entities belonging to i and j.

3.5 Applying the Optimization Model

To apply the optimization model to an example graph the budget constraints have to be implemented as nodes between the cost centers and the start node as described in subsection 3.4. The edges between the start and the budget nodes function as the budget constraints according to (5) in section 3.4. Cost-centers are provided by different budgets which are represented as the connecting edges between those nodes. Figure 3 shows an example of such a graph where the cost centers *Human Resources* and *Liquidity Management* are each sharing their budgets with *ERP*. Here *Application 1* and *Application 2* are not yet used in the model and can be activated by applying the denoted transition costs. To calculate an optimal solution we apply our above-described approach to such a graph with the given costs. Figure 4 sketches the optimal solution calculated by our model regarding a minimum cost-flow. This optimal solution suggests to include *Application 1* and *Application 2* with a total cost of 70 for each consisting of transition and operating costs. The total costs for this solution amounts to 165 and is reflected to the minimum cost-flow in the network.

4. IMPLEMENTATION

In order to evaluate the performance of the modeling approach we have presented and in order to apply it to exemplary inputs, we felt it be warranted to implement rudimentary software capable of generating and processing appropriate enterprise architecture representations.

4.1 Requirements

A solution stack suitable for easy implementation of the defined objective and constraints with simple and reliable interfacing with commonly utilized solvers was deemed necessary. It should also be interoperable with software capable of rendering graphs to enable visual representation. The solver should provide sufficient performance and be capable to process quadratic constraints.

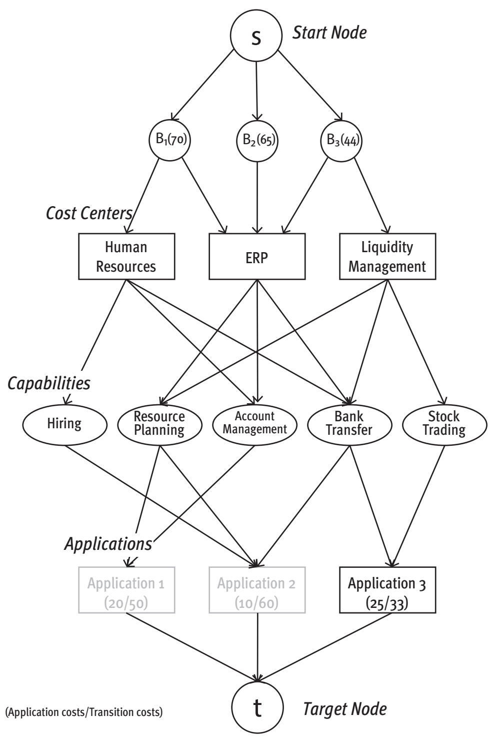


Figure 3: Concrete example with budgets and costs

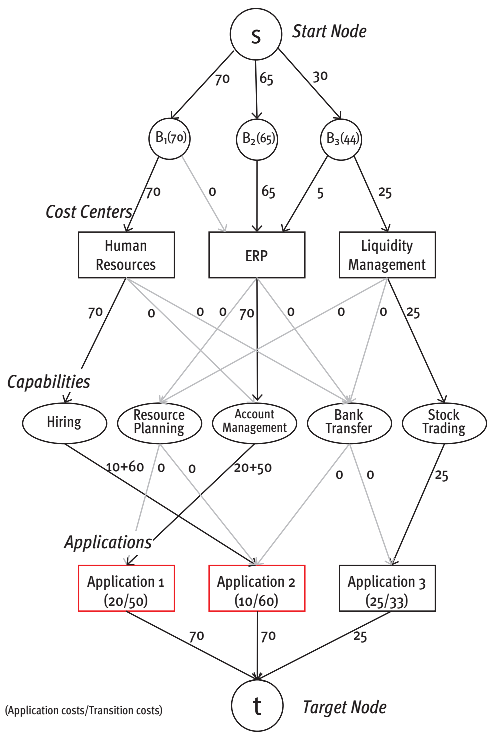


Figure 4: Solution for concrete example with cash flow

4.2 Technology

To accomplish the desired goals we have reformulated the objectives and constraints from the previous section using the Pyomo[1] optimization modeling DSL and utilized the Gurobi[2] solver to derive an optimal solution.

Furthermore, to allow for visual output of the generated or processed graph in order to increase comprehensibility we employed Graphviz[3] and the pyGraphviz module[4] for plotting the graph.

We have decided on this solution stack to allow for implementation of our solution in as little time as possible. However, we would like to point out that with some effort our results should be reproducible with an open-source solver implementation, such as GLPK[5].

Our Python program consists of three parts: a module responsible for providing the command line interface and calling appropriate methods to process and draw the graph, one using Pyomo's DSL to state the problem as an AbstractModel , and a third to allow random graph generation for evaluation purposes.

Pyomo provides a DSL to supply input parameters to an AbstractModel. The command line interface we have implemented is capable of processing these input files, as well as generating random architectures by supplying thirteen parameters: the number of budgets, cost centers, capabilities and applications, minimal and maximal budgets, transitional and operational costs, as well as density values for budgets, cost centers and applications which determine how well connected corresponding nodes are.

We have provided a copy of the source code of our implementation in a git repository[6].

5. EVALUATION

Building on our implementation we will now provide both a reasoning for our expectations pertaining to the algorithm's runtime, as well as an experimental evaluation of this runtime. This will allow us to predict the practical value of the solution we have presented.

5.1 Scalability

5.1.1 *Runtime Expectations*

As is evident from eq. 4 the objective function of our model is a linear expression and dependent upon the flow on edges between applications and the sink node t. The number of linear objective terms in the model is therefore equivalent to the number of applications.

We expect an increase in the number of edges, for instance, due to a positive variation of the amount of budget, cost center and capability nodes, to lead to linear increases in runtime due to the additional linear constraint terms.

5.1.2 *Experimental Runtime Evaluation*

For experimental evaluation, we have generated random architecture representations to be processed by our software with different values for the number of budgets, cost centers, capabilities and applications. For the first part of the evaluation, all density values were set to 0.1, and for each of these value sets the average runtime and variable count over 20 runs was determined on a system with 102 GFlops[7] of floating-point performance and 8 GB of RAM. At this density we expect the variable count to be closely related to the number of edges.

Table 1 shows the resulting runtimes for varying numbers of budgets, cost centers, capabilities and applications with fixed ratios at a connectivity density of 0.1. We have decided on these values, as they represent a common balance between the various entities, even though the connection density is higher than in most real-world architectures. [6] However, this reduces the impact of time spent in pre-solving and optimization phases on overall measurement results, and is still representative of actual use cases, as we will show the relationship between density and runtime to be inversely proportional.

Time measured includes time spent on problem generation by Pyomo and time in the Gurobi solver, including preprocessing and pre-solving time. Figure 5 shows the runtime of the solver compared to the number of variables corresponding to edge and vertex counts. We have fitted a linear model to our measurements using MATLAB and present the result in Figure 5 as well.

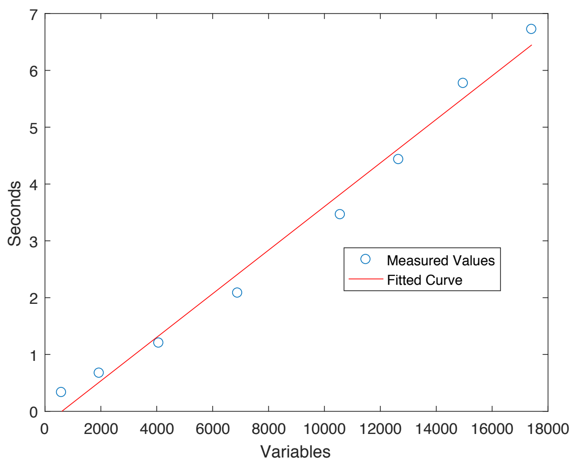


Figure 5: Time spent in solver and linearly fitted corresponding model (data from Table 1)

This shows that with increasing architectural complexity measured in terms of connectivity between different entities in the organization graph, in accordance with our expectations, the runtime of our algorithm increases linearly. We have found that this relationship holds in most cases for architectures with equivalent connection densities.

By varying connection densities in a second test run we confirm that execution time is not solely dependent on the edge count, but on connectivity as well. The results of these measurements are shown in Table 2.

Comparison of varying density values against the average time spent in the solver for each edge in the architecture yields Figure 6. This is to show that connection density is

[1]http://www.pyomo.org/
[2]https://www.gurobi.com/
[3]http://www.graphviz.org/
[4]https://pygraphviz.github.io/
[5]https://www.gnu.org/software/glpk/
[6]https://git.rwth-aachen.de/kevin.koopmann/swcseminar

[7]determined using LINPACK

Table 1: Runtime in Solver for varying numbers of budgets, cost centers, capabilities and applications with fixed ratios, average over 20 runs

Budgets	Cost Centers	Capabilities	Applications	Density	Variables	Edges	Runtime
10.0	20.0	60.0	40.0	0.1	595	514	0.331073772907s
20.0	40.0	120.0	80.0	0.1	1945	1784	0.67232862711s
30.0	60.0	180.0	120.0	0.1	4076	3835	1.19684612751s
40.0	80.0	240.0	160.0	0.1	6897	6576	2.07876989841s
50.0	100.0	300.0	200.0	0.1	10571	10170	3.45684739351s
55.0	110.0	330.0	220.0	0.1	12659	12218	4.43302592039s
60.0	120.0	360.0	240.0	0.1	14971	14490	5.768829s
65.0	130.0	390.0	260.0	0.1	17419	16898	6.72238627672s

inversely linearly related to execution time.

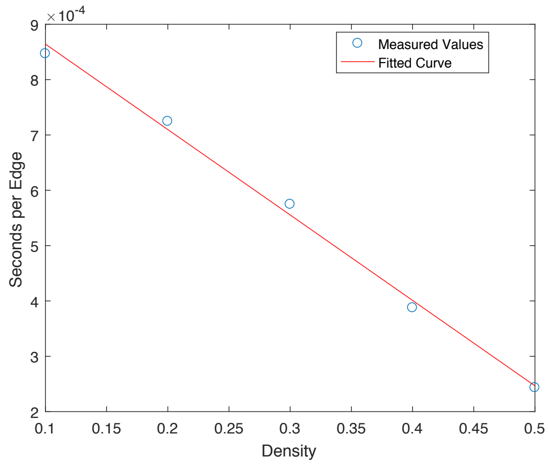


Figure 6: Seconds per edge depending on graph connectivity (data from Table 2)

We assume that the reduction in runtime in very highly connected architectures compared to architectures with lower edge counts is related to particular details regarding the implementation of the relaxation or SOCP algorithms of the solver we have used, or in the simplification of certain problem parameters in highly connected graphs. However, a detailed analysis of this effect is beyond the scope of this article.

In the following section we will analyze the consequences this entails for the application of this algorithm on real-world optimization problems.

5.2 Value for Real-World Applications

As is evident from our runtime analysis, the execution time of the Integer Program we have described is dependent upon a multitude of factors, including the number of entities on different architectural layers and the connectivity between such entities. There was a tendency towards a linear correlation between variable or edge count and runtime for all evaluated architectures.

Even at linear runtimes, this may still present challenges for scalability towards optimizing extremely large-scale enterprise architectures. Even though such optimizations will not be performed frequently in most cases, and can often be executed on large-scale systems, it is conceivable that on certain architectures execution time might prove to be prohibitive.

However, most architectures encountered in real-world applications seem to be of a rather limited size in comparison to the scenarios of our evaluation model that required high execution times. Lagerstrom et al.[6] for instance considered an architecture consisting of 407 nodes and 1157 edges, translating to a density of just 0.00698 in our notation. Evaluating a similar architecture on our test system consistently yielded a runtime of less than one second.

6. CONCLUSION

Information Technology projects within enterprises demand novel solutions of the organization problem, devised with awareness of business requirements such as departmental budgets and transition costs. With the ever increasing relevance of IT systems in business cases, we predict a correlating rise in demand for consolidation of such systems.

We have shown how this consolidation can currently be performed using optimization models and presented a different, novel model taking the business need of budgeting for varying cost centers into account, and incorporated the transition cost model introduced by Hacks and Lichter [5].

Furthermore, we have demonstrated through our runtime evaluation that our approach performes adequately for most real-world use cases, which we showed both for synthetic test cases and instances from literature.

Currently, the approach considers budgets and transitional as well as operational costs only at a single point in time. In reality, certain costs and budgets may constantly change; an extension of the approach to determine the ideal point of investment in time could address this challenge.

7. REFERENCES

[1] S. Aier and R. Winter. Virtual decoupling for it/business alignment–conceptual foundations, architecture design and implementation example. *Business & Information Systems Engineering*, 1(2):150–163, 2009.

[2] U. Franke, O. Holschke, M. Buschle, P. Narman, and J. Rake-Revelant. It consolidation: an optimization approach. In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2010 14th IEEE International*, pages 21–26. IEEE, 2010.

[3] V. Giakoumakis, D. Krob, L. Liberti, and F. Roda. Technological architecture evolutions of information systems: Trade-off and optimization. *Concurrent Engineering*, 20(2):127–147, 2012.

[4] S. Hacks and H. Lichter. Optimizing enterprise architectures using linear integer programming

Table 2: Runtime in Solver for with varying node counts and varying connectivity, average over 10 runs

Budgets	Cost Centers	Capabilities	Applications	Density	Variables	Edges	Runtime
20.0	40.0	120.0	80.0	0.5	7960	7799	1.89743793011s
40.0	80.0	240.0	160.0	0.4	25151	24830	9.62528171539s
60.0	120.0	360.0	240.0	0.3	42225	41744	23.9882300377s
80.0	160.0	480.0	320.0	0.2	50185	49544	35.8965565205s
100.0	200.0	600.0	400.0	0.1	40188	39387	33.3532226801s

techniques. *INFORMATIK 2017*, 2017.

[5] S. Hacks and H. Lichter. Optimierung von unternehmensarchitekturen unter berücksichtigung von transitionskosten. *HMD Praxis der Wirtschaftsinformatik*, 55(5):928–941, 2018.

[6] R. Lagerström, P. Johnson, and M. Ekstedt. Architecture analysis of enterprise systems modifiability: a metamodel for software change cost estimation. *Software quality journal*, 18(4):437–468, 2010.

[7] M. Lankhorst. *Enterprise architecture at work: Modelling, communication and analysis.* Springer, 2009.

[8] M. D. Mesarovic, D. Macko, and Y. Takahara. *Theory of hierarchical, multilevel, systems*, volume 68. Elsevier, 2000.

[9] M. A. Rood. Enterprise architecture: definition, content, and utility. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1994. Proceedings., Third Workshop on*, pages 106–111. IEEE, 1994.

[10] L. Trevisan. Cs261-optimization paradigms lecture notes, lecture 15. *Electrical Engineering & Computer Sciences Department, Berkeley University of California. Available at http://theory.stanford.edu/˜trevisan/cs261/lecture15.pdf [Accessed 12 Dec 2018], 2010.*

[11] S. Vodanovich, D. Sundaram, and M. Myers. Research commentary—digital natives and ubiquitous information systems. Information Systems Research*, 21(4):711–723, 2010.*

[12] R. Winter and R. Fischer. Essential layers, artifacts, and dependencies of enterprise architecture. In Enterprise Distributed Object Computing Conference Workshops, 2006. EDOCW'06. 10th IEEE International*, pages 30–30. IEEE, 2006.*

Towards Defining the Concept of View, Viewpoint, and Perspective in Domain Modeling

Marlon Schröter
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
marlon.schroeter@rwth-aachen.de

Aleksandra Pazova
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
aleksandra.pazova @rwth-aachen.de

ABSTRACT

The fundamental logic behind large-scale software solutions is almost always complex. To understand, analyze, and improve these solutions an all-embracing knowledge of the corresponding problem domain is necessary. This knowledge needs to be depicted in a way that groups with differing interests can access the knowledge relevant to them easily. A domain model is a suitable tool for depicting knowledge about a domain. However, it is hard to design a domain model depicting the all-embracing knowledge of a problem domain, while it stays accessible to stakeholder specific interests and maintains the domain's complexity and interconnectivity. Moreover, the model is the basis on which all future decisions will be made. Therefore it is necessary to make as little mistakes as possible while creating the domain model. To ensure the correctness of the model, despite the complexity of its content, a structure to base the model on is indispensable. The concept of view, viewpoint, and perspective is a concept dealing with similar issues in other model-based environments. It draws the model's focus on relevant aspects in regard to different specifications. Unfortunately, none of the existing definitions taken from other fields fit domain modeling and multiple contradictions exist among them. To overcome this, we compare several definitions from software architecture, information systems, business, and open distributed processing and check which aspects of them are applicable to domain modeling. Having analyzed and assessed the existing definitions we present a new definition specifically addressing domain modeling.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management—*programming teams, productivity, software configuration management*

Keywords

definition, defining, concept, domain model, domain, model, modeling, view, viewpoint, perspective

1. INTRODUCTION

Software solutions are applied in a vast range of domains, a few examples are managing health records in a hospital, organizing global shipping, and automatically detecting stock exchange fraud. In these examples, the problem-domains are complex and need to be fully understood to provide software assisting with the problem. However, the developers behind the software are computer scientists who are seldom familiar with the domain-specific concepts, processes, and properties. They need to acquire the necessary domain knowledge to be able to work on such projects. As such knowledge is complex and has a high degree of interconnectedness, it is not possible to obtain the necessary knowledge without any assistance. One possibility to assist with obtaining relevant knowledge is by providing a domain model in which all the domain-specific properties and their connections are depicted. Such a model is created by an expert from within the domain, who understands all the relevant interconnections and dependencies of the domain. By consulting the domain model a developer can then obtain any information necessary to tailor the solution to the needs of the respective domain. Therefore, the general purpose of the domain model is to help make complex information accessible. As different development teams work on different parts of the domain, they need to be able to draw specific information only relevant to their cause from the model. In contrast, the creator of the domain model has to depict the whole domain knowledge in it. Taking all this into account, the domain model is highly complex and needs to be accessible to the needs of software developers and domain experts. This is hard to achieve without any structure to base the model on.

Our approach to solving this problem is to apply the concept of view, viewpoint, and perspective to domain modeling. With a supporting structure, the domain expert can use the different viewpoints and perspectives to guide himself along creating the model. The concept gives software developers multiple views to draw information only relevant to their cause from the domain model.

There are several existing definitions of view, viewpoint, and perspective in fields related to domain modeling or other modeling fields. We closely examine the fields of software architecture, information systems, business, and open distributed processing.

However, none of the existing definitions fit the specific needs of domain modeling and to a certain degree, they even contradict each other. This makes it impossible to adopt an already existing definition. To create our own, domain modeling specific definition, we analyze, assess, and compare the most promising definitions from the aforementioned fields with domain modeling requirements in mind.

2. BACKGROUND

A domain model is created for a specific purpose. Therefore not everything regarding the domain needs to be depicted in the model, only information concerning the task the domain model is crafted for is supposed to be depicted in the domain model.[21] This includes the implications drawn from this knowledge. Therefore a domain model is more than just a knowledge representation, "it is a rigorously and selective abstraction of that knowledge".[7]

A domain model is coined by its purpose, which ultimately is to ensure the correct implementation of a software solution regarding a domain specific problem. It ensures this by making complex fundamentals accessible to software developers and by only containing information relevant to the software solution. Thereby the domain model serves as a backbone to the software project, shaping the design choices and being shaped by the project's purpose.[7]

3. METHODS

A domain model is created by experts from the respective domain. These experts may not be familiar with the concept of view, viewpoint, and perspective. Therefore, we begin with definitions from a dictionary and analyze the etymological relations between view, viewpoint, and perspective. As this shows what anyone would expect the concept to be just from knowing the meaning of those words. Our definitions should uphold the correlation between the three components to make applying the concept as intuitive as possible.

Continuing from the etymological analysis, we will analyze and asses the existing definitions from one field and compare them to domain modeling requirements. In order to do so, we will examine definitions inside the context of their field. The applicable parts of this context will then be assessed and documented while keeping domain modeling and etymology in mind. This should result in questions or not addressed aspects of domain modeling, which will then be resolved by consulting further fields. Doing this often enough leads to a refined concept of view, viewpoint, and perspective in domain modeling, which will then be formulated in a final definition.

To ensure quality, the used definitions have to have widespread acceptance in their respective fields. This will be determined by citation count of the defining source or importance of their author in their scientific community.

4. RESULTS

We start by analyzing the etymology of view, viewpoint, and perspective, to help domain experts, who are not familiar with the concept, to understand the fundamental correlations of the three components just from knowing the meaning of the words. Therefore we look up the dictionary definitions of these words and highlight relations and interconnections.

4.1 Etymology

Definition of view:

1. extent or range of vision
2. the act of seeing or examining

[14]

Point 1 and 2 express what is contained in a view; that the content is related to each other due to it being visible(1) or having a contextual relation(2). So a view needs to have some context regarding what is visible in the view and what is outside of the view. The contents of the view depend on the point of view.

Viewpoint relates to point of view and standpoint, it has no specific definition.
Definition of standpoint:

1. a position from which objects or principles are viewed and according to which they are compared and judged

Definition of point of view:

1. a position or perspective from which something is considered or evaluated

[15]

The position plays a crucial role as the evaluation depends on it. A change of position results in a different evaluation. Important hereby is what coins the position and how the position influences the evaluation.

Definition of perspective:

1. the interrelation in which a subject or its parts are mentally viewed
2. the capacity to view things in their true relations or relative importance
3. the technique or process of representing on a plane or curved surface the spatial relation of objects as they might appear to the eye specifically

[13]

Point one to three emphasize the interrelation of objects which are in perspective. One and two generalize this and three is an example of transforming a higher dimensional situation into a lower dimension, while maintaining the relation of objects by applying a perspective. This translates well to domain modeling, as the model is an abstraction of the real world and any relations need to be maintained.

Taking each of the etymological analyses into account we result with these interrelations: A view is a coherent collection of objects, which are all related to each other regarding at least one common aspect. The relation between the objects and which properties need to be fulfilled to be a part of the view is given by the corresponding viewpoint. Therefore one viewpoint generates one view, which can be only be assessed from the corresponding viewpoint by applying the evaluation principles of that viewpoint. A perspective is maintaining a specific relation between objects. In the context of view and viewpoint, a certain perspective is altering the view to maintain the perspective specific relations between all objects in the view. In short, a specific viewpoint generates a specific view and any perspective can be applied to that view resulting in alterations to the view, due to the purpose of the perspective.

4.2 Definitions of View

We start by analyzing the concept of view as it is the most general of the three components. From an etymological standpoint the viewpoint dictates what is inside the view, therefore it is necessary to define the concept of view in domain modeling as a foundation for the definition of viewpoint.

4.2.1 View in Software Architecture

Modeling is an essential part of creating an architecture for a software system. The architecture model represents the software system's design, which is key for a functioning software solution.[8] As the model plays a similar in domain modeling these two fields relate well.

The first definition, which plays a significant role in system architecture is by Nick Rozanski:

> A view is a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders.

[17]

In domain modeling, a view should represent the concerns of its stakeholders. However, these concerns might not only be addressed by structure and architecture, but there are also several different aspects of a domain which need to be represented (e.g processes, information flow, ...). Therefore, in regard to domain modeling, this definition reduces to a rather basic one, as only the stakeholders' concern is a common denominator between the fields.

The next definition we will look at is the definition of Philippe Kruchten's 4+1 View Model.

> For each view, we define the set of elements to use (component, containers, and connectors), we capture the forms and patterns that work, and we capture the rationale and constraints, connecting the architecture to some of the requirements. Each view is described by a blueprint using its own particular notation. For each view also, the architects can pick a certain architectural style, hence allowing the coexistence of multiple styles in one system.

[12]

Kruchten describes the content of a view as the captured forms, patterns, rationale, and constraints which fit the requirements. Adapting this approach, we can eliminate the need to address specific aspects of a domain. We can rather specify components necessary to describe aspects. This results in a modular definition, which is not subject to a fixed set of aspects. The exact technical ways of representing them are described in this definition as well, but those are not necessary for us, as we do not define the exact way of formulating a view, we only define the concept of it.

Another definition worth taking into consideration is the one by IEEE, as they define the industry standard.

> Each architecture view shall adhere to the conventions of its governing architecture viewpoint. Each architecture view shall include:
>
> a) identifying and supplementary information as specified by the organization and/or project;
>
> b) identification of its governing viewpoint;
>
> c) architecture models that address all of the concerns framed by its governing viewpoint and cover the whole system from that viewpoint;
>
> d) recording of any known issues within a view with respect to its governing viewpoint.

[11]

IEEE relates one view to exactly one viewpoint, which is also applicable for domain modeling. Moreover, the view's content is defined by the corresponding viewpoint in point c. This part of the definition fits the word analysis of view and viewpoint, as one's viewpoint determines what is inside one's view. Transferring this to domain modeling, solely the domain and the viewpoint determine the view's content. Additionally, any information deriving from the view itself is recorded inside the view, specified in point a and d. This makes sense, as it is the logical place to store such information.

Models in system architecture heavily rely on the representation of a structural aspect relevant to their stakeholder. How this aspect is depicted varies depending on how technical a definition's approach is. A similarity between system architecture and linguistic analysis is the direct relation between viewpoint and view, with the viewpoint governing the view. What parts of the architecture are represented in the view solely depends on the viewpoint.

Applicable to domain modeling, is the relationship between view and viewpoint, the representation of stakeholders' concerns and Kruchten's modular approach to define the content of a view.

4.2.2 View in Information Systems

In an information system information is collected, stored, and distributed.[16] A domain model's purpose is to depict domain knowledge in a structure, which can be easily understood by non-domain experts. Therefore similarities between information systems and domain modeling can be expected.

The Zachman Framework is a "logical construct (or architecture) for defining and controlling the interfaces and the integration of all of the components of the system".[25] To do this, Zachman has created a table in which each row depicts a certain view on the subject and each column an abstraction. An abstraction is supposed to reduce the complexity of the subject and focus on a single aspect of the whole.[24]

The rows of the Zachman Framework closely relate to what a viewpoint is supposed to do in a domain model.

The Abstractions depict the independent variables that constitute a comprehensive depiction of the subject or object being described, including:

1. Material Description, Structure
2. Functional Description, Transform
3. Spatial Description, Flow
4. Operational Description, Operations
5. Timing Description, Dynamics
6. Motivation Description, Strategies

[24]

Each abstraction deals with a different aspect of the same subject. What the subject is, is defined by the viewpoint. Therefore the total of the abstractions can be related to a view. It is necessary to look at all the different abstractions to understand the interrelationships inside of the subject. The first abstraction deals with the material composition - the structure - of the subject. As already stated, while assessing software engineering, the structure is only one aspect of any domain. More aspects are defined in the other abstractions of the Zachman Framework. The aspects of transformation, location, operation, dynamics, and strategies are helpful additions to our findings.

Information Systems maintain the approach of directly relating a view and it's viewpoint. They provide more aspects to better understand and depict the complexity and interrelations of the assessed subject. These aspects are depicted separately from each other in a tabular form. The way the information is divided and presented is not applicable to domain modeling, but multiple of the aspects are valuable.

4.2.3 View in Business

Business is a domain in which modeling is a strategy to answer questions about the business idea and how it is supposed to generate profit. As the business model is "an architecture for the product, service and information flows, including a description of the various business actors and their roles"[19] it depicts multiple aspects of the underlying business in its context. This is very similar to our purpose of the domain model.

With their business model, Eriksson and Penker provide a tool for companies to assess their products and services. To be able to assess the product one needs to understand the product's environment. This is done by modeling the business the product is settled in, resulting in a business model. A key component of their model is the view.

Each view is expressed in one or more diagrams. The diagrams can be of different types, dependent upon the specific structure or situation in the business that it is depicting. Diagrams capture the processes, rules, goals, and objects in the business, and their relationships and interactions with each other. The Eriksson-Penker Business Extensions use four different views of a business, and they are:

1. Business Vision View
2. Business Process View
3. Business Structural View
4. Business Behavioral View

[6]

Not imposing the types of diagrams fits domain modeling well, as each domain varies and a fixed way of depicting the domain will not suit the vast amount of domains. An individual approach is more suitable as it can be tailored to the needs of the domain. A new aspect is the one of depicting rules inside a view. This aspect is not present in any of the models we have analyzed so far and it is a welcome addition. The other aspects have already been assessed while analyzing previous definitions. The four views presented by Eriksson and Penker are not applicable to domain modeling, as the focus on only one of the subject's aspects. To understand a domain it is necessary to have all aspects present and not only one of them. Multiple views are important, but the difference between the views is not due to the focus shifting between aspects, it is due to the effect different viewpoints have.

4.3 Definitions of Viewpoint

In this section, we will analyze the concept of viewpoints. The viewpoint is the structure we are using to create the model, the components inside the frame view. We insert the viewpoints to the view so we can have more detailed structure.

4.3.1 Viewpoint in Software Architecture

The approach of defining viewpoint in the software architecture is similar to this in the domain modeling. That's why we will find here a lot of definitions which can be applied also in the domain modeling.

The first definition we will look at is that of Ian Sommervill. In his opinion, in order to be able to manage effectively viewpoints and to satisfy the requirements of all involved sides, we must arrange them hierarchically:

A viewpoint is a way of collecting and organizing a set of requirements from a group of stakeholders who have something in common. Each viewpoint, therefore, includes a set of system requirements. Viewpoints might come from end-users, managers, etc. They help identify the people who can provide information about their requirements and structure the requirements for analysis.

[18]

In the domain modeling, the concept of structuring the viewpoints in a hierarchy is not applicable. The domain model should work as a whole mechanism which has all parts together at the same level but connected with different logic or functional connections. We are still separating the model but not in different layers. The Domain itself is a layer from the division of the software system.

The second definition we will look at is this of IASA, it says:

> A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views.

[5]

This definition explains the same as the conception of Eric Evans for the subdomains and the modules. ”Identify cohesive subdomains that are not the motivation for your project. Factor out generic models of these subdomains and place them in separate modules.”[7] In the domain model, we are also collecting templates which matter to us and then we apply them to the model. If we replace the word subdomain with view and modules with viewpoint we will receive the same definition but instead of defining stakeholders, it's guidelines, principles, and template models we should be directly influenced by the viewpoint and it's view's purpose itself.

Now we will look at the definition published in a paper in the University in London. In the opinion of the authors, we should avoid the use of a single or common representation scheme and instead of this we should use multiple viewpoints to partition the domain information, the development method, and the formal representations.

> A ViewPoint is a loosely coupled, locally managed object which encapsulates partial knowledge about the application domain, specified in a particular, suitable formal representation, and partial knowledge of the process of software development.

[3]

That means that in their opinion in every part of the developing process there is a stakeholder who has a particular responsibility and role in building it. In domain modeling, these stakeholders are mostly the domain expert and developers. They both have different knowledge, for example, the expert has limited knowledge about the software development and the developer cannot understand the concepts and jargon the domain expert uses. If we take that their different knowledge, in the sphere they work, is the particular objects that they encapsulate in own concepts we can easily say that we receive the different viewpoints and can refer this to our understanding of domain modeling.

Another important definition is this from IEEE. It says:

> An architecture viewpoint shall specify:
>
> a) one or more concerns framed by this viewpoint;
>
> b) typical stakeholders for concerns framed by this viewpoint;
>
> c) one or more model kinds used in this viewpoint;
>
> d) for each model kind identified in c), the languages, notations, conventions, modeling techniques, analytical methods and/or other operations to be used on models of this kind;
>
> e) references to its sources.

[11]

We will look at this definition dividing it into the separate parts it has. The first two parts correspond also to the domain modeling and are the basic concept in creating a good model. We should focus on the concerns and look at them from all viewpoints, in particular, considering the opinion of all stakeholders who are involved in building the model. Only when we have gathered much of all the concerns, we can start creating the model. But still in the different situation, different stakeholders will have misunderstandings and this is why we have to frame them in viewpoints and apply these viewpoints to the model so everyone is satisfied. Now we will look at the third part. Generally, we have one main model which is our basic structure and we are developing it as we apply viewpoints, perspectives, etc. But if we look at the different kinds of models as the subdomains we can refer this part to the IASA definition and say it is also applicable. Part d) on the other hand can be met with a metamodel for the model kind that defines the structure and conventions of its models which we already have looked addressing the metamodel definition and concluded that is not relevant to the domain modeling, so this one is also not relevant. And the last part of this definition is not relevant because the domain model is an aggregate work by many people which ideas in many situations comes from brainstorms from all stakeholders in the equip and it is not possible to define exactly who is the author of which part. And we really do not if we know it was teamwork and who are the members of the team.

Next we will look at the most common and widespread definition during the time of writing, by Nick Rozanski and Eoin Woods.

> A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views. Architectural viewpoints provide a framework for capturing reusable architectural knowledge that can be used to guide the creation of a particular type of (partial) AD.

[17]

As IASA are using the same definition for viewpoint except for the second part. We are not going to look at the first part again. Instead, we are going to give more attention to the second part of this definition and particularly the last sentence. In their book, Rozanski and Woods try to give the whole conception of structuring a software model for a software system. Their aim is to collect long-term applied methods and norms and to put them in a common conception which can be used by everyone like a template for creating a software architecture. And that is exactly what we try to do with the domain model. We need some frameworks and rules to apply so we can define the domain model as a common concept. In their book, Rozanski and Woods are also defining the viewpoint as:

> Viewpoints (and views) are an approach to structuring the architecture definition process and the architectural description, based on the principle of separation of concerns. Viewpoints contain proven architectural knowledge to guide the creation of an architecture, described in a particular set of views (each view being the result of applying the guidance in a particular viewpoint).

[17]

Structuring the model in different subsection as can give us the flexibility to change some details in it without to have to change the whole model or the whole structure of the model. If we apply this also to the domain modeling for example as subdomains it will be much easier when the domain experts and the developers have disagreements later. An important word in this definition is "approach" as actually, we are looking for the best approach to define domain modeling.

Next, we will present a definition that diverges from the earlier ones. This is the definition that relies on metamodels:

> A viewpoint is defined in relation to one or more metamodels. For each viewpoint, a non-empty set of view types is defined. In a viewpoint instance, any number of instance views for each of the view types can be dynamically created.

[20]

This concept is not applicable to the domain modeling because it says that the viewpoint is based on one or more metamodels, but in fact, the domain model is the main structure which can have different layers in it, but it is not based on them. We apply them like additional qualities, but the main and center structure stays the model. It also says that depending on the data we are applying also the viewpoint will be different. Our aim is that we create a system which has a concept which serves us as a template and then by different needs we apply some details and perspectives to bring it in context. That is why this definition is not relevant to our concept of a viewpoint.

4.3.2 Viewpoint in Business

Firstly, we will present the definition of IEEE of a business viewpoint.

> Through the analysis of the conceptual model and the process model, a business process viewpoint was built, composed by processes and objects - represented between the grouping relationships.

[22]

This definition is also applicable to the domain modeling as we also have different groups of processes and objects which have relationships between them and we want to group them in the peculiarities every group carry with it and their different purposes. The difference here is that the IEEE business definition relies on analysis of governance, risk, and compliance, and they build the conception of viewpoint above this. But in the domain modeling, this is not the main topic. In the domain modeling, we are looking at the risks but it is most probably that we put it in separate viewpoints, like security and not building the whole concept of viewpoint above this. In domain modeling, the risk and the security are also important but are very often forgotten. That is why we want to create a concept like a viewpoint in the domain modeling so we can be able always to add new categories. To model, the behavioral, structural and informational structure of the business viewpoint IEEE is using ArchiMate. ArchiMate is an open and independent enterprise architecture modeling language of business domains. The definition ArchiMate gives us for a viewpoint is the following:

> Viewpoints define abstractions on the set of models representing the enterprise architecture, each aimed at a particular type of stakeholder and addressing a particular set of concerns. Viewpoints can be used to view certain aspects in isolation, and to relate two or more aspects.

[1]

The concept of this definition is similar. Again if want to apply certain aspects we are going to use viewpoint. That is why this definition is also applicable.

At next we will look at the definition of Prof. Dr. Knut Hinkelman about a business viewpoint:

> A view is what you see and a viewpoint is where you are looking from.

[9]

This definition says that what is and what is not shown in a view depends on the scope of the viewpoint and on what is relevant to the concerns of the stakeholders. And this matches exactly with our understanding of view and viewpoint in the domain modeling. The view is used as a frame of the complete model and the viewpoints are different categories in this model. How we see the model depends on what for a viewpoint we are going to take. But when we apply everything together than we receive the complete model.

The Open Group Architecture Framework (TOGAF) is a framework for enterprise architecture that provides an approach to designing, planning, implementing, and governing an enterprise information technology architecture. It defines the viewpoint as:

A viewpoint defines the perspective from which a view is taken. It defines:

1. How to construct and use a view
2. The information needed
3. The modeling techniques for expressing and analyzing it
4. A rationale for these choices (e.g., by describing the purpose and intended audience of the view)

[2]

If we take this definition, first of all, we will admit that the view depends on the perspective and the perspective depends on the viewpoint. So, in this case, the viewpoint will be the main part of the model. It will define how to construct and use a view which means that the view will be the subpart of the viewpoint and if we want to change or add something first we will have to change the viewpoint appropriate. So in the domain modeling, the viewpoint will be the base in building the model and the view will be the different additional structures which describes more detailed the different spheres and themes of the viewpoint. In this definition, the viewpoint has the needed information. As we assumed that the viewpoint is the main part it is obvious that it will contain the information we are building on. Maybe in the next level, the view, we will have some additional information but the main information still will be in the viewpoint. The same is with the modeling technique. The main part will be defined from the viewpoint as a fundamental part. In the last point as a purpose audience we can use our client and the customer how will use the domain after it is ready. His purpose will be the topic which contains the domain model (to inform, to entertain, as means of payment... etc).

4.3.3 Viewpoint in Open Distributed Processing

For the standardization of open distributed processing is presented the reference model RM-ODP. This is a framework which is separate in 5 different viewpoints. The idea is the following:

A viewpoint is a subdivision of the specification of a complete system, established to bring together those particular pieces of information relevant to some particular area of concern during the analysis or design of the system. Although separately specified, the viewpoints are not completely independent; key items in each are identified as related to items in the other viewpoints.

[23]

This concept is to provide separate viewpoints, each satisfying an audience with an interest in a particular set of aspects of the system. The viewpoint language is associated with each of this viewpoints and that optimizes the vocabulary and presentation for the audience of each viewpoint. This definition could help us with the problem of the Ubiquitous Language that we find in building the domain model. This language should be a summary of the domain expert language, the developer language, and any other stakeholders. As most complex system specifications are so extensive that no single individual can fully comprehend all aspects of the specifications. If we add that we all have different interests in a given system and different reasons for examining the system's specifications, we will easily see that it is impossible to define one complex viewpoint for everyone. But if each of these stakeholders put his own technical expression in one viewpoint which describes the problems he is working on we can solve this problem. For a more in-depth analysis of the definition, we will look at each of the viewpoints separately and analyze it.

The enterprise viewpoint, which focuses on the purpose, scope, and policies for the system. It describes the business requirements and how to meet them.

[23]

In the domain modeling, this viewpoint can make the connection between the developers and the world outside the creating participants (the client the marketing strategist and so on). That means it can be useful in the domain modeling.

The information viewpoint, which focuses on the semantics of the information and the information processing performed. It describes the information managed by the system and the structure and content type of the supporting data.

[23]

This viewpoint will be also useful and important for the domain modeling as it can translate the terms and the specific vocabulary of every stakeholder, as the developer, the expert and also the external users, and translate it to one another. It also works with the structure of the data every stakeholder applies and to separate and combined it appropriately.

The computational viewpoint, which enables distribution through functional decomposition on the system into objects which interact at interfaces. It describes the functionality provided by the system and its functional decomposition.
The engineering viewpoint, which focuses on the mechanisms and functions required to support distributed interactions between objects in the system. It describes the distribution of processing performed by the system to manage the information and provide the functionality.

[23]

In the domain modeling normally we connect this two viewpoints in one fundamental viewpoint. This functional decomposition we define as a complex viewpoint which consists of every subviewpoint we already mentioned. But if we want to take the example from RM-ODP and separate them in different subsections this both viewpoints can be put together because in the domain modeling the managing with information is included in the functional decomposition.

The technology viewpoint, which focuses on the choice of the technology of the system. It describes the technologies chosen to provide the processing, functionality, and presentation of information. and the domain itself is our technology.

[23]

The last viewpoint of the RM-ODP is not applicable in the domain modeling because in the domain modeling the domain itself is the technology we need and any other additional technologies are not needed.

4.4 Definitions of Perspective

Now we are going to analyze the concept of perspectives. The perspectives are the qualities that a viewpoint or a group of viewpoints has. It allows us to apply additional properties to the objects in the model.

4.4.1 Perspective in Software Architecture

Let's look at the Nick Rozanski and Eoin Woods definition:

An architectural perspective is a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the systemâĂŹs architectural views.

[17]

As this is a general definition of a perspective it should be applicable to domain modeling as well. If there are aspects which influence multiple viewpoints, then these aspects should be applied by a certain activity, guideline or tactic throughout the whole domain model. These are formulated as a perspective. With them, we can easily change a certain part or characteristic of the model without the need to change the whole model. This allows us a flexible work with the model and as in later stages of the developing are always appear some updates or changes it saves us a lot of time to repair it because anyway we will have to change the whole structure of the domain. That's why perspectives are not only applicable but also really useful.

4.4.2 Perspectives in Information systems

The first Definition which is from Information systems and we will look at is of a student from the Copenhagen Business School.

Each use case describes a scenario in which a user interacts with the system being defined to achieve a specific goal or accomplish a particular task ... The perspective provided by use cases reinforces the ultimate goal of software engineering: to create products that let customers do useful work.

[10]

In the domain modeling, there is always the conflict between the domain expert and the developer who are trying to do the best so at the end, the customers have something useful, easy and understandable to work with. Although there are different approaches to tackling the conflict between these two stakeholders, such as urban language, there is still no concrete solution to the problem. In this case, the conflict also involves the client who has requirements that are impossible or extremely difficult to implement. This leads to situations in which the experts are dissatisfied with the task and are opposed to the client, the client, however, remains disappointed with the failure to fulfill his wishes and so on. But if we apply this definition to the domain modeling, we will have a chance to resolve this conflict much easier and quicker. In this way, the end user will be satisfied with the work done and the easy handling of the product. That is why this definition can be useful also in the domain modeling. The next definition is also from the information systems. It is defined as:

A modeling perspective in information systems is a particular way to represent pre-selected aspects of a system. Any perspective has a different focus, conceptualization, dedication, and visualization of what the model is representing.

[23]

This definition is easy to apply to the domain modeling because if we use the concept of view as a frame and this of a viewpoint to characterize the different topics and parts of the system we need also a perspective to all preselected aspects which are different for every part (viewpoint) of the system. As the perspectives in this definition have different focus they will be appropriate to the different qualities one viewpoint has. They can either upgrade the viewpoint or they can be used as a completely different part which is connected with the viewpoints like a subsystem, one more layer. So practically we can use the definition as an additional description or property of a viewpoint.

4.4.3 Perspectives in Business

An understanding of the service provider and IT Services from the point of view of the business, and an understanding of the business from the point of view of the service provider.

[4]

We can assume that in our case the services provider and the IT Services are the domain experts and the developers and the business is the domain model. But if we take this concept we are not going to be able to connect it with the concept of view and viewpoint as they both are about the structure and this is about the meaning and understanding of the domain model. Moreover, the fact that it is about the concept of a model it will take it directly at the beginning as a first step that will determine a next as view and viewpoint. Like this, the whole perception will be changed. And we will have to put the whole quality properties in one structure- the viewpoint- which will do the building it much more complex and difficult to understand. So as a conclusion we can say that the definition is applicable but not useful in the domain modeling.

Evaluating the definitions found was essential to create an understanding of differing applications for the concept of view, viewpoint, and perspective. As we want to apply the same concept to the development of a domain model, we will discuss in the next section how the results of our research influenced the creation of our definition for view, viewpoint, and perspective in domain modeling. We were influenced by the total amount of definitions we looked in this chapter, but we manage to take the essentials of each and summarize it. For example, the definition of the viewpoint aims to describe that the viewpoint should be used as a sample that can be repeatedly used and its applying is important to make all of the stakeholders content. It uses the concepts of IASA, Nick Rozanski, the reference model RM-ODP and others. In this way we came up also with the definitions of view and perspective.

5. DISCUSSION

Before presenting our definitions of view, viewpoint, and perspective we want to highlight the relation between the three components. A viewpoint and a view are directly related to one another as the viewpoint defines the content of the view. The difference between them is that the view contains the relevant information about the domain, while the viewpoint describes the observer's interests in the domain. A perspective is not limited to one view or one viewpoint. It is applied after having generated the view from the viewpoint with the intention to either maintain specific relations between objects inside the view or to ensure certain qualities. Therefore a perspective is applicable to any view and alters it depending on the relations it is supposed to maintain or ensure.

As the view is the most fundamental part of the domain model we will present our definition for it first.

Each view is depicted as a set of elements in one or more diagrams. The way of expressing the view is not limited to any modeling language or other technical constraints. A view represents several aspects of the domain relevant to the interests of the view's corresponding viewpoint. The aspects that need to be included are:

1. The aspect of structure
2. The aspect of transformation
3. The aspect of location
4. The aspect of dynamics
5. The aspect of strategy
6. The aspect of regulation

Their relationships and interactions with each other need to be depicted as well.

As in our opinion without the viewpoint the view can not be completely defined we will present also our definition for viewpoint.

A viewpoint is a collection of a set of requirements, roles, and responsibilities. It is a template which guides the process of developing the view. Moreover, all viewpoint collectively define the different aspects of a system and reflect the concerns of all stakeholders by collecting all needed requirements and priorities for the total satisfaction of all interested, in the domain model, sides.

As last we are going to present our definition of perspective.

A perspective defines a particular quality property. The quality properties of the perspective define the behavior and the characteristic of a system. The perspective analyzes and modifies the domain model to make sure it exhibits a particular quality property. If necessary, these quality properties are applicable to some or all of the views.

Our definitions are strongly influenced by the etymological analysis and the resulting relationship of the components. As Nick Rozanski is the only author presenting definitions for all three components, his and our findings of how to relate the components correspond. Besides this, the applicable results gained from analyzing the field of software architecture are minimal, as software architecture is too focused on structural aspects and how to technically define the model. However, information systems and business construct models with a broader underlying subject and their principles are therefore more applicable to domain modeling. This results in a higher number of represented aspects being depicted in the model and a flexible way of describing the model itself.

6. CONCLUSION

As a conclusion we can say that there are many definitions and concepts in several scientific fields regarding what view, viewpoint, and perspectives are. Some of these definitions are easy to compare with the concept of the domain model but there is still no precise and clear concept to capture all cases, exceptions, and details that arise when we develop a domain model. However, we are of the opinion that a universal concept would make the development of the domain model much easier and would prevent frequent and repetitive mistakes. So we tried to gather all these definitions and extract the most valuable of each by combining it into a new, more comprehensive, domain-defined modeling definition.

7. REFERENCES

[1] Architecture viewpoints. 2013.

[2] T. 9. Views and viewpoints.

[3] M. G. Anthony Finkelstein, Jeff Kramer. Viewpoint oriented software development. *Imperial College of Science, Technology & Medicine, University of London*, 21.

[4] G. T. Authority. Business perspective.

[5] C. Cooper-Bland. Views and viewpoints. *An association for all IT architects.*

[6] H.-E. Eriksson and M. Penker. Business modeling with uml. *New York*, pages 1–12, 2000.

[7] E. Evans and M. Fowler. *Domain-driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley, 2004.

[8] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 2002.

[9] P. D. K. Hinkelman. Enterprise architecture views and viewpoints. pages 1–12.

[10] J. Holck. 4 perspectives on web information systems. page 4, 2002.

[11] IEEE. Iso/iec/ieee systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, Dec 2011.

[12] P. B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, Nov 1995.

[13] Merriam-Webster. Perspective | definition of perspective by merriam-webster.

[14] Merriam-Webster. View | definition of view by merriam-webster.

[15] Merriam-Webster. Viewpoint | definition of viewpoint by merriam-webster.

[16] G. Piccoli. *Information Systems for Managers: Texts and Cases*. Wiley Publishing, 1st edition, 2007.

[17] N. Rozanski and E. Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional, 2005.

[18] I. Sommerville. *SOFTWARE ENGINEERING*. Pearson Education, 9th edition, 2011.

[19] P. Timmers. Business models for electronic markets. *Electronic markets*, 8(2):3–8, 1998.

[20] I. R. M. A. USA. *Computer Systems and Software Engineering: Concepts, Methodologies, Tools and Applications*, volume 2068. IGI Global, 2018.

[21] V. Vernon. *Implementing Domain-Driven Design*. Pearson Education, 2013.

[22] P. Vicente. A business viewpoint for integrated it governance, risk and compliance. Sept 2011.

[23] Wikipedia. Rm-odp. 2018.

[24] J. Zachman. The zachman framework for enterprise architecture. *Zachman International*, 79, 2002.

[25] J. A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3):276–292, 1987.

Assessment of cost factors for cloud vs. on-premise software operation

Julian Krebber
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
julian.krebber@rwth-aachen.de

ABSTRACT

Operating big and complex software applications requires an analysis to assess individual requirements and most importantly occurring costs. Technical circumstances and specific software requirements may influence the price for hosting on-premise or subscribing to a cloud making a cost forecast difficult. Therefore, cost models and load based optimization methods are needed for a prediction in order to operate software at minimal cost.
An overview of the current capabilities, options and cost parameters is provided in order to discuss the advantages and disadvantages of the most significant approaches focusing on saving costs. To achieve a reliable cost calculation the paper presents current research on the cost assessment for a centralized data center and compares several models such as Calculating Cloud Computing Cost Effectiveness (CCCE) and Cloud Cost Amortization Model to determine the Total Cost of Ownership (TCO). Completing the analysis, a detailed interpretation and discussion on the quality and possible improvements of the mentioned models is given.

Keywords

cloud, on-premise, hybrid cloud, cloud application, cost model, cost factors

1. INTRODUCTION

For a long time, organizations operating software applications had no other choice than to acquire necessary hardware and manage it on their own. At this time no alternative solution pattern was available on industry level and thus no different choice than on-premise execution existed. In order to get enough computational power or storage it was necessary to estimate the needed capacity while fulfilling individual requirements. In general cost assessment focused on implementing the solution at minimal cost by comparing hardware products and services needed to operate the IT systems. With increasing research and publications dealing with topics about distributed cloud systems and the possibility to provide resources based on dynamic demand, several cloud providers commercialized this approach by offering a new paradigm of operating software and introducing new business models. From this time on big tech companies built a wide range of cloud services satisfying customer's needs of all sizes. With that paradigm shift organizations face new challenges in the field of executing applications, its cost assessment and optimizing their optimal software operation strategy.
In this paper a structured overview of my research is provided regarding the question what and how cost factors must be considered when deciding whether to execute software in the cloud or on on-premise systems and if already proposed cost models cover relevant aspects. After a detailed search on related work no overview paper with a conclusive discussion could be found and hence this is the aim of this paper.
In section 2 an overview of three major computing patterns is provided and their advantages and disadvantages are explained. Afterwards in section 3 the focus lies on relevant cost factors and present approaches using metrics to assess these. Based on the identified factors three proposed cost models are presented and discussed. In the end a conclusion and possible improvements are given.

2. BACKGROUND

2.1 Definition

2.1.1 *On-Premise*

As mentioned in the introduction the cloud solution is relatively innovative and gained popularity in the last ten years. Before that on-premise execution was the primarily followed pattern. Compared to cloud services an on-premise solution is described as the static pattern because of an often fixed number of hardware systems and its given limitation.

Before cloud services evolved on-premise hosting was referred to hardware residing in the local building. With the rise of cloud vendors not only cloud systems evolved but on-premise architecture shifted towards a more flexible pattern taking advantage of massive data centers. Cloud providers recognized the need of companies to keep hardware which is not shared and under total control of the respective organization. Therefore, the option of fixed-cost hardware was created combining the advantage of providing dedicated hardware with no need of managing the hardware itself, maintaining the data center and around the clock observation.

The advantage of the mentioned architecture is the complete control of the system which resides at the company. Specific hardware can be chosen, special operating systems

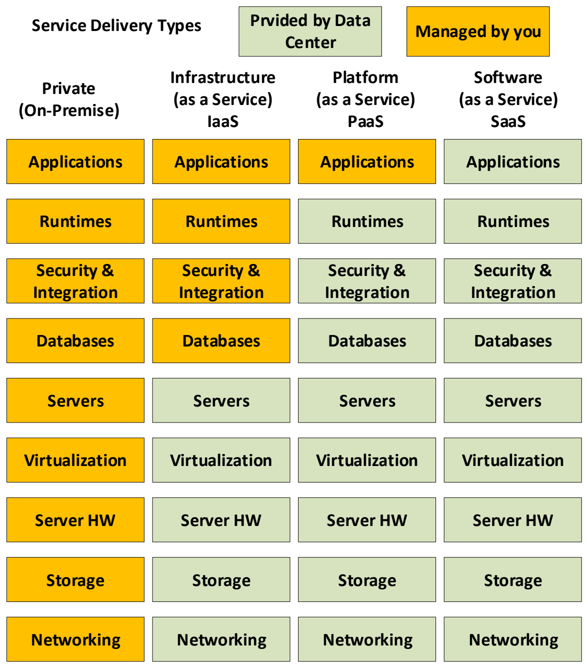


Figure 1: Service Delivery Types

installed and every single component customized. The company can design and operate its own cloud by their means. Two options of on-premise hosting exist: On the one hand a company can operate its own hardware and oversee the whole life cycle, including buying the hardware. On the other hand, on-premise hosting, offered by cloud vendors, can be chosen. This option abstracts from the low-level detailed management of hardware.

2.1.2 *Cloud*

The term *Cloud* is a broad term which describes the approach of pooling computational power, storage or software and providing them as a service over the internet to serve multiple consumers [1]. The allocated services can vary depending on the individual needs of the customers. Generally speaking cloud environments offer networking, storage, servers, virtualization, integration, security and ready-to-use software applications abstracted from the customer.

The aim of this service based approach is to hide the complexity of managing IT infrastructure [6]. Without this abstraction the user would have to deal with detailed configuration and management of hardware devices making the operation of cloud applications more complex. Three types of cloud services evolved in the past years, namely Infrastructure-, Platform- and Software as a Service (IaaS, PaaS and SaaS) which are briefly defined in Figure 1 [12]. In this paper SaaS will not be discussed, due to its comparatively specialized nature and no differentiation will be made between IaaS and PaaS because of their similar nature concerning cost. Cloud vendors can offer clients specific supplementary services such as load-balancing or api-gateways.

Aside from choosing how much control is wanted, benefits of the cloud are scalability and the possibility to get resources on-demand or even use distributed Data Centers [1]. This is described in more detail in section 2.2. Due to the use of the pay-as-you-go model [3], very high upfront costs for acquiring hardware can be avoided and redirected to lower recurring cloud leasing costs. To make this possible the cloud must be a measured service[1]. It has to be assumed that a trusted cloud provider is chosen in regard to reliability and security. Otherwise the cloud could pose a major issue for clients.

When it comes to Efficiency, Data Centers often have less than 30% utilization [2] due to uncertain demand. When using a cloud this is not the businesses concern [8] and the cloud providers have the possibility to achieve a much better efficiency, because of multiple and diverse clients. For the cloud providers it is therefore crucial to know their clients demand for an optimal resource distribution. Therefore, most providers will offer different subscription models. For example either to reserve resources in advance or get them on-demand as described by Sakir and Ibrahim Yucel [22].

These models make forecasting a businesses own demand crucial, because using cloud resources without reservation is more expensive, making cost assessments and optimization even more important. Whether the cloud fits a businesses needs depends on the requirements and can be assessed with this paper.

2.2 Advantages and Disadvantages

As the key concepts of cloud and on-premise were given there are certain characteristics that differ between them. A short comparison between the key factors can be seen in Table 1 followed by a more detailed explanation of each criteria:

Criteria	Cloud	On-premise
Hardware, staff managing cost	Low	High
Reliability	Depends on backup	Depends on backup
Scalability	Fast (on-demand)	Slow
Security	Partially out of hands	Businesses responsibility
Customization	Can be limited	Unlimited
Expandability	Not needed	Slow, Investment required

Table 1: Differences between cloud and on-premise criteria

- Reliability
  An unreliable software system can be problematic: Downtime of the software system, unexpected inaccessibility of the web services etc. will result in profit loss. To ensure reliability it is crucial to minimize down- and recovery time. The recovery time especially regarding hardware failure may be shorter when using a cloud provider due to unused hardware on standby. Reliability should be ensured by the cloud provider if applicable. It is essential to use backup systems to prevent loss of data and boosting recovery time.

- Scalability
  Scalability means how well a change of available resource can be made to fit the demand. When using an own data center there are limited resources. If a certain consistent demand exists that the businesses data center meets, scalability is not an issue. But strongly

varying demands can make the businesses data center uneconomical due to mostly unused hardware that is required in order to meet peak demands. To deal with this issue one can either just use the clouds on-demand advantage or own the resources needed to fulfill the average demand and use the cloud when additional resources are needed. It should be taken into consideration how fast or uncertain the demand changes. If there are rapid changes it must be taken into account that shut down hardware needs time to be available, while keeping it in standby minimizes this time it maximizes operation cost. Clouds may also be able to deal with this issue more easily.

- Security
  When working with sensitive and restricted data, there is a risk of losing the data or getting it leaked to a third party. When the businesses does not use a cloud provider, security is completely in the businesses hands. That means all the on-site security must be handled to prevent unauthorized access or physical damage. Concerning the system level, if security is needed but the IT department is not capable of maintaining it, a trusted cloud provider with security knowledge is obviously favorable. Otherwise the cloud does pose more security risks. Because of the availability over the internet it can be attacked more easily [4]. This risk can be reduced by using a more sealed off network with an owned data center. When handling restricted data, privilege abuse may be problematic [13]. In this case the cloud provider must be trusted as well as the enterprises employees. More interesting could be privacy issues [13]. On the businesses data site the location of the data is known. This must not be true for a cloud provider or may introduce additional fees for choosing the vendors hosting site.

- Network
  It can be helpful not having resources centralized at one site. There are cloud services owning different hosting sites, eventually globally, that will offer distributed hosting. As shown in Figure 3 most data centers are located near strongly populated areas or near the internet backbone, achieving a stable and fast internet connection with low latency. Be aware that choosing special hosting locations can result in higher vendor prices.

- Customization
  On-premise does leave everything to the business. Concerning customization this is beneficial making the use of specific hardware, operating systems, software and databases customized to the businesses needs, possible. Cloud providers are aware that their customization options are limited [7]. Ergo these options may not be sufficient, especially for large companies [16].

- Expandability
  Buying additional hardware requires an investment and must be planned long in advance. Delivery times can take several months, while upgrading the businesses cloud resources can be relatively quick as described in the scalability section.

- Dependence
  When the businesses whole system is set up in the cloud it is strongly depending on the provider. To increase flexibility and reduce dependency migration capability is relevant. If there is a issue with the provider like unreliability, costs can be reduced by changing the provider. So there should be an awareness of any kind of technological [19] or contract lock-in issues. [13]. There are further dependencies such as the Internet. Note that the internet is not necessarily required with a data center on-premise.

2.3 Hybrid Cloud

After introducing the cloud and on-premise approaches and underlining the key factors of each alternative a definition and explanation of a possible combination of both variations is given in this section. This pattern is called hybrid cloud and consists of mainly two hosting parts specifically the public and private cloud. Thereby the private cloud can be an on-premise self-hosted hardware system, or a rented fix-cost based cloud system stationed in a data center and maintained by external companies. A survey conducted in 2015 revealed that 58% of the 930 questioned companies were using a hybrid cloud for business purposes while 30% based their IT just on public clouds [15].

While the previous section dealt with advantages and disadvantages of single services, the hybrid approach melds the advantages of both and offers a structure to meet critical requirements of organizations as for example data privacy, security or compliance while simultaneously retaining the possibility to outsource to cost-efficient public cloud systems [9]. The basic motivation of establishing and using a hybrid approach is to gain the flexibility and agility to execute specific software components regarding their specifications in different cloud systems while keeping the overall system running without disruptions. Following this pattern companies can store sensible data in their self-operated cloud enforcing individual regulations while running the business logic in a public cloud for its dynamic scalability, elasticity and the advantages mentioned in the previous parts. Furthermore, it represents an attractive approach for companies, which have already invested in own IT systems and need to expand their capacities without huge investment volumes in local hardware and software.

Another option is to use only specific cloud services as for example data storage to outsource single components to a public cloud or the other way around adding missing capabilities as easy and fast as possible. To guarantee the seamless operation of such distributed software systems, communication and data transfer between these clouds is crucial, introducing new technical and economical parameters, that should be considered. Thus the hosting setup needs to allow workloads to be switched between clouds preferably automated [21]. This represents one disadvantage of the hybrid cloud pattern, namely the more complex communication relation between clouds. Therefore, software must be adjusted and subsequently made executable for different environments. For that reason, the settings and runtime environment must be compatible. This can for example be achieved through bundled virtual machines or containerized software components which are then executed independently of the current cloud and underlying hardware. This configuration leads to the opportunity of assessing the cost for executing software

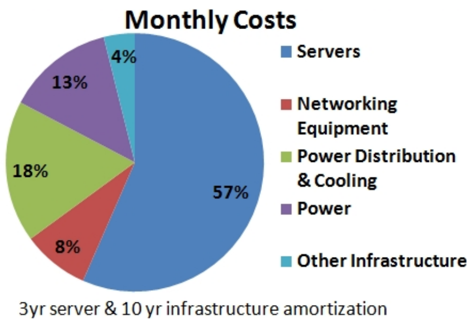


Figure 2: Chart of monthly data center cost [5]

Upfront cost	Recurring cost
Site	Site rent
Hardware	Hardware maintenance and upgrades
Software	Software maintenance and updates
Migration	Staff
	Electricity
	Cooling
	Network(ISP)
	Tax

Table 2: Upfront and recurring cost for an on-premise site

in different clouds and optimizing the hosting costs by choosing the best-fitting cloud with lowest costs even daily. For that purpose, general cost models are required, but before discussing these I am going to define and present relevant cost factors in the next section.

3. COST ASSESSMENT

To clarify the term cost used in this paper, it mainly refers to the expenditure for using a data center. That means not easily quantifiable cost or very individual costs will not be discussed in this paper. Before investing capital, it should be planned well ahead what data center solution saves the most money. In this section the main expenses will be covered, that are partly derived from Tak et al. [17].

3.1 Cost factors

Before diving into the different cost models (see section 3.2), I am going to present cost factors that are relevant for using an on-premise or cloud solution. First it should be differentiated between upfront costs and recurring costs. In the previous section three possible IT hosting architectures which can be implemented to execute software on a large scale were introduced and described. A major difference between cloud and on-premise costs are the upfront costs: While they are comparatively small when using a cloud provider, it is a massive investment to build up an own data center. In this chapter the focus will therefore lie on the on-premise cost when it comes to upfront costs. A quick overview of the factors is given in table 2. It is understandable that most of the costs cannot be evaluated in great detail due to a case to case dependence Tak et al [17].

3.1.1 Upfront costs

First of all, a site for the data center is needed. There are various possibilities to acquire such a site such as building or renting. Of course, renting would make it a recurring cost instead of an upfront cost. To get the site operational, hardware must be acquired. There is a variety of hardware racks available. What kind of hardware should be used, is influenced by the requirements that should be met concerning processing power and storage. To determine processing power and storage the CCCE method (see section 3.2.2) may be used from [3]. If a hybrid approach is not used, there may be extra hardware needed to cope with peak loads reducing overall cost effectiveness. Also, additional hardware like switches must be taken into account. The hardware is not of much use without software, so software licenses need to be purchased. This must not necessarily count to the upfront investment, because software vendors can also have different pricing models such as subscriptions. The purchase of the software is equally relevant for cloud usage, if required. Furthermore, switching to a cloud provider can pose migration costs but makes the company more flexible.

3.1.2 Recurring costs

Without an already owned data center the cloud is without question the cheapest short-term solution. To determine the optimal long term solution, the upkeep or operational costs are of great importance for a future-proof company. When the business can calculate the approximate upkeep of owning a data center it is possible to estimate how high the prices of the cloud will reasonably be. As the acquisition of an on-premise data center is much more expensive than using the cloud [1], the upkeep cost for on-premise needs to be lower than the cloud cost. Otherwise the on-promise solution is not profitable.

Running an own data center will introduce costs for staff and maintenance [1]. More technicians, software administrators and further employees are necessary in comparison to the cloud to keep hardware, software and databases fully operational. With time hardware will inevitably fail and must be repurchased or upgraded. The same is applicable for software, that needs to be maintained and updated, too [1]. This may introduce further cost when software licenses are not acquired with a subscription model that includes maintenance or free upgrades.

Choosing a site location is not only a matter of real estate prices and taxes. Cooling the servers is necessary and will lead to higher electricity bills, if the environmental climate cannot be used for it. That is one of the reasons, why it is advisable to build a data center in relatively cold regions. Much more important are electricity prices in general, because electricity will make up a huge part of the upkeep costs as shown in Figure 2. Electricity is clearly essential for the data center, which will consume huge amounts of energy proportionally to its size. Placing the data center at a site with a stable power supply near customers (see Figure 3) and handling electricity concerns is convenient. For example power generators may be necessary to prevent damage to hardware and data [23].

To reduce the amount of power required in an own data center the power required by energy consumers must be reduced. Servers that are currently unused should therefore be set to an energy saving mode or shut down. Instead of solely measuring the used energy of a data center the usage or efficiency of the data center is of essence. Because the cloud commonly uses a pay-as-you-go model it is necessary to determine cost by resource usage. The main resources that

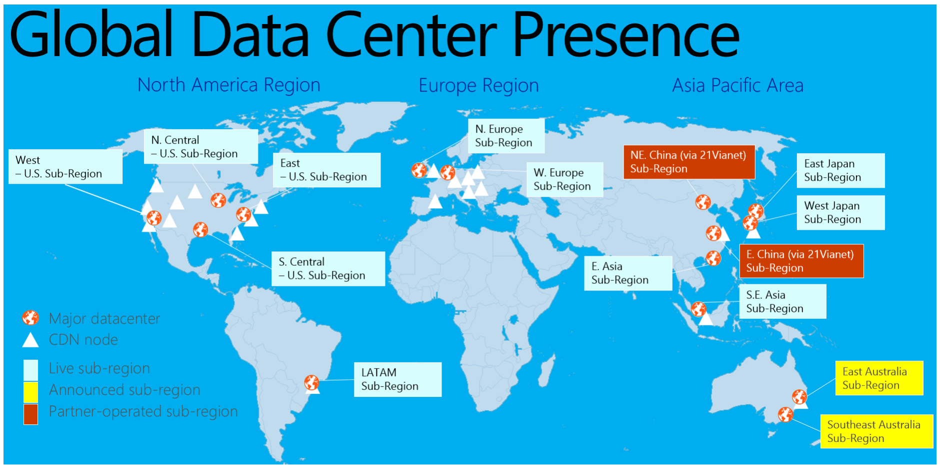


Figure 3: Microsoft Azure data center locations [10]

are relevant are therefore CPU time, memory and storage. If transfers with a significant amount of data are required, networking costs can rise [7].

3.1.3 *Execution time, Storage and Data transfer*

To run a data center efficiently and reduce costs, the execution time of the programs, the memory and storage needed and the data that must be transferred needs to be calculated. As mentioned in section 2.1.2, cost can be greatly reduced by determining these factors in order to shift the usage from on-demand resource acquisition to resource reservation and resource bidding [22]. While calculations will be done in the cost models chapter 3.2, I am going to give a summary for execution time, storage, and data transfer:

- Execution time
  Execution time mainly relates to the number of used computers [3] and CPU time. CPU time is the duration a program gets 100% of the CPUs attention. Why is it more relevant to use the CPU time rather than the number of cores or the CPU speeds? The most prominent reason is that the CPU time is comparable. It is difficult to determine how high the utilization of the CPU is and therefore its power consumption, which is depending on the currently used software. In the end it comes down to the time a program needs to finish. The CPU time can be translated to cost normally given in price per hour. This is very helpful for instance for choosing a CPU. If a program has a maximum time limit, a CPU must be chosen that guarantees to upkeep the limit. Otherwise by calculating with the price per CPU hour one can determine if it is less expensive to execute a program on a slower CPU with a lower cost per hour or a fast CPU with a higher price per hour.

- Storage
  The price for storage given by hard drives depends on their capacity. The only way of reducing costs is to minimize the needed storage volume. Still, purchasing larger quantities of storage will lead to a price reduction per GB [20]. Depending on the hard drive and cloud provider there is a limited number of transactions for the hard drive and the provider may charge for them [6]. When Cloud storage is used I/O latency can exist [20]. Cloud providers can also offer backup systems or dedicated database systems.

- Data transfer
  Transferring data can become another cost factor. Sending or receiving significant amounts of data can be necessary. This may not cost if data is transferred in the local network or inside the data center. But as soon as a third party is involved cost will occur. This may be the enterprises data center Internet Service Provider (ISP) or the cloud provider. In this case moving data inside the cloud can be affected, too. None the less the maximum data traffic allowed will be limited or charged per GB transferred [6]. For this reason, using the hybrid cloud model is less attractive, if the cost for moving or accessing data between data centers is too high.

3.2 Cost models

In this section cost models are presented which focus on calculating the cost for owning a data center or using the cloud. The cost models can be used as a guideline to predict the most beneficial way of using a data center. In other words: To lease or not to lease from the cloud. All variables used can be found in Table 4.

3.2.1 *Net Present Value Decision Model*

Edward Walker presented a way of calculating the cost of CPU time [19] and cost of storage [20] represented by the net present value (NPV). His calculations form an approach for comparing own resources with leased resources and are therefore used as a basis for additional methods. The idea behind the NPV is to consider the change of money value over time. By doing so, money received immediately would be considered worth more than receiving it at a later date. This can be expressed by [18]:

$$PV = \frac{FV}{(1+k)^T} \tag{1}$$

To calculate the NPV of an investment summing up the annual cash flow (profit-cost) C_T for Y years leads to [19]:

$$NPV_P = \sum_{T=0}^{Y-1} \frac{C_T}{(1+k)^T} \tag{2}$$

To incorporate Moore's Law that seems to apply to IT including CPU performance and storage capacity, equation 1 is altered to [19]:

$$PC = \frac{FV}{(\sqrt{2})^T} \tag{3}$$

To consider the available processing capacity the following can be used:

$$TC = TCPU * H * \eta \tag{4}$$

To include CPUs performance obsolescence, the total capacity TC can be combined with (1) [19]:

$$NPC_P = TC * \sum_{T=0}^{Y-1} \frac{1}{(\sqrt{2})^T} = TC * \frac{1 - \frac{1}{(\sqrt{2})^Y}}{1 - \frac{1}{(\sqrt{2})}} \tag{5}$$

The real cost of an CPU hour is [19]:

$$C_R = \frac{NPV}{NPC} \tag{6}$$

With given equations (2), (5) and (6) buying computing resources leads to [19]:

$$C_P = \frac{\left(1 - \frac{1}{\sqrt{2}}\right) \sum_{T=0}^{Y-1} \frac{C_T}{(1+k)^T}}{\left(1 - \left(\frac{1}{\sqrt{2}}\right)^Y\right) * TC} \tag{7}$$

In case of leasing resources as in a cloud the NPC should be altered. No depreciation of computing capacity is needed, as the cloud provider will keep them up to date [19]:

$$NPC_L = Y * TC \tag{8}$$

Equation (7) is altered accordingly:

$$C_L = \frac{\sum_{T=0}^{Y-1} \frac{C_T}{(1+k)^T}}{Y * TC} \tag{9}$$

When it is assumed that purchased CPUs are kept up to date, depreciation does not matter either, but repurchase/upgrade costs must be included [19]:

$$NPV_U = C_0 + \sum_{T=1}^{Y-1} \frac{C_T - A}{(1+k)^T} \tag{10}$$

This gives me the new cost of a CPU hour for purchasing and upgrading [19]:

$$C_U = \frac{C_0 + \sum_{T=1}^{Y-1} \frac{C_T - A}{(1+k)^T}}{Y * TC} \tag{11}$$

Neven Vrček and Slaven Brumec have conducted an analysis of Edward Walker's real cost of CPU hour calculation with focus on the utilization rate η. They concluded that the cloud is significantly cheaper if the utilization rate is low. As already highlighted in this paper, they confirm that many variables play a role for cost. Therefore, the variation of the variables should be reevaluated constantly [18]. Now I will shift from CPU performance to Storage where the NPV approach can be applied, too. For purchase [20]:

$$NPV_{SP} = \sum_{T=0}^{Y} \frac{P_T - C_T^S}{(1+I_K)^T} + \frac{S}{(1+I_K)^Y} - E \tag{12}$$

For lease [20]:

$$NPV_{SL} = \sum_{T=0}^{Y} \frac{P_T - C_T^S}{(1+I_K)^T} - \sum_{T=0}^{Y} \frac{L_T}{(1+I_R)^T} \tag{13}$$

This gives us the possibility to calculate ΔNPV [20]:

$$\Delta NPV = NPV_{SP} - NPV_{SL} \Rightarrow \tag{14}$$

$$\Delta NPV = \sum_{T=0}^{Y} \frac{C_T^{SL} - C_T^{SP}}{(1+I_K)^T} + \frac{S}{(1+I_K)^Y} + \sum_{T=0}^{Y} \frac{L_T}{(1+I_R)^T} - E \tag{15}$$

When considering that a disk controller is needed, disks must be purchased overtime to satisfy storage needs and disks must be replaced because of failure, getting the disks salvage value E_T and the capital cost E as the result [20]:

$$E = \frac{E_T}{(1+I_K)^T} + C \tag{16}$$

$$= \frac{((\lceil V_T \rceil_\Omega - \lceil V_{T-1} \rceil_\Omega) * \Omega + R_T) * G_T}{(1+I_K)^T} + C \tag{17}$$

$$E_T = ((\lceil V_T \rceil_\Omega - \lceil V_{T-1} \rceil_\Omega) * \Omega + R_T) * G_T \tag{18}$$

To include the varying capital cost E_T substitute E in equation (15):

$$\Delta NPV = \sum_{T=0}^{Y} \frac{C_T^{SL} - C_T^{SP} - E_T}{(1+I_K)^T} + \frac{S}{(1+I_K)^Y} + \sum_{T=0}^{Y} \frac{L_T}{(1+I_R)^T} \tag{19}$$

Now the operating cost for storage C_T^{SP} and C_T^{SL} will be calculated, considering electricity needed and the human factor:

$$C_T^{SP} = (365 * 24) * \delta * (P_C + P_D * \lceil V_T \rceil_\Omega) + \alpha * H_T \tag{20}$$

$$C_T^{SL} = \beta * H_T \tag{21}$$

Using $\rho = (\alpha - \beta)$ results in:

$$\Delta NPV = \sum_{T=0}^{Y} \frac{-\rho * H_T - (365 * 24) * \delta * (P_C + P_D * \lceil V_T \rceil_\Omega) - E_T}{(1+I_K)^T} + \frac{S}{(1+I_K)^Y} + \sum_{T=0}^{Y} \frac{L_T}{(1+I_R)^T} \tag{22}$$

By watching the market prices Walker found out that the SATA disk prices can be predicted with

$$G_T = K_0 * e^{-0.438 * T} \tag{23}$$

Assuming an annual disk replacement rate of 3%:

$$R_T = 0.03 * \Omega * \lceil V_T \rceil_\Omega \tag{24}$$

At last simplifying I_R and I_K to I_F and assuming the disk salvage value S as the disk price multiplied by a depreciation factor gets the final storage formula:

$$\Delta NPV = \sum_{T=0}^{Y} \frac{C_T^S - E_T + L_T}{(1+I_F)^T} + \frac{S}{(1+I_K)^Y} - C$$

$$S = \gamma * \Omega * \lceil V_T \rceil_\Omega * K_0 * e^{-0.438*T} \quad (25)$$

$$C_T = -\rho * H_T - (365 * 24) * \delta * (P_C + P_D * \lceil V_T \rceil_\Omega)$$

$$E_T = (1.03 * \lceil V_T \rceil_\Omega - \lceil V_{T-1} \rceil_\Omega) * \Omega * K_0 * e^{-0.438*T}$$

For an more detailed description of each step, look at the sources [18, 19, 20]. With these formulas Walker discovers with his case values: The longer the life expectancy of a disk is, the more purchasing storage should be the preferred option instead of leasing it. This summary of Walker's model showed how the formulas were derived, creating the possibility to easily alter them according to variables a company wants to consider or adjusted to the current market.

3.2.2 *Cloud Computing Cost Effectiveness*

One way of calculating the costs which will help to negotiate with cloud providers is the Calculating Cloud Computing Cost Effectiveness (CCCE) method [3]. This method consists of 10 steps, illustrated in Figure 4, giving Walkers calculations a structured real world applicable approach:

1. Select the Application (App):
   Assuming there is an Application App which is equally suited for cloud and on-premise computing.
2. Determine how long the execution takes (T_A):
   How long TA is varies, but mostly there is a minimum Tmin limited by technical reasons and a Tmax where if App took longer than Tmax, operating the Application App would lose its purpose. So Tmin<=T_A<=Tmax should hold [3].
3. Estimate how many computers are needed to achieve T_A(r):
   Estimating how many computers are needed can be tricky, so [3] can be used as reference. It is assumed that the execution time T_A depends on the number of computers used, as well as on the processing power of the used computers, the application complexity, the volume of data to be processed, the database size and the number of crude actions per query. Their exact correlation for calculating T_A is unknown. Slaven Brumec et al. still present a way to determine the factor for each variables by setting some of them to a constant value and measuring others [3].
4. Calculate the cost for on-premise resources (C_P):
   This can be done by using formula (7).
5. Calculate the cost for cloud provider resources (C_S):
   This can be done by using formula (9).
6. Compare C_P and C_S:
   Depending on the outcome either leasing the resources or purchasing them will be more beneficial. This can be difficult to decide, but if a certain utilization of the own resources cannot be guaranteed, the cloud will likely have the better cost per CPU hour [18].
7. Estimate data volume and increase rate (b):
   To run the application successfully the required volume of data should be determined as well as the annual increase of it.
8. Calculate the ratio between leasing or buying storage:
   In other words, the formula (25) is required. If $\Delta NPV \geqslant 0$ purchase of storage is more favorable. Otherwise leasing storage would have a better value. Three different case studies were made with different storage capacities ranging from 500GB to 10TB. The conclusion was that leasing is favorable for individuals or very small enterprises, as long as the technological lifetime of a drive is not exceeded. For small and medium sized enterprises leasing is more favorable, but for large enterprises it is buying. As many parameters factor in, a much more concrete statement should be possible for individual cases [3].
9. Analyze the cost of leasing in the cloud:
   Most cloud providers will provide a list price if the business knows its required resources. Otherwise these may be calculated [3]. Main parameters that have an direct cost effect in the biggest clouds, as with Microsoft, Amazon or Google can be listed: Number of used computers, processing power of the computers, volume of sent/received data, size of DB, Number of r/w transactions, amount of data in RDB, the OS etc. [3].
10. Bargain for an acceptable and fair price:
    Cloud vendors usually publish their prices. The smaller the cloud provider the bigger is his interest to get the enterprise as his customer. The bigger the enterprise is, the bigger will a cloud vendors interest be. That means if the business is not sure where to get its resources, there will be bargaining potential. As already discussed in this paper, the business can calculate how much an on-premise solution would roughly cost. If the Cloud vendor exceeds this limit, there is in this aspect, no gain in using the cloud. I also discussed how to calculate the cloud cost. That cost is the lowest achievable bargaining amount. Descending below it would not generate enough profit for the cloud provider. So the business can use this information to get a reasonable price in the bounds of $C_L < x < C_U$ for processing power and storage [3].

3.2.3 *Addition to Walker*

For further research purposes: Based on the NPV concept Tek et al [17] introduce the formulas to calculate hardware, os licenses, database licenses, electricity and cloud instance costs. The needed input is determined by benchmarking (empirically), similarly to the CCCE method. This can be considered a addition to Walker because it includes costs that are not covered in his NPV model.

3.2.4 *The Financial Model*

This model uses a different calculation method than the NPV in order to calculate the TCO. In Table 3 all needed variables can be found. The annual depreciation is determined by EL, SV and C_{pc}.

$$C_d = \frac{C_{pc} - SV}{EL} \quad (26)$$

The annual power consumption equals the power consumption per instance, times all systems. The power consumption per instance is the product of the maximum power consumption per system, power utility charge per system and the

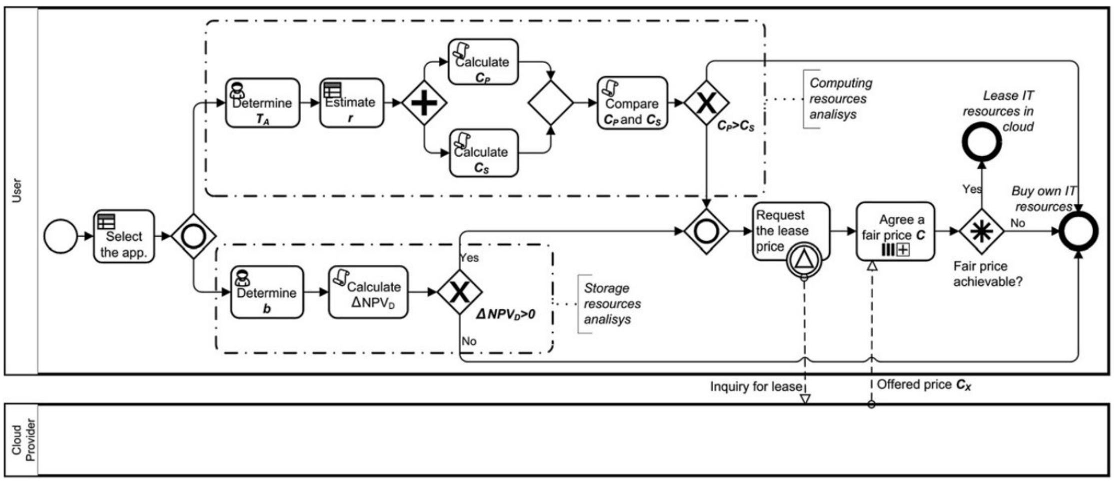


Figure 4: CCCE method [3]

power utilization efficiency of the datacenter [23]:

$$C_p = H * P_{max} * P_u * PUE \tag{27}$$

The C_m, C_{sc} is given. Ajeh et al state, that one system requires about $23m^2$. So:

$$C_s = S * C_a \tag{28}$$

Now the TCO can be calculated [1]:

$$TCO = (C_{pc} + C_p + C_m + C_sc + C_d + C_s) * N \tag{29}$$

$$\mu = \frac{H}{24 * 365} \tag{30}$$

The equations are very simple but show how different parameters can be taken into account [1].

3.2.5 Cloud Cost Amortization Model

The cloud cost amortization model is similar to the financial model, but it is more adjusted to the real world and more detailed. Detailed meaning more parameters are considered that have not been included yet, such as Virtual Machines(VM) or different types of software licenses. This is shown in table 5, where all needed variables for this model are displayed. To make different costs comparable amortization (depreciation) must be considered. A cost amortization parameter can be calculated by [8]:

$$A_{rp}(t) = \frac{(1+\alpha) * t}{A_p} \tag{31}$$

Now server, software, networking, support and maintenance, power, cooling, facility and real estate space cost can be calculated. The sum gives us the TCO:

$$TCO = C_{se} + C_{so} + C_n + C_{su} + C_p + C_c + C_f + C_{sp} \tag{32}$$

As in the other models, all the needed input can be obtained by either benchmarking, contacting a cloud operator or collected from industry statistics [8]. Server cost:

$$C_{se} = VI_{ps} * N_{serv} * A_{rp}(t) \tag{33}$$

Concerning software cost, it is assumed that three types of software licenses exist:
Type 1 software includes the OS and software being licensed by suite and priced per virtual image.
Type 2 software includes Application Server, VM software and every other software that is licensed by the number of processors.
Type 3 software includes management software that is licensed by the number of processors it manages, which leads to [8]:

$$C_{so} = (S_s * VI_s * N_s + S_o * VI_o * N_o + S_m * VI_m * N_m) * A_{rp}(t) \tag{34}$$

This method does not consider the network cost to be associated with data transfer cost, but the cost of the hardware the network consists of, which would be e.g. switches:

$$\begin{aligned} C_n &= P_s * N_{switch} * A_{rp}(t) \\ N_{switch} &= S_{NIC} * P_{NIC} * \frac{N_{serv}}{N_{port}} \end{aligned} \tag{35}$$

In this model, the support and maintenance cost would be better described as staff cost:

$$C_{su} = N_t(T_{use} * N_{serv} + T_{idle})R_{sal} \tag{36}$$

The total power cost not only includes the servers and switches but uninterruptible power supplies (UPS) and lighting etc., too:

$$C_p = L_s * E_s * S_{rp} * N_{rack} * A_{rp}(t) \tag{37}$$

To calculate the cooling cost, it is assumed that all the power consumed is converted to heat. Therefore, a parameter L is defined, that represents how much of that power is consumed by the cooling system. Statistically L could have a value of 0.6 leading to the formula [8];

$$C_c = \frac{L * (1 + P) * C_p(t)}{H} \tag{38}$$

C_f consists of the equipment needed in the racks, like PDU, KVM and cables etc.:

$$C_f = N_{rack} * VP_{fp} * A_{rp}(t) \tag{39}$$

Real estate can be very costly due to special requirements,

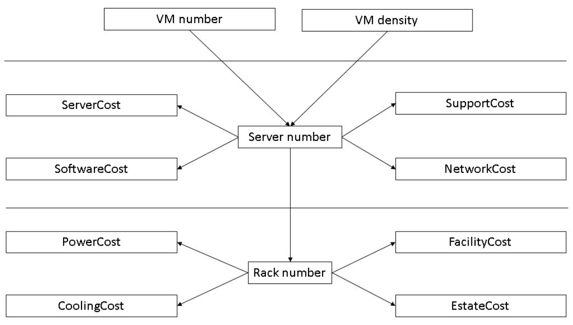


Figure 5: Three-layer model to calculate utilization cost [8]

such as cooling [17]:

$$C_{sp} = A_p * S_{SPACE} * A_{rp}(t)$$
$$S_{SPACE} = \frac{R_{SF} * N_{rack}}{R_{SPACE}} \tag{40}$$

The racks can be quite heavy, so the maximum weight the floor is able to hold must be considered. That means equation (41) should be true for the parameters used in formula (40):

$$C_{pressure} \geqslant \frac{N_{serv} * W_{server} + N_{rack} * W_{rack}}{S_{SPACE}} \tag{41}$$

In order to include the utilization cost derived from the used VMs, a calculation strategy consisting of three layers as shown in figure 5 can be used. The figure is an overview of the order in which the presented costs can be calculated, by only using the VM number and VM density as input. To apply this method, the Number of physical servers N_{serv} and the number of racks N_{rack} must be calculated. N_{rack} can be calculated by applying a bin-packing problem algorithm [8]:

$$N_{serv} = \left\lceil \frac{N_{VM}}{VM_{dens}} \right\rceil \tag{42}$$

$$N_{rack} = Bin - Packing(N_{serv}, V_{rack}, UVserver) \tag{43}$$

By using formulas (42) and (43), finally the eight aspects of cost are gotten. Their sum will be the utilization cost, which dynamically copes with the users demands. This concludes the cloud cost amortization model, with the idea of using VMs as a parameter.

3.2.6 *Practical Use*

For calculating the TCO for cloud and on-premise operation, cloud providers might offer the necessary tools. These may not be objective, although they come in handy as a good guideline or indicator for cost calculation with a specific provider. Based on the cost calculation it can be important to negotiate prices when using the cloud [13].

4. MODEL DISCUSSION

Edward Walker assumed that a server cluster has no salvage value after its retirement because only a small market exists for used CPU equipment. He also assumed that the server cluster's operational life has no expected cash revenue and that the lease price is stable. These assumptions simplify his formulas, but if the market changes, his formulas can become inaccurate. To calculate a realistic result, I would recommend further development of his formulas. For example, price volatility in the online CPU market could be incorporated. I still consider his method as an appropriate and useful way of calculation, because he considered: Moore's Law, depreciation, storage and disk replacement rates, capital expenditure, utility consumption, human operator cost and salvage value. Some formulas may need altercation like G_T the cost per GB. The formula was created by analyzing the SATA market, so it may need to be updated and checked if it also holds for new types of drives like a SSD. I will not be comparing the financial model, instead the more detailed cost amortization model will be discussed: The cost amortization model assumes that all servers have the same CPU, memory and disk, due to commonly homogeneous hardware in data centers [8], thus not differentiating between processing power and storage like the CCCE method. This makes it difficult to assess the profit of using the cloud, if the cloud provider does not offer all-in-one VMs but processing power and storage separately.

5. CONCLUSION

The possibility to outsource processing power and storage leads to the assessment whether outsourcing is a profitable option. Therefore, I presented the main advantages and disadvantages of the concepts on-premise and cloud. It becomes clear that there are certain aspects such as scalability and security, that could result in a preference for a system but that depends on the use case. Combining on-premise and the cloud results in a hybrid approach, that can be used to just partly lease from the cloud. To determine the best approach, the costs for it must be calculated. The parameters that can be taken into account differ, but the major ones were identified. To make a cost-effective future proof decision, I looked into models that provided a guideline of how to take the parameters into account in order to calculate the cost of owning or leasing the needed resources. Due to the size of the topic and its diversity still much more research could be done. For example, a guideline for optimal resource division in a hybrid cloud setting could be created.

References

[1] D. E. Ajeh, J. Ellman, and S. Keogh. A cost modelling system for cloud computing. *Proceedings - 14th International Conference on Computational Science and Its Applications, ICCSA 2014*, pages 74–84, 2014. doi: 10.1109/ICCSA.2014.24.

[2] A. Benik. The sorry state of server utilization and the impending post-hypervisor era, 2013. URL `https://gigaom.com/2013/11/30/the-sorry-state-of-server-utilization-and-the-impending-post-hypervisor-era/`.

[3] S. Brumec and N. Vrček. Cost effectiveness of commercial computing clouds. *Information Systems*, 38(4): 495–508, 6 2013. ISSN 03064379. doi: 10.1016/j.is.2012.11.002.

[4] D. A. Fernandes, L. F. Soares, J. V. Gomes, M. M. Freire, and P. R. Inácio. Security issues in cloud environments:

A survey. *International Journal of Information Security*, 13(2):113–170, 2014. ISSN 16155270. doi: 10.1007/s10207-013-0208-7.

[5] J. Hamilton. Overall Data Center Costs, 2010. URL `https://perspectives.mvdirona.com/2010/09/overall-data-center-costs/`.

[6] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson. Cost-benefit analysis of cloud codlputing versus desktop grids. In *IPDPS 2009 - Proceedings of the 2009 IEEE International Parallel and Distributed Processing Symposium*, pages 1–12, 2009. ISBN 9781424437504. doi: 10.1109/IPDPS.2009.5160911.

[7] M. J. Lee, W. Y. Wong, and M. H. Hoo. Next era of enterprise resource planning system. Review on traditional on-premise ERP versus cloud-based ERP: Factors influence decision on migration to cloud-based ERP for Malaysian SMEs/SMIs. In *Proceedings - 2017 IEEE Conference on Systems, Process and Control, ICSPC 2017*, volume 2018-Janua, pages 48–53, 2018. ISBN 9781538603864. doi: 10.1109/SPC.2017.8313020.

[8] X. Li, Y. Li, T. Liu, J. Qiu, and F. Wang. The Method and Tool of Cost Analysis for Cloud Computing. *2009 IEEE International Conference on Cloud Computing*, pages 93–100, 2009. ISSN 978-1-4244-5199-9. doi: 10.1109/CLOUD.2009.84.

[9] D. S. Linthicum. Emerging Hybrid Cloud Patterns. *IEEE Cloud Computing*, 3(1):88–91, 2016. ISSN 23256095. doi: 10.1109/MCC.2016.22.

[10] J. Markx. MSAzure, 2014. URL `https://joranmarkx.wordpress.com/2014/09/15/update-on-microsoft-azure-data-center-locations/`.

[11] R. McFarlane. Let's Add an Air Conditioner, 2005. URL `https://searchdatacenter.techtarget.com/news/1148906/Lets-add-an-air-conditioner`.

[12] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Technical report, National Institute of Standards and Technology, 2011. URL `http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf`.

[13] J. Opara-Martins, R. Sahandi, and F. Tian. A Business Analysis of Cloud Computing: Data Security and Contract Lock-In Issues. In *Proceedings - 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2015*, pages 665–670, 2015. ISBN 9781467394734. doi: 10.1109/3PGCIC.2015.62.

[14] C. D. Patel and A. J. Shah. Cost Model for Planning , Development and Operation of a Data Center. *HPl-2005-107 (R.1)*, 107:1–36, 2005. doi: 10.1128/JB.183.19.5751. URL `http://www.hpl.hp.com/techreports/2005/HPL-2005-107R1.pdf`.

[15] RightScale. State Of The Cloud Report. Technical report, Right Scale, 2015.

[16] S. Schilling. *Cloud-Computing als Betriebsmodell f{ü}r ERP- Systeme – Nutzenaspekte und Bedenken im Vergleich zwischen Theorie und Praxis.* PhD thesis, Universität Koblenz Landau, 2012.

[17] B. C. Tak, B. Urgaonkar, and A. Sivasubramaniam. Cloudy with a chance of cost savings. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1223–1233, 2013. ISSN 10459219. doi: 10.1109/TPDS.2012.307.

[18] N. Vrcek and S. Brumec. Role of utilization rate on cloud computing cost effectivness analysis. *International Conference on Information Society (i-Society 2013)*, pages 177–181, 2013.

[19] E. Walker. The real cost of a CPU hour. *Computer*, 42 (4):35–41, 4 2009. ISSN 00189162. doi: 10.1109/MC.2009.135.

[20] E. Walker, W. Brisken, and J. Romney. To Lease or Not to Lease from Storage Clouds. *Computer*, 43(4):44–50, 4 2010. ISSN 0018-9162. doi: 10.1109/MC.2010.115.

[21] J. Weinman. Hybrid Cloud Economics. *IEEE Cloud Computing*, 3(1):18–22, 2016. ISSN 23256095. doi: 10.1109/MCC.2016.27.

[22] S. Yucel and I. Yucel. Estimating the cost of digital service delivery over clouds. *Proceedings - 2016 International Conference on Computational Science and Computational Intelligence, CSCI 2016*, pages 623–628, 2017. doi: 10.1109/CSCI.2016.0123.

[23] J. Yuventi and R. Mehdizadeh. A critical analysis of Power Usage Effectiveness and its use in communicating data center energy consumption. *Energy & Buildings*, 64:90–94, 2013. ISSN 0378-7788. doi: 10.1016/j.enbuild.2013.04.015.

APPENDIX

A. TABLES

Variable	Name	Unit
N	total number of instances	-
H	total hours of operation per year	h
C_{pc}	cost of system acquisition	\$
C_p	cost of power per sys.	\$
C_s	cost of space	\$
C_m	cost of maintenance	\$
C_d	cost of deprecation	\$
C_{sc}	cost of setup and configuration	\$
C_a	cost per square meter	\$
S	required space of one sys.	m^2
SV	salvage value	\$
EL	economic life	y
TCO	total cost of ownership	\$
μ	utilization	%
P_{max}	max. power consumption per sys.	kWh
P_u	power utility charge per sys.	\$/kWh
PUE	power utilization efficiency	-

Table 3: Variables for the financial model

Variable	Name	Unit
N	total number of instances	-
PV	present value of capital	\$
PC	present capacity	\$
FV	future value of capital	\$
FC	future capacity	\$
T	time	y
Y	total number of years	y
k	annual interest rate	%
NPV	net present value	\$
NPC	net present capacity	CPU hour
TC	total processing capacity	CPU hour
TCPU	total number of CPUs	CPU
H	working hours per year	h
η	server utilization	%
A	annual investment for CPU	\$
S	asset's salvage value after Y	\$
E	asset's purchase cost	\$
E_T	disk salvage value	\$
L_T	lease payment	\$/Gb
P_T	annual profit	\$
C_T^S	annual storage operation cost	\$
C_T^{SL}	annual storage lease cost	\$
C_T^{SP}	annual storage purchase cost	\$
C_T	annual fixed cost	\$
C_0	initial investment	\$
C_S	price of server	\$
C_R	real cost of CPU hour	\$/CPU hour
C_P	C_R for purchasing resources	\$/CPU hour
C_L	C_R for leasing resources	\$/CPU hour
C_U	C_P including upgrading	\$/CPU hour
I_K	interest rate for purchase	%
I_R	interest rate for lease	%
I_F	risk free interest rate	%
δ	cost of electric util	\$/kWh
Ω	size of purchased drives	GB
ρ	difference in training needed	%
γ	disk depreciation factor	%
H_T	annual salary	\$
V_T	storage requirement	GB
R_T	disk replacement	GB
G_T	cost per Gbyte	\$/GB
K_0	lowest(T=0) cost	\$/GB
C	disk controller cost	\$
P_C	controller power requirement	kW
P_D	disk power requirement	kW
α	proportion of H_T	%
β	proportion of H_T leased	%
C_{pc}	cost of system acquisition	\$
C_p	cost of power	KW/h

Table 4: Variables for NPV and NPC model

Variable	Name	Unit
t	time	h
A_p	amortization period	h
$A_{rp}(t)$	amortizable rate parameter	-
α	money cost	%
N_{serv}	number of servers	-
VI_{ps}	cost per server	\$
VI_s	unit price of type 2 SW	\$
VI_o	unit price of type 1 SW	\$
VI_m	unit price of type 3 SW	\$
S_s, S_o, S_m	subscription factor	%
N_s	number of type 2 SW license	-
N_o	number of type 1 SW license	-
N_m	number of type 3 SW license	-
N_{switch}	annual number of network switches	-
S_{NIC}	number of NIC per VM	-
P_{NIC}	number of ports per NIC	-
P_s	price per switch	\$
N_{port}	number of network switch ports	-
C_{se}	total cost of servers	\$
C_{so}	total cost of software	\$
C_n	total cost of networking	\$
C_{su}	total cost of support/maintenance	\$
C_p	total cost of power	\$
C_c	total cost of cooling	\$
C_f	total cost of facility	\$
C_{sp}	total cost of real estate space	\$
N_l	number of administrators	-
T_{use}	time spent on utilized sys.	h
T_{idle}	time spent on idle sys.	h
R_{sal}	rating of salary average	\$/h
S_{rp}	sum: power rating of working serv.	kWh
E_s	electricity price	\$/kWh
L_s	steady-state constant	-
L	cooling load factor [14]	%
P	airflow redundancy constant [11]	-
H	inefficiency constant [11]	-
N_{rack}	number of racks	-
$VPfp$	Price of facilities per rack	\$
A_p	cost to build Cloud	$\$/m^2$
R_{SF}	area per rack	m^2
R_{SPACE}	taken space by racks	m^2
S_{SPACE}	total amount of space	m^2
W_{server}	weight of server	kg
W_{rack}	weight of rack	kg
$C_{pressure}$	pressure confronted by floor	N/m^2
R_a	annual percentage rate	%
V_{rack}	units taken per server	-
UV_{server}	units available per rack	-
VM_{dens}	number of VMs per server	-
N_{VM}	number of applied VMs	-

Table 5: Variables for the cloud cost amortization model