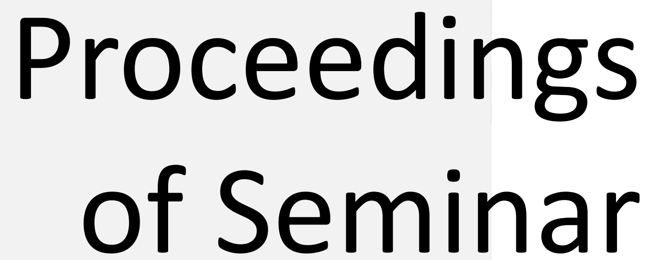


Proceedings of Seminar

Full –Scale Software Engineering

2018

Editors:
Horst Lichter
Andreas Steffens
Firdaus Harun
Konrad Fögen
Christian Plewnia
Simon Hacks
Ana Nicolaescu

SWC Software Construction

RWTH AACHEN UNIVERSITY

Features of Combinatorial Testing Tools: A Literature Review

Joshua Bonn
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
joshua.bonn@rwth-aachen.de

Konrad Fögen
RWTH Aachen University
Software Construction
Ahornstr. 55
52074 Aachen, Germany
konrad.foegen@swc.rwth-aachen.de

ABSTRACT

Combinatorial Testing (CT) can be used to cover a very large number of parameter combinations for a System Under Test (SUT) with a comparatively small number of test cases. For this purpose, several tools and algorithms have been developed to minimize the number of test cases that have to be executed to achieve a specified coverage. In addition to the minimization of the test suit, these tools offer many other features, for example, for modelling parameters or defining constraints.

While many papers have been written to introduce new algorithms or give an overview of CT research in general, previous work mostly failed to compare the features of these tools to each other. Furthermore, no overview of the distribution of said features over the different programs is available.

Therefore, we review and compare 32 tools in terms of their abilities in different feature areas concerning the modelling of input parameters. These areas are parameters, constraints, seed tests, and testing strength. In each area, the distribution of tools using certain feature implementations is given, and in the end, the tools are compared across all areas.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.4 [**Software Engineering**]: Software/Program Verifiction; D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools, Combinatorial Testing*

Keywords

Combinatorial Testing, Tools, Software Testing

1. INTRODUCTION

In studies from 2002 and 2004 Kuhn et al. showed that the interactions between 2 parameters or the specific setting of one parameter caused 50-95% of the examined software failures. The highest number of parameters that caused a software failure in a specific combination was 6 [12, 13, 16]. These findings have an impact on software quality assurance, as testers can now assume that they find nearly all failures by testing all possible combinations of 6 or fewer parameters. While this may still be a large number of tests, more would be necessary for exhaustive testing which tests all possible combinations of values.

Although most papers focus on efficient ways to generate test suites which cover all combinations of t parameters, another important area in Combinatorial Testing (CT) research is defining the Input Parameter Model (IPM). If we expect the algorithms to generate test suites tailored specifically to a software system, modelling this system precisely is important. As over 45 tools for CT exist, different approaches to model software systems can be found. Until now, no one has compared these tools in their abilities in different areas of modelling. Furthermore, to our knowledge, no paper exists which discusses the distribution of the different approaches over the tools. To close this gap in research, we look at 32 tools and classify them according to their features in the different areas.

In the next chapter, we will first examine the background of CT (section 2). After this, we will present the identified features and their implementation approaches in the tools in section 3, and the differences will be further discussed (section 4). At the end, we will summarise our findings in a conclusion (section 5).

2. BACKGROUND

In this section, we will first introduce the basics of CT. Next, we will discuss papers which relate to our research and explain our research process to discern the tools' features.

2.1 Combinatorial Testing

Combinatorial Testing was first introduced in 1985 by Mandl to test the Ada compiler [14]. In the first step for CT, the tester has to find parameters which describe a System Under Test (SUT). These parameters can be user inputs, environment descriptions, and other factors which influence the execution of the software [16]. For example, if we test a website for finding bus routes, parameters could be the browser which displays the site, the start station, and the end station. Each of these parameters can then have different numbers of values. For the browser parameter, Chrome and Firefox could be acceptable values.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SWC Seminar 2017/18 RWTH Aachen University, Germany.

The basic idea of CT is now to find test cases which cover all combinations between t parameters for a given t ($1 \leq t \leq$ number of parameters). This is also called t-way testing. The expectation is that testing all combinations for a comparatively small t can identify nearly all errors [13]. For example, if a system has 20 parameters, which can each take 2 different values, exhaustive testing would require $2^{20} = 1048576$ test cases while 6-way testing with the ACTS tool needs 375 test cases, since one test case covers multiple 6-way combinations.

2.2 Related Work

Some papers about the features of input parameter modelling for CT have already been published. One paper is "Pairwise Testing in Real World", written by Jacek Czerwonka [7]. In his paper, Czerwonka lists different options which combinatorial testing tools give to the user and analyses their usability in real-world situations. While the paper discusses many features, it focusses on PICT and does not compare the capabilities of different programs.

"Interaction Testing of Highly Configurable Systems in the Presence of Constraints", by Cohen et al. concentrates on constraints and compares the different handling of constraints in nine tools. The paper categorises the constraint handling techniques into 4 categories ranging from "none", for tools with no constraint support, through "remodel", which encompasses tools where users have to split their model into unconstrained models, to "full", for tools which support all types of constraints [5].

In [9], Grindal et al. focussed on providing a method with which testers can convert the SUT into an IPM. The approach uses features such as invalid values, constraints, and seed test cases to define complete models.

2.3 Research Process

This paper is the result of a literature review. Cronin et al. defined a literature review as "an objective, thorough summary and critical analysis of the relevant available research and non-research literature on the topic being studied" [6]. The first step of a literature review, after finding a topic, is to search for literature discussing this topic. In our case that meant finding the combinatorial testing tools. While no complete list is available, research papers often cite pairwise.org. Other tools were found with simple internet searches for papers on the topic of combinatorial testing and examining relevant internet forums. When we had gathered the list of all tools, we tested them for availability. Of the 47 we had found, 23 could be acquired directly. These tools were then tested according to their features, and the documentation has been examined. Further 9 tools had papers written about them, and we could identify the features by reviewing them. This leads to a total of 32 tools: CATS [38], AETG [4], PairTest [39], TConfig [43], AllPairs (Perl) [35], Pro-Test [26], Jenny [10], Test-Cover [21], TestVectorGenerator [1], TESTONA [23], AllPairs (Python) [31], PICT [19], rdExpert [33], OATSGen [11], ATD [22], ACTS [32], IPO-s [2], VPTAG [41], FOCUS [37], Hexawise [28], PictMaster [30], NTestCaseBuilder [15], Tcases [25], Pairwiser [29], NUnit [17], ecFeed [27], JCUnit [40], CITLAB [3], TOSCA [36], Pairwise (Ruby) [42], Pairwise (RetailMeNot) [34], JCombinatorial [18].

In the second step of a literature review, the findings have to be categorised, analysed, or otherwise evaluated. To do this, we identified similarities between the tools in the feature areas and placed related programs in the same categories. We present these categories in the first main chapter.

Lastly, we identified patterns in the distribution of features, which we will present in the discussion (section 4).

Table 1: Running Example: Parameters and Values

Host OS	Client OS	Browser	Connection	Speed
Windows	Windows	Chrome	LAN	1
Unix	Unix	Safari	WLAN	10
	Android	Edge		100
	IOS	Firefox		500
				1000

3. IDENTIFIED FEATURES

In this main chapter of the paper, we will categorise the features we found in the 32 tools. To make the process of explaining each feature clearer, we will first introduce the running example of a computer game we wish to test.

Following this, we will at first present the core features which most of the tools offer, and in this section also explain the feature categories. Next, we delve into each of the presented categories and introduce advanced features.

3.1 Running Example

Since the explanation of the features can be abstract, we introduce a running example with which we will explain them. Imagine we are at a software company, and want to release a game which groups of users can play in a Local Area Network (LAN). One of the player's devices simultaneously hosts the current game. Users can play our game on Windows, Unix, Android, or IOS in the Chrome, Safari, Edge, or Firefox browser. The host has to be either a Windows or a Unix desktop machine. The client connects to the host either via LAN-cable or WLAN, and we want to test the game at $1, 10, 100, 500$, and $1000\frac{Mbit}{s}$. We now wish to test if the game works in different host OS, client OS, browser, connection type, and connection speed combinations. Exhaustive tests would require $4^2 * 2^2 * 5 = 320$ test cases while testing all 2-way combinations reduces this number to 20 (computed with the ACTS tool).

3.2 Core Features

As with many categories of software tools, there are some core features which are present in almost every tool. Advanced tools can then expand these features with more sophisticated implementations. In this section, we will present the core features we found in most of the CT tools.

3.2.1 Parameters

As said before the first step of defining an IPM for a CT tool, is to define the parameters. In a typical CT tool, the user will give each of these parameters a name and pass the values as an enumeration of strings. This means that s/he has to convert numbers or boolean values into their text representation. For example, the user has to model the host OS and connection speed parameters in the following way: "hostOS": "Windows", "Unix"; "connection speed": "1", "10", "100", "500", "1000". As the parameters are modelled with descriptive names, it is easy for testers to understand and later change them, should the need arise. The tools which

exclusively use enumerations can be found at [38, 39, 43, 35, 26, 21, 1, 23, 33, 11, 22, 2, 41, 37, 28, 25, 29, 36, 42, 34].

3.2.2 Constraints

When modelling a system with parameters and values, it can often happen that illegal combinations are possible [7]. In our running example, we do not tell the tools that Safari is not available on Android, and no longer supported on Windows. When creating a test suite to cover all pairs of values, there will be test cases which include (Safari, Windows) or (Safari, Android). The tester cannot remove these test cases since this would also discard all the other pairs tested by these test cases. As a result, the tools have to know the constraints on the parameters during the creation of the test suite. The basic mechanism of allowing constraints is to give the user the option of specifying illegal combinations. This means that the tester has to provide all pairs, triples, or n-tuples which should not appear in the final test suite. In our example, we would simply give the pairs (Windows, Safari) and (Android, Safari). [4, 26, 10, 11, 37, 28, 29] use this method to define constraints.

3.2.3 Seed Tests

As an opposite to constraints, it can be beneficial to allow the tester to define combinations which s/he wants to have in the final test suite in any case. Such situations could arise if these combinations occur more often in practice, or if certain value combinations often cause problems. In our running example, it could be that we know that a Windows host connected to an Android device over WLAN causes software failures. If such a scenario is the case, we would wish to test the combination explicitly. One solution to the problem is to add a test case which includes this combination to the test suite generated by the CT tool, but this would result in more test cases than strictly necessary since the added combination includes some t-way combinations which are also present in other test cases. This is no great problem if the number of explicitly tested combinations is small, but with many requirements, this could cause the test suite to grow in length considerably. More test cases result in longer execution time, so this is not a desirable outcome. Some tools solve this problem by giving the user an option to specify so-called "seed tests". The tools guarantee the test cases to be in the final generated test suite and the t-way combinations which are already covered are not considered when generating the complementing test cases [44].

Furthermore, the tester can use seed tests if s/he expands parameters with new values after s/he already tested some combinations. The tools can receive these combinations as seed tests, and the CT tools will then generate test cases for the uncovered value combinations. The tools [38, 4, 39, 10, 23, 31, 19, 33, 32, 28, 30, 25, 29, 27, 3] allow the definition of seeds.

3.2.4 Testing Strength

After the user has now provided the tool of his choice with parameter, values, invalid combinations, and required combinations, the next step would be the generation of the actual test suite. For this stage, the tools need the desired testing strength. A test suite is defined to have a testing strength of t if it covers all t-way combinations [16].

All tools we examined allow the generation of test suites with strength 2, often called pairwise testing. More than half of the tools then allow the user to use a testing strength between 1 and 6, and 15 tools allow the generation of test suites with arbitrary testing strength.

Table 2: Number of Programs with Certain Types for Values

None	Enumeration	Small Set	Full Set
1	20	6	5

3.3 Parameters

In this first feature area, we will now present more advanced features for the definition of parameters. For this, we will first examine type systems and then move on to invalid and important values.

3.3.1 Types

As we showed in section 3.2.1, most of the tools use enumerations to model the parameter's values. While this may be enough if text values are given, it is not optimal for modelling numbers. Some tools recognized this and provide the user with a greater type system. Providing more types the user can choose from brings the resulting model of the SUT closer to the real software system. This has the advantage that the model is easier to understand and maintain.

When mentioning tools using more types to model parameters, it is also important to mention that one of the tools, Jenny, does not use any type at all [10]. Instead, a tester gives the number of values for each parameter and the resulting test cases contain the indices of the values. This means s/he would model the client OS parameter as "4" since it has four values (Windows, Unix, Android, IOS). If the user then would receive the test case $(1, 2, 4, 1, 3)$, s/he would have to manually map this to (host OS: Windows, client OS: Unix, browser: Firefox, connection type: WLAN, connection speed: $100\frac{Mbit}{s}$).

We divided the tools into four categories regarding their type system. The first is having no types at all (Jenny), followed by the previously presented enumerations, small type sets, and full type sets. Table 2 shows the number of tools which are in each category.

Small Type Set.

In this category we find [4, 19, 32, 30, 27, 3]. These tools offer more than enumerations, but there is some variance in how many types are available. While PICT just adds the numeric type to better model numbers, ACTS also offers boolean values as an option [32]. The tester can use the modelling of numbers in PICT to model the connection speed in our running example [20]. The different types which are available here will later be important in the options some tools give to model constraints (section 3.4).

Complete Type Set.

Here we can find libraries which testers use for automatic unit/integration tests in programming languages. One example of such a library is JCUnit, which is under development and extends the popular testing framework JUnit to allow combinatorial testing in the Java programming language [40]. It uses the factory pattern to create values for each parameter of any given type from the Java language. This allows for a modelling of the SUT that is nearer to the

Table 3: Number of Programs with Certain Important Value Feature

Defaults	Weights	Arbitrary Values
1	4	2

real system than any other tool available. Of all the tools we examined, [31, 15, 17, 40, 18] offered the complete type set of a high-level programming language.

3.3.2 Negative Values

In some cases testing whether the allowed values do not lead to a software failure is not enough. Sometimes, the tester has to evaluate the system's behaviour if s/he enters unallowed values. While it can be an option to add these illegal values to their respective parameters, this may lead to untested behaviour, as the so-called masking effect can occur. If one illegal value is present in a test case, all other valid combinations which are in the test case may not be in other test cases, which means the test suit loses coverage [7].

For avoiding this problem, some tools offer features to mark specific values as illegal. In such a case, the algorithm which creates the test suite then knows that it has to test combinations in test cases with an illegal value again in a normal test case. PICT implements this feature by allowing the user to mark one value per parameter as illegal, while ACTS allows multiple illegal values [7, 32]. [4, 19, 32, 30, 25] provide some form of illegal value handling.

3.3.3 Important Values

With the generation of combinatorial tests, minimizing the number of test cases in the test suite is always the topmost priority. In some cases, it can be good to specify secondary goals the tools should take into account. One such secondary goal could be that the tester wants to test some values more often than others [8]. In our running example, it can be the case that the game studio expects the host OS to be Windows in most cases. Therefore, it would make sense to have a greater number of test cases which use the value "Windows" for the parameter "host OS". Seven of the 32 tested tools offer a support for this feature. In the implementations the programs offer, we can observe three different approaches. Table 3 gives an overview of the number of tools which implement each of them.

Defaults.

The ACTS tool offers users to mark certain values as the "base choice" for a parameter. This effectively means that every time the value the parameter can take is one of several options, for example, since all t-way combinations with this parameter are already covered, the algorithm picks this specified value in the test case as a default value. The ACTS tool is the only one which uses this approach to the problem [32].

Weights.

While the "default value" way to solve the problem of important values works if one value is more important than the others, it does not scale if we have to consider multiple important values. If for example, we would say that 50% of all users will use Chrome, 35% Firefox, 10% Edge, and 5% Safari as the browser for our game, we would like the distribution of values in the test suite to match these percentages if possible. For this requirement, [19, 33, 37, 30] introduce the concept of weights. Every value has a default weight of 1, and higher weights specified by the user tell the tool to use these values more often [20].

Arbitrary Values.

The third approach to the "important parameter" problem is the output of special characters in the test suite. For example, AllPairs (Perl) prints a "∼" every time the value is not important to the test case. The user can then assign values s/he would like to test more than others. AllPairs (Perl) and OATSgen use this method [35, 11].

3.4 Constraints

In section 3.2.2, we already discussed the definition of invalid value combinations to define constraints in the IPM. In this part of the paper, we will present three other ways to model constraints: relations, logical expressions and programming functions. Table 4 shows the number of tools in each category.

3.4.1 Relations

In contrast to modelling constraints by giving all invalid combinations, another method is to give some values for each parameter which can validly be used in combination with each other. The tester can exclude certain combinations by providing multiple relations.

While these relations can model any constraint since there is a finite amount of values and parameters, disadvantages of using relations to model all constraints exists [4]. For example, the number of required relations can be high even if the tester defines few constraints. Secondly, relations are not easily readable for humans, as testers have to observe all relations to see if any valid combination of values is missing. And if one combination is missing, the resulting test suite may not cover all t parameter combinations. The tools which mainly use relations to define the constraints are [38, 21].

3.4.2 Logical Expressions

One step up from defining either relations or invalid combinations are logical expressions. [19, 32, 41, 37, 30, 25, 27, 3] offer them. Logical expression allow the program's user to describe the restrictions with some predefined logical connectives, and some base terms. The concrete implementations in the tools differ slightly.

Nearly all tools use a subset of the operators known from propositional calculus to connect some base terms. These base terms can always be of the form "<parameter1> = <value1>" or "<parameter1> != <value1>". In our example system the tester can now express "Android and Windows cannot use Safari" as "(browser = Safari) => !(clientOS = Windows || clientOS = Android)" [32]. Tools with more types than enumerations also allow for more complex base terms. For example, ACTS can carry out calculations between numeric parameters with elementary arithmetic expressions [32].

3.4.3 Programming functions

A small set of tools ([31, 15, 40]) uses real programming functions as constraints. These tools all are libraries which programmers can use to write unit, integration, or system tests. Since the libraries use the whole type set of the un-

Table 4: Number of Programs with certain Constraint Type

None	Relations	Invalid Combinations	Logical Expression	Programming function
12	2	7	8	3

Table 5: Number of Programs Allowing Certain Values for t

t =	1	2	3	4	5	6	n	exhaustive
	21	32	20	18	18	18	15	21

derlying language, the tester can use complex types and check them according to the required constraints for the test cases. The functions can call any sub-function and can compute complex mathematical expressions which makes this the constraint type with the most expressiveness [40].

3.5 Partial Seed Tests

In section 3.2.3, we mentioned the example of a Windows host which communicates with an Android device over WLAN. In such a scenario, three of the five parameters are set in the combination we want to cover explicitly. For tools which require full test cases as seeds, the user would have to choose the other two values arbitrarily, which may not always lead to a minimal test suite. To counter this issue, [4, 19, 28, 30, 3] allow partial test case definition, where the tester does not assign values to all parameters, and the program can then assign the other values in the generation process to archive a smaller number of test cases.

PICT implements this by leaving out the value assignments in the given seed tests, but CITLAB takes it one step further and allows so-called "test goals" definitions akin to logical expression constraints. These test goals can, for example, use the "$\vee$" operator to give the tool choices between two values [8, 20].

3.6 Testing Strength

In this main chapter's last section we will discuss the testing strength definition. As mentioned in section 3.2.4, most tools do not allow the tester to define a testing strength of over 6. Usually, testing all 6-way combinations of values is more than enough, but in the case that users want to test further combinations, some tools also allow an entirely arbitrary testing strength definition or the generation of all possible test cases. Table 5 shows the number of tools which allow t-way testing for specific values of t. "exhaustive" stands for exhaustive testing, and n means the tool allows arbitrary values for t.

3.6.1 Mixed Parameter Strengths

More important to the input parameter modelling than defining the general testing strength can be setting it for a specific parameter subset. Pairwiser, and Hexawise allow the testing strength definition per parameter [29, 28]. As a consequence, we can now say that we want a general testing strength of two, but we want the final test suite to include all 3-way combinations in which the parameter "host OS" appears.

Literature presents an important use case for using mixed parameter strength testing. Testers can use the parameter strength like weights for parameters to show which one is more important. This means the tester can define that, for example, the "client OS" causes a high percentage of failures, and should be tested in all three-way combinations [20].

3.6.2 Hierarchies

While mixed parameter strengths focusses on certain parameters, hierarchies target whole parameter groups. For these groups, the testing strength can then be overwritten. An example would be defining that the test suite should generally contain all pairwise combinations, but for the group of browser, client OS, and connection type it should also include all 3-way combinations. The difference to mixed parameter strength is that the test suite has to test combinations inside the group to the given strength, not the combination of one parameter with all other parameters. In mixed parameter strength, there would also be test cases with all 3-way combinations of host OS, client OS, and connection speed.

PICT implements hierarchies by introducing sub-models. These constructs are the combination of an arbitrary number of parameters and a given testing strength. We would have to translate our example to "{ clientOS, browser, connection } @ 3". PICT's sub-models add the benefit that the tool creates fewer parameter combinations inside one sub-model as it constructs them separately [20].

In contrast to the sub-model implementation, the maven plugin Tcases allows arbitrarily deep hierarchies using so-called VarSets. The tester defines the input parameters in an XML document where the VarSet tag allows for arbitrarily many nested tags. This enables the tester to logically model on different hierarchical levels [24]. All in all, [4, 23, 19, 32, 30, 25] implement hierarchies.

4. DISCUSSION

In this section, we will discuss and evaluate our paper's findings.

The first feature group on which we focussed was the types which parameters can take in the tools. What can be noticed here is that a high percentage of the tools uses enumerations to give the types certain values. While enumerations can be enough to enter values and use them in constraint types such as invalid combinations or relations, they restrict the user to always explicitly listing all values which are in these invalid combinations. Using types allows for a shorter constraint definition in logical expressions due to comparison operators (<, >, <=, >=), and is easier for the user. This may be the reason why nearly all tools which support logical expressions have a good type system (e.g. PICT, ACTS). Tcases tries to obviate this problem by introducing properties which the user can assign to multiple values of one parameter and ease the constraint formulation, but it introduces a certain overhead of defining these properties each time.

Another observation we made, is that generally, the tools with the most features are freeware. For example, the program with the most features was PCIT. It offers a small type system, weights, negative values, the use of logical expressions for constraints, full and partial seed tests, and ar-

bitrary strength testing with hierarchy definitions. Behind PICT we find ecFeed and ACTS which are freely available, too. Commercial products often have an average number of features. On the other side, most of their features are in the lowest category inside a specific feature area. For example, none of the commercial programs which are available right now allows for any type other than enumerations, and logical expressions are also not supported.

The programs with the least features are programs which researchers presented in papers as a proof of concept for a new algorithm. They are not intended to be loaded with features.

All in all, we can say that the feature distribution across the tools is pretty uneven. While many programs support up to five individual features, the number of programs diminished as the number of features rises.

Another important point which we can see in the data is that twelve programs only offer up to 2-way testing. This can be dangerous as the percentage of errors found with pairwise testing is 50-95% which may seem like a high number, but it does not sound good to say "This software has 5-50% of all bugs left". At the same time, pairwise testing can still be better than testing combinations without any strategy.

5. CONCLUSION

Combinatorial Testing is an important strategy to find errors in software systems with a relatively small amount of test cases. Many tools are available to minimize the test suite, but there are immense differences in modelling the input parameter models for these programs. In this paper, we analysed different approaches in four feature areas, the parameters themselves, constraints, seed tests, and testing strength.

As section 3 has shown, the differences in the areas are immense, with the options of types for the parameters ranging from none or enumerations to complete type sets found in programming languages. We identified that in most categories the tools tend to stick to the more basic implementations with 19 tools using enumerations as a type option, and only 5 offering partial test cases.

In section 4 we have found that the programs with the most features are freeware, and commercial tools often lack a good type and constraint system. Furthermore, we identified the risk of programs offering nothing more than 2-way testing.

5.1 Outlook

Future work could expand the definition of hierarchies in parameters. Until now, 6 tools support this feature, and in most the user can define hierarchies of height one. It would be interesting to see what is possible in this field.

Furthermore, based on this paper, we can identify some future area of research for CT. For example, we did not examine the exact implementation of each feature in the tools and did not focus on usability. In this area, further research could be required.

Furthermore, one could analyse the tools according to their performance in the feature areas. While eight tools offer logical expressions, no one has analysed how big the performance deficit of evaluating them can be.

6. REFERENCES

[1] J. Arshem and P. J. Schroeder. Test vector generator. `https://sourceforge.net/projects/tvg/`, 2004. Retrieved October 19, 2017.

[2] A. Calvagna and A. Gargantini. Ipo-s: Incremental generation of combinatorial interaction test data based on symmetries of covering arrays. In *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*, ICSTW '09, pages 10–18, Washington, DC, USA, 2009. IEEE Computer Society.

[3] A. Calvagna, A. Gargantini, and P. Vavassori. ATD. `http://atyourside.pt/downloads/our_downloadcenter.html`. Retrieved November 2, 2017.

[4] 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 Trans. Softw. Eng.*, 23(7):437–444, July 1997.

[5] M. B. Cohen, M. B. Dwyer, and J. Shi. Interaction testing of highly-configurable systems in the presence of constraints. In *Proceedings of the 2007 international symposium on Software testing and analysis*, pages 129–139. ACM, 2007.

[6] P. Cronin, F. Ryan, and M. Coughlan. Undertaking a literature review: a step-by-step approach. *British journal of nursing*, 17(1):38–43, 2008.

[7] J. Czerwonka. Pairwise testing in real world. In *24th Pacific Northwest Software Quality Conference*, volume 200, 2006.

[8] A. Gargantini and P. Vavassori. Citlab: a laboratory for combinatorial interaction testing. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 559–568. IEEE, 2012.

[9] M. Grindal and J. Offutt. Input parameter modeling for combination strategies. In *Proceedings of the 25th Conference on IASTED International Multi-Conference: Software Engineering*, SE'07, pages 255–260, Anaheim, CA, USA, 2007. ACTA Press.

[10] B. Jenkins. Jenny. `http://burtleburtle.net/bob/math/jenny.html`, 2003. Retrieved October 15, 2017.

[11] R. Krishnan, S. M. Krishna, and P. S. Nandhan. Combinatorial testing: Learnings from our experience. *SIGSOFT Softw. Eng. Notes*, 32(3):1–8, May 2007.

[12] D. R. Kuhn and M. J. Reilly. An investigation of the applicability of design of experiments to software testing. In *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, pages 91–95. IEEE, 2002.

[13] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, Jr. Software fault interactions and implications for software testing. *IEEE Trans. Softw. Eng.*, 30(6):418–421, June 2004.

[14] R. Mandl. Orthogonal latin squares: An application of experiment design to compiler testing. *Commun. ACM*, 28(10):1054–1058, Oct. 1985.

[15] G. Murphy. NTestCaseBuilder. `https://github.com/sageserpent-open/NTestCaseBuilder`, 2014.

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

2011.
[17] C. Poole and R. Prouse. NUnit. http://nunit.org/docs/2.6.4/pairwise.html, 2009.
[18] J. J. Reeder. JCombinatorial. https://github.com/jeremyjohnreeder/jcombinatorial, 2011. Retrieved November 5, 2017.
[19] Microsoft Corporation. PICT. https://github.com/microsoft/pict, 2004. Retrieved October 15, 2017.
[20] Microsoft Corporation. PICT Documentation. https://github.com/Microsoft/pict/blob/master/doc/pict.md, 2017. Retrieved October 15, 2017.
[21] L. TestCover.com. Testcover. http://www.testcover.com/index.php, 2003. Retrieved October 15, 2017.
[22] AtYourSide Consulting. ATD. http://atyourside.pt/downloads/our_downloadcenter.html. Retrieved November 4, 2017.
[23] Berner and Mattner Systemtechnik GmbH. TESTONA. http://testona.net, 2017. Retrieved October 26, 2017.
[24] Cornuntum Project. Tcases Documentation. http://www.cornutum.org/tcases/docs/Tcases-Guide.htm, 2016. Retrieved November 26, 2017.
[25] Cornutum Project. TCases. https://github.com/cornutum/tcases, 2016.
[26] Digial Computations, Inc. Pro-Test. http://www.sigmazone.com/protest.html, 2002. Retrieved October 17, 2017.
[27] ecFeed AS. ecFeed. http://ecfeed.com/, 2017. Retrieved October 16, 2017.
[28] Hexawise Inc. Hexawise. https://hexawise.com.
[29] Inductive AS. Pairwiser. https://inductive.no/pairwiser/, 2017. Retrieved October 15, 2017.
[30] IWATSU System & Software. PictMaster. https://osdn.net/projects/pictmaster/, 2013. Retrieved October 15, 2017.
[31] MetaCommunications. AllPairs. https://github.com/bayandin/allpairs, 2009. Retrieved October 15, 2017.
[32] NIST and University of Texas at Arlington. ACTS. https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software/downloadable-tools#acts, 2016. Retrieved October 16, 2017.
[33] Phadke Associates, Inc. rdExpert Test Planning. http://phadkeassociates.com/index_files/rdexperttestplanning.htm. Retrieved October 19, 2017.
[34] RetailMeNot, Inc. Pairwise. https://github.com/RetailMeNot/pairwise, 2014. Retrieved November 4, 2017.
[35] Satisfice, Inc. ALLPAIRS. http://www.satisfice.com/tools.shtml. Retrieved October 15, 2017.
[36] Tricentis GmbH. Tosca. https://www.tricentis.com/de/tricentis-tosca-testsuite/, 2016. Retrieved November 7, 2017.
[37] I. Segall, R. Tzoref-Brill, and E. Farchi. Using binary decision diagrams for combinatorial test design. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ISSTA '11, pages 254–264, New York, NY, USA, 2011. ACM.
[38] G. Sherwood. Effective testing of factor combinations. In *Proc. of the Third Int. Conf. on Softw. Test., Anal., and Rev.(STAR94), Washington, DC, Software Quality Eng*, 1994.
[39] K. C. Tai and Y. Lie. A test generation strategy for pairwise testing. *IEEE Trans. Softw. Eng.*, 28(1):109–111, Jan. 2002.
[40] H. Ukai and X. Qu. JCUnit. https://github.com/dakusui/jcunit/, 2017. Retrieved October 15, 2017.
[41] R. Vanderwall. VPTag. https://sourceforge.net/projects/vptag/, 2010. Retrieved October 15, 2017.
[42] J. Wilk. Pairwise. https://github.com/josephwilk/pairwise, 2016. Retrieved November 4, 2017.
[43] A. W. Williams. Determination of test configurations for pair-wise interaction coverage. In *Proceedings of the IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems: Tools and Techniques*, TestCom '00, pages 59–74, Deventer, The Netherlands, 2000. Kluwer, B.V.
[44] C. Yilmaz, S. Fouche, M. B. Cohen, A. Porter, G. Demiroz, and U. Koc. Moving forward with combinatorial interaction testing. *Computer*, 47(2):37–45, 2014.

Metrics in Agile Projects - Does that matter?

Matthias Hansen
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
matthias.hansen@rwth-aachen.de

Horst Lichter
RWTH Aachen University
Software Construction
Ahornstr. 55
52074 Aachen, Germany
lichter@swc.rwth-aachen.de

ABSTRACT

Managing software development is a difficult problem and the development process usually does not go as planned. Since the goal of a software development project is the collaborative creation of a technical system, measuring relevant attributes of both the development process and the emerging product can be expected to lead to better outcomes. To this end several metrics and applications of metrics have been proposed in the literature. Agile process models have gained much traction in recent decades. They emphasize short feedback cycles, incremental planning and continuous improvement in relatively small teams. However, it is not clear how metrics can and should be applied to teams using these process models. This paper addresses this question by formulating common goals of metric usage and proposing methods to achieve these goals in agile projects.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management

Keywords

Software Engineering, Agile Methodologies, Metrics

1. INTRODUCTION

Agile process models have seen widespread adoption in recent years. Their emphasis on short feedback cycles both for the continuous self-improvement of development teams and the direction of the software they develop has shown promising increases in successful project delivery[7]. In particular, this is achieved by two practices. Firstly feedback from the customer is continuously incorporated to ensure the software meets their needs precisely. Secondly, the team continuously reflects on their process and improves their methodology.

In the context of software development, a metric is a measurement of some aspect of either the software that is developed or the process used to develop it. In the past, many metrics have been developed to this end. Agile process models have also been accompanied by a new set of metrics and visualizations. However, it is unclear if this obsoletes or recontextualizes previous work on metrics. The goal of this paper is thus to give concrete recommendations on how metrics should be used in agile projects as opposed to non-agile projects. To achieve this goal, we will compare some of the practices commonly used in both approaches.

The rest of the paper is organized as follows: Section 2 gives an overview of the most widespread agile process models. Section 3 discusses the reasons why metrics might be used. Section 4 introduces and defines several metrics referred to later. Sections 5 and 6 detail the use of metrics with respect to work estimation and progress measurement as well as continuous improvement. Finally, section 7 concludes the paper.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SWC Seminar 2017/18 RWTH Aachen University, Germany.

2. AGILE PROCESS MODELS

Several agile process models have been developed and are in use. Most widespread among them appear to be Scrum, Kanban and Extreme Programming (XP) [7]. This chapter will give a short overview of their core principles and their stipulations with regard to the use of metrics as outlined in their seminal texts[1, 8, 2].

Scrum understands itself as a partially fixed process that should be adopted as-is and then modified in certain predefined ways. It prescribes that software be developed incrementally in self-organized small teams. Each increment should be developed in one iteration of at most one calendar month–a "Sprint"–and each such iteration is preceded by a planning meeting to define goals (Sprint Planning) and succeeded by a demonstration and feedback meeting with the customer (Sprint Review) and a reflection meeting, in which the team should try to improve on their process (Sprint Retrospective). Additionally, the team should meet very briefly every day to monitor its own progress (Daily Scrum). Teams should be accompanied by one requirements specialist (Product Owner) and one process coach (Scrum Master). The Product Owner should maintain an ordered list of requirements (Product Backlog). It should be clearly defined when a work item is considered to be completed (Definition of Done) and developers should have no fixed specializations within the team.

Specifically with regard to metrics, the Scrum Guide prescribes that *total work remaining* both overall and towards the end of the current iteration, should be measured regularly. The developers should use this to monitor their progress towards the Sprint Goal. The product owner should

communicate this *total work remaining* quantity to the stakeholders.

Extreme programming (XP) is a set of practices that enables frequent adaptation to new customer requirements. XP mandates that there is a customer on-site in the same office as the development team who can be consulted about the requirements of the software in construction. Software is developed iteratively, taking into account the customer's wishes. In these respects, XP is very similar to Scrum, but somewhat more vague. However, unlike Scrum, XP is a process model that is specific to software development. Thus, concrete prescriptions with respect to programming practice are given: code should be tested very extensively, continuously integrated and programmed in pairs of developers to ensure correctness. With respect to design, there should not be a comprehensive model of the software before it is built. Instead the system's design should grow with the requirements and be in a constant flow.

On metrics, XP mandates measuring of *velocity*, the number of requirements, formulated as user stories, that are delivered on in each iteration. This mostly corresponds to the changes in the quantity of *total work remaining* that is measured in Scrum. Furthermore, acceptance test results should be published to the team to give them a measure of their progress from the customer's point of view.

Kanban is less of a fixed process and more of a process improvement toolset. Inspired by a production technique which had been in use at Toyota Motor Corporation[1], it was first applied specifically for knowledge work in the 2000s. Like Scrum, it makes certain prescriptions, but these are generally less concrete in comparison. For instance, while Scrum sets very clear explicit rules with regard to e.g. regular meetings that should be held, Kanban only states that rules should be made clear and explicit. Similarly, there is a stipulation to visualize and control the flow of work and to use metrics and models to inspire change from within the organization, and some of them are noted as particularly useful. However, no comprehensive list of metrics to use is given.

Where particular metrics and models are recommended, there is an emphasis on queueing theory and probabilistic models. Usually, the object of measurement is understood as an ongoing, stochastic process. *Lead time*, the duration for which an item is in the process and *throughput*, the speed at which items pass through the process should be measured. Both are related to *work in progress* through Little's law ($\text{Lead Time} = \frac{\text{Work In Progress}}{\text{Throughput}}$), a mathematical result from queueing theory. With respect to reporting "we are less interested in reporting on whether a project is 'on-time' or whether a specific plan is being followed. What's important is to show: that the Kanban system is predictable and is operating as designed, that the organization exhibits business agility, that there is a focus on flow and that there is clear development of continuous improvement." [1].

For example, a software development team might know that its process currently consists of three stages: Design, Development and Testing. They might decide to use a whiteboard with columns for each stage, where each work item wanders through the stages. This might then reveal that lots of work items "queue up" behind the development column. This is an indication that items move through the previous stages faster than the Testing stage. The team could then decide to limit how many work items can be in the Design and Development stage simultaneously. Because work in the limited stages cannot progress so long as the Testing stage is overburdened, this forces the team to try to accelerate the Testing stage. They might then decide to devote more resources to Testing or they might look into methods to develop Software that can be can be tested more easily or has fewer defects. By forcing developers to view the flow of items through the process as a whole instead of just particular stages, bottlenecks can be alleviated and the flow of items through the system accelerated.

3. GOALS OF METRICS USAGE

Use of metrics can only be evaluated against the goal that is to be achieved. Generally, metrics are used for several purposes:

One use of metrics is *work estimation and progress measurement*. Knowing the amount of work necessary for the completion of a project allows for calculating the cost of the project, which is necessary to make the economic decision whether it is worth it to continue work on a project or even whether to start it in the first place. Knowing the amount of work required also allows for making decisions on the number of engineers to assign. Ideally, one would like to always be able forecast specifically at what date a project will be completed for a given number of assigned engineers.

There are several factors inhibiting this ability: Engineer availability, technical difficulty and even the desired outcome of the project are not certain ahead of time. In addition, these are generally planned by an employee who may not be capable of doing so objectively, either because of or lack of information or because they are subject to cognitive biases [6] or simply because they are incentivised to be inaccurate, for example in a price-to-win dilemma. Even with perfectly aligned incentives, an estimating employee might still give inaccurate predictions simply due to a bias such as loss aversion.

Another use of metrics is for *continuous self-improvement* of teams. As discussed before, most agile process models explicitly endorse the use of metrics to this end. However, it is not clear how to encourage a team to do this.

Some attempts are also being made to use metrics to quantify *employee performance* to drive personnel decisions. This has been widely established as ineffective as it is not possible to capture the actual goals of a company (usually financial) accurately in quantitative metrics. This can lead to perverse incentives with potentially devastating effects.

In particular with respect to software engineering, it is impossible to measure the effect of any programmer's performance on the financials of their company, because even small oversights can lead to defects with unbounded costs several years afterwards. Thus, any such metric that aims to be informative can only be calculated accurately after it has ceased to be of interest.

4. METRICS

In this section, we will define some relevant terminology that will be used throughout the paper.

Agile process models generally slice development work into small pieces. These are referred to by various terms: *work items*, *features*, *tasks*, *tickets*, *user stories*, *epics* and so forth. The reason why the terminology is varied here is that these terms represent different points of view and are

defined differently by different teams. For instance the term *feature* emphasizes that the content should be formulated from a customer's point of view. Using the term *user story* is additionally associated with the format of "as a [...] I want [...] in order to [...]", but not every team follows this convention. Teams might also use these terms in a hierarchical relationship and define e.g. that a *user story* is composed of multiple *tasks*. For the purpose of this paper, we will only use the term *work item*. This will sometimes seem odd: One would not negotiate with a customer which *work items* a software system has to consist of, but this is the easiest way to deal with inconsistent terminology. Furthermore, we will assume that there is no hierarchical relationship between work items.

The Scrum guide mandates tracking *total work remaining*. This is simply the count of *work items* that are currently formulated in some form or another and have neither been completed nor definitively rejected.

Velocity is a metric that is commonly associated with XP. It has two different definitions: One is

$$\text{Velocity} = \frac{\#\text{ work items completed}}{\text{Sprint}}$$

the other is

$$\text{Velocity} = \frac{\#\text{ work items completed} - \#\text{ work items added}}{\text{Sprint}}$$

Notably, the first definition only encompasses the team's speed at delivering items to initial satisfaction, whereas the second definition also includes the customer's and product owner's creativity with respect to new *work items* and the amount of work items that are created as a result of e.g. bugs or previously underspecified behavior. We will refer to the first definition.

In Kanban, *Throughput* is often used instead of Velocity, since Kanban does not have a concept of Sprints. Essentially,

$$\text{Throughput} = \frac{\#\text{ work items completed}}{\text{time unit}}$$

where the time unit can be chosen arbitrarily. In general, we will denote the used time unit where relevant by writing for example *throughput/week*

Lead time is a metric from the Kanban process model: The *lead time* of a work item is the elapsed time from the moment where the item enters the process to the moment where it leaves the process. Typically these *process boundaries* are the moment where the team *commits* to delivering the work item and the moment where the work item is handed off either to the customer or another team.

Cycle time is similar to lead time in that it measures a time to completion. However, *cycle time* is the elapsed time from *actively starting work* on an item to delivering it.

Work In Progress (WIP) is the number of work items that are actively being worked on at any point in time. In general, this metric is specific to Kanban and is there usually limited by a policy set by the team.

In addition, we define a metric we call *Rate of Change* (RoC). This metric refers to the rate at which new work items are added and removed by the customer. It is computed as

$$\text{Rate Of Change} = \frac{\#\text{ work items added}}{\text{time unit}}$$

where the time unit can be chosen arbitrarily. This metric does not measure anything about the development process itself. Instead, it measures a characteristic of the customer: How uncertain are they about the scope of the project. If they are very uncertain, they may add many new work items regularly. A negative RoC means that the customer is uncertain "in the other direction", i.e. they initially assumed the project to have a bigger scope than was necessary. For example, an accounting firm might commission a software that replaces an old solution that they have been using for several decades. They initially assume that the new software has to have the same exact functionality as the old solution. During the project, it is discovered that large parts of the functionality of the old solution were only there to satisfy certain regulatory requirements. These requirements are not in effect anymore, so they don't have to be implemented in the new software. Similarly, it might be discovered that certain features of the old solution were actually never used and also don't have to be implemented again. In this way, the RoC of this project might become negative. We stress, however, that this is not the usual case and that for most projects the RoC will be positive.

5. WORK ESTIMATION AND PROGRESS MEASUREMENT

Progress in projects is traditionally often measured in terms of creating fixed milestones of functionality that should be available at preset dates. This approach is incompatible with the goals of agile methodologies:

If a team commits at the beginning of a project to delivering a certain milestone at a certain date then all of the associated functionality can never be removed from the scope of the project even if it turns out that the functionality is not actually useful for the customer. Similarly, fixed milestones make it difficult to add any functionality to the scope of the project. After all, a development team that is evaluated by whether they meet milestones is highly incentivised to sacrifice all other functionality in favor of what is explicitly mentioned in the milestone. One could, of course attempt to solve this by adjusting the dates and scopes of the milestones. However, then milestones cease to be a proper measure of progress. If one can remove and add functionality to a milestone and adjust the date accordingly, then it is not clear anymore whether the team is actually on schedule or whether the milestones have just been adjusted to simulate being on schedule.

To cope with this, agile process models have some associated metrics of their own for the purpose of progress measurement as mentioned in previous sections. For instance, the prescription of Scrum to continuously quantify *total work remaining* makes it extremely simple to calculate Velocity (in one of the above definitions) of the development team. While not directly prescribed in the Scrum Guide, there is a tendency to attach an estimate of required effort to each work item to handle differently sized work items. Typically this is given in some fictional unit such as *Story Points* or *Dings* which is not meant to be compared to other teams or any temporal quantity. Work items are then weighted by their Story Points to calculate Velocity. The sequence number of the Sprint in which a project will be completed can then be calculated as

$$\text{current sprint} + \frac{\text{Velocity} - \text{RoC/Sprint}}{\text{story points of total work remaining}}$$

.

In Kanban, this deterministic model is replaced with a probabilistic one[10], which in our case would take into account all empirical *cycle times*, *WIP*, *RoC* and the number of remaining work items. The Monte Carlo method that is used to achieve this then yields not a single estimated completion date, but a range of completion dates with attached confidences. This requires measuring *lead times* on a *per work item* basis, but leads to a more precise result. Algorithm 1 shows how this method works, when assuming constant WIP and constant RoC over the whole development process. The output of the algorithm is an array that can be interpreted as follows: For $n = 200$, $res[180]$ is a date, before which the project will be finished with a probability of $\frac{180}{200} = 0.9$.

```
Data: int wip, int rocPerMonth, TimeDelta[]
      empiricalLeadTimes, int remainingItems, int n
Result: Date[] res, a sorted array of possible
        completion dates
Date[] currentItems = new Date[wip]
setAll(currentItems,today)
int runningMonth = today.month
for i ← 0 to n do
    while remainingItems > 0 do
        ind := argmin(currentItems)
        currentItems[ind] := currentItems[ind] +
         drawUniform(empiricalLeadTimes)
        remainingItems := remainingItems - 1
        if currentMonth ≠ max(currentItems).month
         then
            currentMonth :=
             max(currentItems).month
            remainingItems := remainingItems +
             rocPerMonth
        end
    end
    res[i] := max(remainingItems)
end
res := sort(res)
```

Algorithm 1: Monte Carlo Algorithm for Obtaining Project Estimates

However, it is important to note that both of these approaches are only directly applicable to projects that are already ongoing because empirical data for e.g. velocity, cycle times and the number of remaining items is necessary. This is by design. The Scrum Guide states: "In complex environments, what will happen is unknown. Only what has already happened may be used for forward-looking decision-making"[8]. Despite this, gaining even vague foreknowledge of project durations before projects are started is immensely valuable from a business point of view.

Traditional projects tend to solve this problem by trying to elicit complete requirements from a customer and precisely estimate how long fulfilling these requirements will take using a model such as function points[9] or Cocomo[4] to map requirements to effort estimates. This is not dissimilar to the way agile projects might use story points, the main difference being that these models are more well-defined. In fact, in e.g. Scrum it is not very problematic to use function points instead of story points and calculate velocity on that basis. However, the same problem as before occurs in this case: One has to receive reasonably complete requirements for the final product and empirical data on how fast the team is at delivering this functionality. In addition, these approaches such as function points and Cocomo usually do not take into account that the scope of the project will change over its duration.

To deal with the problem of uncertain scope, one possible approach would be to start each project acknowledging that there is currently insufficient empirical evidence to indicate how long the project will take. Instead, the forecase is built and adjusted once such evidence begins to accumulate. The business decision of whether to pursue the project would then have to be made *after* the project has started, which means that developer time may be wasted on a project that turns out to be unprofitable. Working in this way also requires not only an agile development team but an agile decision making process which few companies have. Specifically, decision makers would have to carefully and continuously evaluate the expected cost and utility of a project and, should it become unprofitable, be able to reallocate the team towards a different project on relatively short notice. The advantage of doing it in this way is that one can then reliably know that estimates have a concrete basis and that the current projects that are being pursued are the most profitable ones.

Another approach to solving this problem would be to forecast agile projects by collecting a minimum viable set of work items for the completion of the project with the prior understanding that these can and will change in the future. Then the duration of the project can be estimated using the same methods used for ongoing projects as described above. Parameters like Velocity and RoC would then have to be estimated based on data from past projects.

For this to succeed, however, work items from past projects must be comparable in size. To a certain extent a process like Scrum guarantees this by prescribing that every work item should be completable within a Sprint of at most one month. However, ideally, work items should be much smaller than that. Care must be taken to ensure that RoC estimation reflects the initial well-definedness of the particular project. Similarly, Velocity should be estimated with respect to the ability and size of the team.

5.1 Example Calculation

Company A wants to decide whether to accept an offer by company B to develop a software product for use in conjunction with a car that company B plans to manufacture. To estimate whether they can complete the project in time for the delivery of the first batch of cars, both companies have a planning session in which they prepare a set of 50 high-level work items corresponding to features that the final product should have. They do this with the agreement that this set of work items accurately represents the *current* assumptions that company B has about which features the product needs to be shippable. Company B will get an accurate progress report about the current state of development every three weeks. They can then decide which work items will get worked on next. In doing so, they can also choose work items that were not part of the initial set of work items that was agreed upon.

How can company A decide with reasonable certainty whether their agile development team can deliver a satisfactory result by the time company B delivers the first cars to their

customers?

First, company A needs to estimate a suitable RoC. To this end they take into account:

- This is their second cooperation with company B. In the previous project, with similarly sized work items, a RoC of 4 work items every three weeks was measured. That is, on average, the set of work items increased by 4 every week.
- Company B has never delivered a software product of this kind to its customers before. They are likely very uncertain about the desired qualities of the product.

Thus, a RoC of 8 work items can be estimated. Similarly, by considering the size of its team and empirical data for this team on projects with similarly sized work items, a throughput of 13 work items per three-week-interval is estimated. To estimate time-to-completion, one can now solve the equation $\lceil(\text{Throughput} - \text{RoC})x + 50\rceil = 0$. In this example, the project will be completed to the satisfaction of company B in 10 three-week-intervals. This calculation can be repeated at any time during the execution of the project, using empirical data for *RoC*, *Throughput* and the number of outstanding work items to give up-to-date estimates.

5.2 Discussion

As the example has shown, allowing for changes to the scope of the project can dramatically affect project duration if the rate of changes is not close to zero. Therefore it is important to include it in initial estimates.

In practice, there are many additional factors that have to be taken into account: Company B has an incentive to keep the initial set of work items small and the work items themselves large. Thus company A has to initially push for many, relatively small work items.

In addition, the velocity of a team will depend on factors such as the skill of the programmers and their knowledge of the domain as well as their ability to work productively as a team. To work with data on both velocity and RoC from old projects, work items need to have a comparable size across projects. If necessary, detailed data from old projects can be used to convert between them. However, one should keep in mind that this will never work perfectly if the teams for these projects were different, because velocity as a measure of productivity does not scale linearly with the number of programmers[5] as might be assumed.

We have omitted the Monte-Carlo Method above in this example for brevity. Applying that method generally yields a more detailed model but is more complicated to apply.

6. CONTINUOUS IMPROVEMENT

The Agile Manifesto explicitly prescribes the principle that teams should regularly meet to reflect on their effectiveness and make changes in their methodology accordingly. With respect to metrics, this means that every single team even in a large organization must be given the permission and the means to apply metrics to their processes at their own discretion and set process policies with regard to this.

For instance, a team might notice an increase in defects and failure load and therefore create a policy mandating full statement coverage for every module affected by a work item, before the work item can ever be marked as completed. However, this team should also be permitted to break even company wide policies if they are deemed unhelpful by experience. For example, if company policy dictates that there is an upper limit on the number of lines of code of any program, but a team finds that this constraint leads to an overly complex, poorly readable code style, then they should be able to ignore the policy.

Note that this does not mean that teams can work however they want. They are still monitored with regard to metrics like velocity and they need to produce rational justification and ideally empirical data in support of their policies. Furthermore, in highly regulated industries, certain policies are dictated by law. These have to be accepted as constraints on the process.

The reasons why teams should be given this high degree of autonomy over their use of metrics are that only they understand their process well enough to choose relevant metrics to track and make changes based on them. Furthermore every project has its own profile of requirements. For instance, in Kanban methodology, it is possible to mark certain work items as "expedited" and prioritize them over all other items. Depending on the other policies around the number of items to be worked on, the simple act of allowing expedited items significantly increases the variability of *lead times* for work items, thereby making it harder to predict when each item will be finished. However, in some projects, expediting work items may be necessary to deliver the maximum amount of business value. Thus, no policy with regard to this can ever set organization-wide for any organization. It must be possible for each team to make informed policy decisions on their own.

However, teams having the *autonomy* to make decisions on their use of metrics is only a necessary precondition. How does one concretely move a team into the direction of using metrics without pushing them? Several approaches to this problem have been suggested: The main approach is to provide every team with a coach who is knowledgeable about metrics and can introduce them to the team. The team can then slowly learn to implement and interpret metrics and make their own informed decisions. In e.g. the context of Scrum, this is one of the responsibilities of a scrum master. A similar method is to give teams a "budget" that can be spent on metrics. This could take the shape of a concrete financial budget or a ruling such as "up to $\frac{1}{10}$ of the work items in each increment can be spent on improving and interpreting metrics". Note that this budget does not need to be particularly large. The purpose of this is primarily to communicate the *permission* to work on metrics independently without direct orders and only secondarily that of allocating resources.

Based on this, we can give the concrete recommendation that organization-wide constraints on the usage of metrics should be lifted wherever possible. It must be possible for each team to use metrics as they see fit. This requires a certain amount of trust in the teams' ability to make reasonable decisions. Furthermore, additional steps as outlined above can be taken to encourage the practical use of metrics by the development teams. These recommendations, especially the first one, are in line with the fifth principle from the agile manifesto: "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done." [3]

7. CONCLUSIONS

We have seen the two main types of change that agile process models encourage: Change of goals and change of methodology. Our use of metrics should be able to cope with both of them.

With respect to changes and uncertainty in goals, we have seen that it is necessary to account for them when forecasting and estimating. A project's completion date is determined not only by the current perception of its scope, but also by the change in this perception. Care has to be taken to estimate properties of the project such as this change correctly from past data. Ultimately, all plans will have to be adapted to a changing reality.

Similarly, change of methodology in agile projects is bottom-up and constant. Thus, an organizational approach to metrics has to be used that ensures that teams can govern themselves. In accordance with the agile manifesto, it must be possible for a team to set their own policies with regard to their use of metrics.

8. REFERENCES

[1] D. J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business.* Blue Hole Press, 2010.

[2] K. Beck. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, 2000.

[3] K. Beck and M. Beedle. *Manifesto for agile software development, 2001.* <http://agilemanifesto.org> 2001. Accessed on 22.12.2017.

[4] B. W. Boehm, R. Madachy, B. Steece, et al. *Software Cost Estimation with Cocomo II.* Prentice Hall PTR, 2000.

[5] F. P. Brooks Jr. *The Mythical Man-Month.* Addison-Wesley Longman Publishing Co., Inc., 1995.

[6] D. Kahneman. *Thinking, fast and slow.* Macmillan, 2011.

[7] A. Komus. Abschlussbericht: Status quo agile 2016/2017: 3. *Studie über Erfolg und Anwendungsformen von agilen Methoden*, 2017.

[8] K. Schwaber and J. Sutherland. *The Scrum Guide.* <http://www.scrum.org/resources/scrum-guide>, Accessed on 21.11.2017.

[9] C. R. Symons. Function point analysis: Difficulties and improvements. *IEEE transactions on software engineering*, 14(1):2–11, 1988.

[10] D. S. Vacanti. *Actionable Agile Metrics for Predictability: An Introduction.* Leanpub, 2015.

A Review on Automated Refactoring Tools

Jonas Hollm
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
jonas.hollm@rwth-aachen.de

Muhammad Firdaus Harun
RWTH Aachen University
Software Construction
Ahornstr. 55
52074 Aachen, Germany
harun@swc.rwth-aachen.de

ABSTRACT

Software systems decay in quality as they grow over time. Refactoring denotes the task of improving the structure of software without changing its functionality. Since refactoring is often time consuming, it is insufficiently applied in practice. Automated refactoring can ease the work of developers as well as save companies money.

Common techniques for automated refactoring are combinatorial optimization and genetic programming. The essence of both techniques is to randomly restructure software, followed by checks for improvements with the help of certain software metrics.

It is hard to decide for an appropriate automated refactoring solution since, besides [8], there is not much research on the comparison of such tools. This paper introduces and compares five tools which are capable of refactoring software automatically. The tools have been chosen because they are one of the few automated refactoring tools with published and comprehensive results.

The examination and comparison of the tools is conducted regarding six attributes: Refactoring goal, refactoring technique, input/output, methodology, experimental design and results.

Finally, this paper makes a recommendation on what tool to use.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.3 [**Software Engineering**]: Coding Tools and Techniques—*automated refactoring, code smell removal*

Keywords

Automated Refactoring, Code Smell Removal, Combinatorial Optimization, Genetic Programming, Software Metrics

1. INTRODUCTION

When non-trivial software systems are developed, it is almost impossible to know all requirements up front. The traditional waterfall model starts from the premise that this is the case, but therefore it has been refined or replaced in practice. Software evolves over time, meaning new features are added and old ones are removed or adjusted. Maintenance is crucial for software in order to stay operational. Additionally, it is also very difficult to get a design right even if all requirements were known. For that reason, software has to be partly or fully restructured after its initial release to improve maintainability. Often developers know that a solution they implement has avoidable downsides, but they postpone the task of rewriting their solution. Mostly this is due to time constraints that have to be fulfilled and developing a high-quality solution takes a lot of time. The problem is that it becomes the more difficult to restructure a solution the more time passes by.

The solution is to make use of automated refactoring tools. In the process of refactoring the structure of software is altered, but at the same time its functionality is preserved. While a developer needs to have deep knowledge of a program in order to refactor it, automated tools can work unsupervised, which saves a lot of resources. Refactoring can be applied immediately after a certain piece of software has been developed to improve the initial design. Another use case is to automatically refactor legacy systems so that they become more maintainable.

There are many tools which detect code smells and antipatterns, but far fewer tools exist that can correct them properly and even fewer that correct software fully automated. Most of the refactoring tools work on object-oriented code. In this context, a frequent objective is to incorporate appropriate design patterns. Also less sophisticated operations such as renaming methods is referred to as refactoring, but this can usually not be done automatically.

Automatic refactoring is a hard task, because if something is a flaw or not often depends on the context. However, software should comply with concepts such as high cohesion and low coupling, so refactoring tools can aim for these goals. Also, an automated tool does not have the sense of elegance developers might have. For example, a tool could refactor software in a way that leads to less code duplication but at the same time reduces the understandability of the source code.

To help with the decision of what tool to use, this paper reviews five automated refactoring tools: A-CMA, TrueRefactor, Dearthóir, REMODEL and DPT. These tools have been selected because most of the other papers that developed au-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SWC Seminar 2017/18 RWTH Aachen University, Germany.

tomated refactoring tools are not very concrete about their results. However, the results are a crucial factor when comparing tools. All of these tools refactor Java programs, although some of them could theoretically refactor programs written in other programming languages if they would be adapted accordingly.

2. TOOL ATTRIBUTES

In order to examine the tools in a structured way and to facilitate the comparison, we introduce six attributes:

1. **Refactoring Goal**: The objective of the refactoring process. Examples are the introduction of design patterns or the maximization of a certain software metric. This attribute is most important for a developer to decide for a tool, because if it is unclear what the tool actually does, it makes no sense to use it.

2. **Refactoring Technique**: The approach of the tool. Examples are combinatorial optimization or genetic algorithms. Different techniques imply different capabilities regarding the refactoring process, such as large-scale refactoring versus small-scale refactoring. This can play a role when deciding for a tool.

3. **Input/Output**: What does the tool expect as input and what does it produce as output. It is important to know what the tools can be used for (e.g. Java programs) and what to expect as result (UML diagrams etc).

4. **Methodology**: The refactoring technique in detail. This attribute is useful to get a deeper insight into how the tools work.

5. **Experimental Design**: What kind of test programs have been used to evaluate the tool. This attribute helps to assess to what extent the results can be trusted.

6. **Result**: The findings of the experiment. The result of the experiments is one of the most essential attributes, because it is an indicator for the effectiveness of the tools.

3. TOOLS

3.1 A-CMA

3.1.1 Refactoring Goal

A-CMA [5] focuses on reducing technical debt. Technical dept (TD) occurs when a reduced quality in software in accepted to achieve a short-term goal. Usually it is assumed that the quality will be improved later on, but this will be the more expensive the longer the developers wait.

3.1.2 Refactoring Technique

A-CMA treats refactoring as a combinatorial optimization problem with different structures of the input software as search space. It supports a set of refactorings that can be applied in order to explore the search space. Two different metaheuristics can be used to guide the search. Solutions are assessed with different fitness functions. Each fitness function is composed of a combination of weighted metrics.

3.1.3 Input/Output

A-CMA refactors Java programs and expects the bytecode as input.

3.1.4 Methodology

Supported metaheuristics are hill climbing and simulated annealing, but also a random search is supported. A hill climbing heuristic selects a better solution based on the current solution. The neighbor solutions in the search space are considered and either the first solution which yields an improvement is taken (first ascent) or the solution which yields the greatest improvement (steepest ascent). A neighbor solution is a solution which can be reached by applying one refactoring operation. Which kind of hill climbing heuristic is used is also configurable.

Simulated annealing is similar to hill climbing. The difference is, that this heuristic starts with a so called temperature that represents the probability that a worse solution than the current one can be chosen. This temperature decreases during the search. When it reaches zero, simulated annealing behaves exactly like hill climbing. The purpose of the temperature is to overcome the problem of local optima. A local optimum is a solution that is best in its local neighborhood, meaning all its neighbor solutions are worse. Hill climbing cannot leave such a solution and thus may misses better solutions elsewhere in the search space.

A-CMA can choose from 20 different refactoring operations. These refactorings are grouped in operations that work on fields, methods and classes. A field-level operation is for example changing the security level, meaning it modifies the access specifier (private, protected, public). Another example is to move fields up to the superclass or down to a subclass. Method-level operations can do the same as field-level operations, but additionally the type of a method can be changed (to static or final, for instance). Class-level operations can add or remove keywords to classes, such as abstract or final. Furthermore, factories can be introduced the will be used instead of creating objects directly via constructors. Operations on all levels can also remove unused fields, methods or empty classes. Refactorings are only applied when it makes sense. For instance, a method which is already static will not be made static again.

The fitness function used by the heuristics can be specified by assigning weights to the 24 supported metrics. Examples for metrics are the number of fields per class, the number of implemented interfaces, the level of nested classes, the number of descendants per class and the number of elements that depend on a class. The authors of A-CMA used four fitness functions during their experiments: TD, abstraction, coupling and inheritance. TD has already been described above. Abstraction is a measure of how easy it is to extend a software system. Coupling denotes how many dependencies there are between classes. Inheritance refers to the amount of subclasses and implemented interfaces.

3.1.5 Experimental Design

A-CMA has been tested with six Java programs. Overall, they consist of 601 classes and $42,000$ lines of code.

3.1.6 Result

The results show that simulated annealing was superior compared to random search and hill climbing. Figure 1 shows the overall mean quality gain for the four fitness func-

tions. A-CMA was able to improve 2 out of 4 metrics: coupling and technical dept.

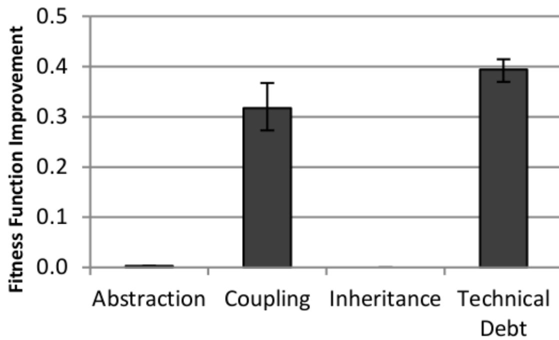


Figure 1: A-CMA - Results

3.2 TrueRefactor

3.2.1 Refactoring Goal

TrueRefactor [3] removes code smells.

3.2.2 Refactoring Technique

TrueRefactor utilizes genetic algorithms (GA) to determine the best sequence of refactorings for code smells it found. The genetic algorithms are guided by software metrics.

3.2.3 Input/Output

TrueRefactor takes the codebase directory of Java programs as input. It provides a GUI the user can use to parametrize the GA and to monitor the refactoring progress. When no parameters are given, TrueRefactor will select default values based on the complexity of the input program. The process stops when a certain quality has been reached which is indicated by a low number of code smells.

The output of TrueRefactor is an UML diagram of the control flow graph (CFG) that resulted from the best refactoring sequence. This helps programmers to get a better understanding of the solution TrueRefactor found.

3.2.4 Methodology

The first step is to build a CFA of the entire program. Based on this graph, code smells are detected. For each individual smell, TrueRefactor generates a sequence of algorithms that removes the particular smell. A GA searches for the best sequence of refactorings, meaning the sequence which removes the most code smells. Hence, the solution space of the GA consists of sequences of refactorings.

To detect code smells, TrueRefactor uses detection algorithms for five different code smells:

1. **Temporary Field**: A variable that is empty in most control flows.
2. **Shotgun Surgery**: A large modification requires many small changes at many different locations.
3. **Large Class**: A class that fulfills to many tasks.
4. **Lazy Class**: A class that fulfills not enough tasks.
5. **Long Method**: A too complex method.

For refactoring the code smells, 12 refactoring operations are available. Like A-CMA, TrueRefactor differentiates between class-level refactorings, method-level refactorings and field-level refactorings. Fields and methods can be moved through the inheritance hierarchy or to different classes. Beyond that, class hierarchies can be modified by splitting a class into a superclass and a subclass or by removing a class from a hierarchy and moving its functionality to a superclass or subclass. Also, methods can be split into multiple simpler methods.

The GA performs iterations until its termination condition is met. In each iteration it tries to find better solutions. The idea of genetic algorithms is to simulate the reproduction of living beings. The genes of the parents are recombined and maybe mutated to create the genes of the child. In the context of TrueRefactor, an individual is a sequence of refactorings. The number of individuals depends on the population size. When new individuals are created, parts of existing refactoring sequences are copied and glued together to form a new refactoring sequence. Then, mutation is applied with a certain usually low probability, meaning refactoring operations are exchanged, added or removed randomly.

Refactorings are evaluated by applying the refactoring sequence to the CFG. The modified CFG is checked for remaining code smells. Afterwards, a fitness function is used to assess how good the individual performed. The fitness function of the GA takes not only software metrics into account, but also reusability, understandability and maintainability. A subset of the best individuals is chosen to remain in the population, the rest is removed. This procedure is inspired by natural evolution, where only the fittest individuals survive. In the next iteration, the fittest individuals from the previous generation are recombined with the hope that this results in ever better refactoring sequences. The parameters of the GA, such as population size, mutation rate or maximum number of iterations can be adjusted via the GUI.

3.2.5 Experimental Design

TrueRefactor has been tested with a small tool for navigating a virtual vessel through a 2D environment.

3.2.6 Result

Figure 2 shows a comparison between code smell counts before and after the genetic algorithm refactored the source code. The abbreviations mean (from left to right): lazy class, long method, long class, shotgun surgery, temporary field. TrueRefactor was able to reduce the number of 3 out of 5 types of code smells.

3.3 Dearthóir

3.3.1 Refactoring Goal

Dearthóir [7] has its focus on improving the quality of software by optimizing multiple conflicting metrics. It defines quality as the compliance with a given set of heuristics that are built up from different metrics.

3.3.2 Refactoring Technique

Similar to A-CMA it uses combinatorial optimization and simulated annealing to explore the search space which consists of alternative designs. Dearthóir stochastically applies

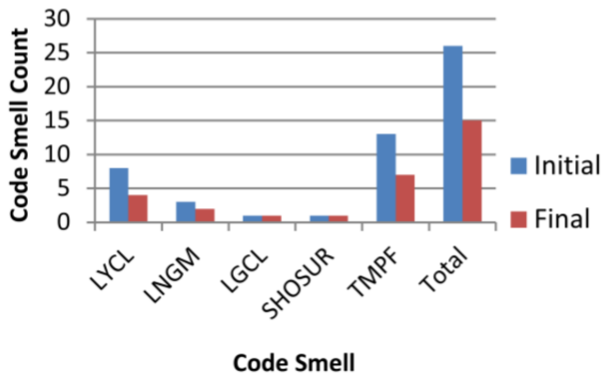


Figure 2: TrueRefactor - Results

refactorings to a program and evaluates the result. Heuristics steer the search into the right direction.

3.3.3 Input/Output

Dearthóir refactors object-oriented Java programs.

3.3.4 Methodology

The problem of many other refactoring approaches is that they concentrate on improving software with respect to one metric. That can lead to an overall poor design. For instance, restructuring the inheritance hierarchy often results in a reduction of cohesion. Object-oriented design often deals with trade-offs like sharing responsibilties to reduce the number of small classes. Unfortunately, this leads to stronger coupling.

Defining suitable metrics for automated refactoring is challenging. Quality metrics such as error-proneness are too abstract and are more useful for project management. For refactoring software in a way that it becomes more extensible and reusable, more detailed metrics are needed. An example for a suitable metric is *minimize accesses to attributes that are not defined inside the corresponding class.* In other words, *ensure high cohesion.*

As already mentioned above, Dearthóir defines quality as the compliance with a given set of heuristics. This set can be adjusted to the needs of the user. To compare different designs, a single quality value is calculated as the weighted sum of multiple metrics. The link between heuristics and metrics is that a heuristic is an intuition of what is good design. One or more metrics are derived from that intuition, while each metric is a formula that calculates a single value.

To explore the search space, simulated annealing has been chosen because it has proven to be effective in research. Refactorings are applied stochastically. In order to search the search space without restrictions, it must be possible to undo particular refactorings. For instance, pulling a method up to a super class has to be reversible by pushing it down to the subclass. Other supported refactorings are for example moving classes in the inheritance hierarchy or extracting a subclass from a superclass. The refactorings are limited to those that change the position of a method in the inheritance hierarchy.

There are several constraints when choosing a suitable heuristic. First of all, the heuristic must relate to a measurable property that has to be minimized or maximized. Moreover, the metrics derived from that heuristic have to have a precedence. Often metrics influence each other. For example, maximizing one metric can result in minimizing another metric. Therefore, it has to be defined which metrics are more important. More important metrics are assigned a heavier weight. The metric dependencies form a a directed graph which has to be acyclic. If it has a cycle, the set of metrics is not suitable for refactoring.

The heuristics used in Dearthóir are

- Same functionality in multiple classes is extracted and moved to a common superclass
- All base classes should be abstract, except it results in featureless subclasses
- The number of messages that can be sent to a class is minimized
- Useless classes are removed

The derived metrics are

- **RM**: Minimize rejected methods (Methods that are inherited but not used)
- **UM**: Minimize unused methods
- **FC**: Minimize featureless classes
- **DM**: Minimize duplicate methods
- **AS**: Maximize abstract classes

Figure 3 shows that duplicate methods and unused methods are most important and abstract classes are least important.

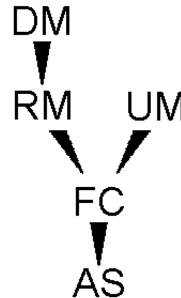


Figure 3: Dearthóir - Metric Precedence Graph

3.3.5 Experimental Design

Dearthóir has been tested with an artificial test Java program consisting of 8 classes.

3.3.6 Result

Figure 4 shows that Dearthóir could improve 3 out of 5 metrics.

3.4 REMODEL

3.4.1 Refactoring Goal

REMODEL [4] introduces design patterns into object-oriented software.

	RM–	UM–	FC–	DM–	AS+
Input	5	2	2	3	1
Output	0	2	2	0	2

Figure 4: Dearthóir - Results

3.4.2 *Refactoring Technique*

REMODEL uses genetic programming with software metrics as fitness function.

3.4.3 *Input/Output*

The output of REMODEL is a strategy of how to change the input software to a better design. This strategy can be applied manually or automatically.

3.4.4 *Methodology*

REMODEL consists of three modules: supported design patterns, the design change mechanism and metrics. This modular approach is more flexible than a monolithic approach, because each module can be exchanged independently.

Other refactoring approaches often make iterative changes to a program. The problem is, that such kind of modifications are not suitable for creating complex design patterns. However, evolutionary changes as implemented with genetic programming consider the interaction and composition of multiple design changes. Thus, the evolutionary approach of REMODEL is able to generate more innovative design patterns than iterative approaches. Design patterns are desirable, because they improve the maintainability and reusability of larger software systems.

In the context of REMODEL, an individual in the search space of the genetic algorithm consists of a design graph and a transformation tree. The design graph is comparable to an UML class diagram, but it contains additional semantic information such as function calls or class instantiations. The transformation tree consists of nodes which represent so called minitransformations. Minitransformations are simple refactoring operations, but composed they can create complex patterns. Examples for minitransformations are the introduction of a wrapper or a delegation.

When a transformation tree is executed, it modifies the design graph of the corresponding individual. Nodes are executed from the top to the bottom and from left to right. Besides the transformation nodes, there are also information nodes. They represents elements of the design graph such as classes. Their purpose is to serve as input to the transformation nodes. This is why information nodes have to be the children of transformation nodes.

Supported genetic operations are point mutation and subtree crossover. When point mutation is used, a single node is altered, added or removed. When subtree crossover is used, an entire subtree of the transformation tree is exchanged. It has to be ensured that the resulting tree is still valid. For instance, an information node must not be the parent of a transformation node.

There are two possible optimizations. Firstly, instead of single transformation nodes, REMODEL can work with transformation node templates that are known to introduce proper design patterns. Secondly, it has proven to be a more effective to tweak the genetic algorithm in a way that it introduces less transformation nodes.

The fitness function comprises a value calculated by the QMOOD metric suite [1], a value that reflects the presence of design patterns, a bonus for certain sequences of transformation nodes and a penalty for too many transformation nodes. The presence of design patterns can be detected with the help of a Prolog program or SQL. Each value except the QMOOD value has a coefficient, but it is an open questions which coefficients are most suitable.

3.4.5 *Experimental Design*

REMODEL has been tested with a large, web-based software system. This system is also known as ReMoDD, a repository for model-driven development. It consists of 23 classes.

3.4.6 *Result*

Figure 5 shows the total number of generated design patterns from five independent runs of REMODEL. On average, the best individual in each run had 4.5 transformation nodes.

Pattern	# of Instances
Abstract Factory	2
Adapter	13
Bridge	3
Composite	0
Decorator	2
Prototype	26
Proxy	15

Figure 5: REMODEL - Results

3.5 DPT

3.5.1 *Refactoring Goal*

DPT [6] constructs automated transformations in order to introduce design patterns into software.

3.5.2 *Refactoring Technique*

The programmer has to select the target design pattern. DPT in turn selects from its internal library of transformations the transformations needed for applying the design pattern to the input program. In a strict sense, DPT is not an automated tool, but its technique can be used to create fully automated tools. For instance, REMODEL builds upon DPT.

3.5.3 *Input/Output*

DPT is written in Java and also works with Java programs as input.

3.5.4 *Methodology*

The first step is to select a target design pattern for a specific part of the program. Then, a starting point for the transformation, a so called precurser, is chosen. A precursor can be seen as an intent for a certain design. It is simple and usually developed during the prototyping phase. Despite its simplicity it should not be of poor design. The transformation is now decomposed into a minipattern sequence. For

example, the minipatterns of a factory method are abstraction, encapsulated construction and abstract access. The same minipatterns can be found in different design patterns, meaning they can be reused. Minipatterns are created with the help of minitransformations. The target pattern is created via the composition of minitransformations.

DPT supports seven design patterns: Bridge, Strategy, Singleton, Prototype, Builder, Abstract Factory and Factory Method. REMODEL is based on the minitransformations of DPT.

3.5.5 *Experimental Design*

DPT has been tested with 23 design patterns from the Gamma catalogue [2]. For each design pattern, it was tried to find a suitable precursor and the corresponding minitransformations.

3.5.6 *Result*

Figure 6 shows that the methodology works for almost half of the design patterns.

Assessment	No. of Patterns	Percentage
Excellent	11	48%
Partial	6	26%
Impractical	6	26%

Figure 6: DPT - Results

4. DISCUSSION

The tools use basically two different techniques: combinatorial optimization and genetic algorithms. Genetic algorithms seem to be more useful regarding the introduction of design patterns because it refactors software on a larger scale which presumably leads to better solutions. Combinatorial optimization on the other hand just does iterative small-scale changes.

The tools show a similar success rate: between 48 and 60 percent. However, there are big differences in the experimental design. Some tools have been tested with real world applications whereas other tools have been tested with a small piece of software that has been written just for that purpose. The results of such tools may be not representative for their effectiveness.

The inputs of all refactoring tools are Java programs. The output however is either a formal description of the refactorings or the refactorings are really applied. To output just a description allows a developer to examine the refactoring solution to get a better understanding of how the solution actually modifies the input program. On the contrary, applying the changes fully automated saves developers much more time which seems to be the more significant benefit.

REMODEL is the most interesting tool since it is the only one that introduces large-scale changes fully automated. Furthermore, it has been testes extensively. The only downside is that it does not apply refactorings automatically.

A-CMA is a good choice: it refactors fully automated and underwent a large-scale test. On the other side, it only refactors on a small scale.

Dearthóir has similar characteristics compared to A-CMA, but an additional problem is that it has only been tested with a simple program.

TrueRefactor uses genetic algorithms but unfortunately it does not make use of its potential to refactor on a large scale. Other drawbacks of using this tool are that is has not been tested with complex software and that it just outputs an UML diagram of the refactored software.

DPT is only suitable for use as startpoint for fully automated solutions.

5. CONCLUSIONS

In this paper, we examined and compared five automated refactoring tools and found that all of them show potential, but not every tool is equally suitable for easing the life of a developer. First of all, the tools are restricted to Java programs which leads to less people who can use those tools. Moreover, not every tool actually modifies the input program, meaning the developer still has a lot of work to do.

Regarding robustness, REMODEL and A-CMA are the best choices. When deciding for a refactoring tool, the two most important things to consider are the scale of refactoring and if the changes are applied automatically. When large-scale refactoring is more important than the automatic application of the refactorings, REMODEL is the better choice. Conversely, A-CMA is the better choice.

One should bear in mind that all refactoring tools work with the help of metaheuristics, so there is no guarantee that they will find a good solution or any solution at all.

6. REFERENCES

[1] J. Bansiya and C. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 2002.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Elements of reusable object-oriented software. *Addison-Wesley Professional Computing Series*, 1995.

[3] I. Griffith, S. Wahl, and C. Izurieta. Truerefactor: An automated refactoring tool to improve legacy system and application comprehensibility. *Proceedings of the ISCA 24rd International Conference on Computer Applications in Industry and Engineering*, 2011.

[4] A. C. Jensen and B. H. C. Cheng. On the use of genetic programming for automated refactoring and the introduction of design patterns. *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, 2010.

[5] M. Mohan, D. Greer, and P. McMullan. Technical debt reduction using search based automated refactoring. *Journal of Systems and Software*, 2016.

[6] M. O'Cinnéide. Automated application of design patterns: A refactoring approach. *PhD thesis*, 2001.

[7] M. OKeeffe and M. O'Cinnéide. Towards automated design improvement through combinatorial optimisation. *Workshop on Directions in Software Engineering Environments*, 2004.

[8] J. Simmonds and T. Mens. A comparison of software refactoring tools. *Programming Technology Lab*, 2002.

Enterprise Architecture Model Classification

A Taxonomy for Model Requirements

Niels von Stein
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
niels.von.stein@rwth-aachen.de

Simon Hacks
RWTH Aachen University
Software Construction
Ahornstr. 55
52074 Aachen, Germany
simon.hacks@swc.rwth-aachen.de

ABSTRACT

The discipline of *Enterprise Architecture (EA)* is still objective to very active research that continuously reveals new insights. By employing case studies as evaluation method, the divergence between business needs and research output can be reduced significantly. However, there is a limited accessibility and reusability of real world EA models. In consequence, researchers tend to fall back to using exemplary models which might not adequately represent real world scenarios. This paper does a first step to overcome this by analyzing existing literature and aspects of their investigation. We define a taxonomy to classify EA models according to content and meta-model while also shedding light on the requirements for EA models in the evaluation of recent research.

Keywords

Enterprise Architecture, Model Database, Research Evaluation, Model Classification

1. INTRODUCTION

The continuous establishment of Enterprise Architecture (EA) techniques as a mean to model a holistic representation of corporate structures, processes and Information Technology (IT) infrastructure in practice still attracts many researchers today [1, 15]. At the same time, the application of in Information Science (IS) widely accepted systematic research procedures like Design Science Research (DSR) or adapted approaches ensure a certain level of quality. In DSR the evaluation is a crucial step to actually develop a resilient artifact which can contribute to research [22, 12]. While many tools for evaluation are given, employing case studies effectively offer a unique way to either understand real world phenomena or to observe interdependencies in a real world environment [5]. In contrast to this naturalistic evaluation approach there have also been observations of artificial evaluation approaches as Venable [20] describes. Here the developed artifact undertakes an evaluation at non-realistic conditions that produces results which do not hold in a realistic setting [20]. In EA research the use of case studies as evaluative method can be inhibited by companies concerns as the EA modeling represents a powerful strategic tool [13]. Although, there is no empirical evidence to which degree EA research faces this issue, many examples of a fallback to artificial evaluation by using exemplary data-sets can be given [7, 17, 2, 24].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SWC Seminar 2017/18 RWTH Aachen University, Germany.

In order to support researchers in the evaluation of their contributions, the vision of an Enterprise Architecture Model Database Platform (EAMDP) is born to offer a collection of real world EA models. While the dependency on artificial evaluation due to limited accessibility to real world cases can be circumvented, EAMDP could also serve as a publishing platform for conducted case studies to continuously enhance the pool of available models.

As a first step towards EAMDP this paper investigates which EA model aspects are of interest for evaluating EA research contributions. This breaks down into two parts where in one step a taxonomy is derived in order to establish a uniform measure to classify heterogeneous EA model datasets according to content and meta-model. In a second step recent research in the discipline of enterprise modeling is investigate to answer the research question:

- *What are the requirements for evaluative EA model data-sets?* (**RQ**)

The next section gives insights about existing literature in this direction. Section 3 elaborates in detail the approach to answer the research question. The results are presented in Section 4 while Section 4.2 also gives insights about schema application. In closing, Section 5 discusses the result and gives directions for further work.

2. RELATED WORK

In the past EA practitioners as well as EA researchers have put a lot of effort in formalizing EA model representation by defining sophisticated frameworks and meta-models. While today a large number of EA frameworks and corresponding meta-models exist, certain general patterns can be abstracted. In this context, Franke et al. [8] investigated possibilities of classifying EA frameworks according to shared concepts. The authors abstract entities and ar-

tifacts of EA frameworks to conceptual classes that form a meta framework. This work sheds light on differing modeling scopes of EA frameworks by comparing their coverage of the abstracted concepts in the meta framework. However, this meta framework does not provide any macro perspective on EA models since it focuses on single entities instead of larger model structures.

Furthermore, a layered representation of EA models has become a fixture in the discipline of enterprise modeling. While the top layer comprises strategic positioning and goals of the organization (specifying the "what"), subsequent layers derive required processes and IT architectures from it (specifying the "how"). A general differentiation between 5 layers among EA modeling frameworks can be observed while the actual set of present layers depends on the framework [23]. This encourages heterogeneity between EA models formulated by different frameworks and impedes a direct comparison. Nevertheless, this approach provides a good perspective on macro components of EA models why we take this as a foundation of our model.

In consideration of EAMDP as a platform for EA model data-sets, examples from other disciplines about the valuable contribution of data-set platforms can be given. In the field of machine learning and particular in computer vision ImageNet [4] has become the source for ground truth labeled general image data. The contribution of this database to research appears as the annual ILSVCR image recognition challenge which enabled some very important findings of the field [14].

3. RESEARCH DESIGN

This research follows the widely applied guidelines of DSR [22] in IS by contributing a viable and verifiable artifact which was developed by a structured procedure.

For the development of the model we performed a Systematic Literature Review (SLR) according to Kitchenham et al. [9]. We further use the results of our review as a single case study to the final schema with regards to our **RQ**. The keyword design was intentionally very general as we aimed for a wide coverage of publications in this field. The term set encompassed the terms "architecture", "architect", "enterprise", "enterprise architecture" and "enterprise modeling". As a result of limited accessibility to an sufficient number of journals due to commercial reasons, we decided to focus on conference publications in the retrieval.

We used *dblp*[1] to obtain well ordered collections of proceedings of prominent conferences[2] related to research in EA. Subsequently, we refined the result by applying the disjunction of all terms in the term set. Moreover, we limited the time interval to conferences in the last 8 years as [15] showed a strongly growing interest in EA by journal publications since 2009. Because we use a comparable term set, we expect a similar feedback at conference publications.

[1]http://dblp.uni-trier.de

[2]*X*CIS, CAiSE, ICEIS, EDoc

In the initial search we identified 25 potentially relevant publications by applying the following selection criteria:

- The research explicitly or implicitly follows a Design Science approach according to [22, 12]
- The research conducts an evaluation or application of a proposed method or model
- The research utilizes EA model case studies or exemplary EA model data-sets for their evaluation or application

This set was refined by a more in-depth inspection with the same schema to a total number of 14 relevant documents. Furthermore we excluded 3 documents from the pool as they propose an EA framework extension. Since a framework extension is meant to enhance the capabilities of a framework for certain use cases, each exemplary EA model used in the research demonstrates such a use case. The requirements for the evaluative EA model therefore are very specific and do not contribute to our analysis.

The demonstration of the taxonomy is performed by applying it to the results of the SLR as a single case study. This shows it's applicability while also giving insights to potential improvements. Since we applied small modifications to our initial model based on the feedback from the single case study, the findings of our work are not sufficiently evaluated in a methodical perspective. However, the improved taxonomy compensates for this and an evaluation will be given in future work.

4. EA MODEL CLASSIFICATION

4.1 Schema

The review shows that an essential differentiation of model artifacts is with regards to their general modeling intention. In [18], many model components related to the layered architecture model are investigated. However, the authors also look at governing model artifacts particularly about how well an architecture model is maintained. Consequently, we define the top level class *Category* that differentiates between architecture model and governance model. While the architecture model comprises the actual EA representation, the governance model captures artifacts of EA model management.

By further investigation on the architecture model we observed that all 11 papers operate their evaluation on rather specific components instead of looking at a whole model. Even if the authors introduce a case study with a comprehensive EA model, the evaluation considered very specific parts of it for example only the technical layer or process layer [21, 16]. In [17] the authors used an EA model's visions and goals hierarchy for their evaluation although the exemplary data-set consists of much more information. In this case the relevant components of the EA model were the dependencies between visions and goals. In [24, 2, 11] even larger components were used spreading along multiple layers of the EA model. The demand with regards to content of evaluative EA model data-sets represents a central criteria for data-set selection. Therefore, we define the second class *Scope* to capture this aspect. We adapted the layered approach and agree with Winter's outline of essential EA layers

Table 1: Results of EA model classification

Category	*Scope*	*Format*										
	Paper	[17]	[11]	[19]	[10]	[21]	[7]	[2]	[3]	[16]	[24]	[18]
Architecture	Motivation	◆	◆	◆							◆	❍
	Business		◆	◆	❍	❖	❖	◆			◆	❍
	Process		◆	▲	❍				▲	❍	◆	❍
	Application				❍	❖		◆		❍	◆	❍
	Technology					❖	❖	◆		❍	◆	❍
Governance	Maintenance											❍

◆:*ArchiMate* ▲:*BPMN* ❖:*Other* ❍:*Unspecified*

[23] for this class. However, we noticed in certain cases this outline was too detailed while at certain other cases information was lost when applying the schema. With regards to the architecture model this class consists of the following scopes:

- **Motivation** - We separate the EA motivation from the business model as a holistic perspective obscures differentiated dependencies on either the business model or the business motivation which is the case in [17, 18].
- **Business** - This represents the fundamental corporate structure as well as any relationships between actors or processes of the business architecture [23].
- **Process** - This layer is fully adapted and represents "the fundamental organization of service development, service creation, and service distribution in the relevant enterprise context" [23, p. 2].
- **Application** - Since there was no observation of requirements for a deeper differentiation of business integration and software architecture, we merge the layer "Integration Architecture" and "Software Architecture" representing an organization's enterprise services, application clusters and software services [23].
- **Technology** - This layer is fully adapted and represents the underlying IT infrastructure [23].

Subsequently, we apply this examination in a same way to the governance model. Fischer et al. identifies EA model maintenance as an important aspect of the governance model [6]. This is also reflected by the results of the SLR and is therefore includes as a scope for the governance model [18]. Although, there exists many more aspects of EA model governance (e.g. implementation procedures, governance goals) the SLR does not reflect any requirements for them in evaluative EA model data-sets. Therefore, it is left to mention that the list of scopes may not be completed at this point.

In some papers the proposed method relies on certain properties introduced by specific frameworks [24, 11]. Others require EA models where the actual meta-model was of less importance or they require models that follow either less formalized or more general meta-models [16]. Researchers therefore may require model data to follow a specific conceptual format which is captured by the last class *Format*. In this case, conceptual format serves as a generic term for meta-model or framework. The definition of member instances of this class is omitted as this can be easily derived from naming prominent frameworks. One technical remark about the scenario of EAMDP where researchers could even contribute an EA model of any format by specifying the corresponding meta-model directly in the database.

4.2 Application

Given the result of the proceeding step, the reviewed documents are classified according to their requirements on the evaluative EA model data-set which is presented in **Table 1**. This overview sheds light on the current situation according to the **RQ** and offers insights into the focus of recently developed methods in the domain of EA. In the following, two publications are discussed in more detail to further elaborate the rationale behind the approach.

In [17], the authors investigate methods for risk mitigation according to quantitative and qualitative risk measures of EA models. The evaluation of the method is performed on the prominent exemplary ArchiSurance EA model. However, the proposed method only requires the model's overall motivation as input. Given that, we can determine *Category*

and *Scope* as it targets the business' motivation. The data-set is constructed with ArchiMate which sets our third class.

In [19], a method for an automated re-design of EA's operational level based on data mining techniques is proposed. The authors employ a real case study to evaluate their findings. The case comprises a comprehensive EA model spreading over multiple layers. The evaluation does not require any governance model components so that the *Category* is set to architecture model. Within the architecture model, the required component is manifold. The method looks at business model, business motivation as well as process model. Consequently, the *Scope* reaches through these 3 layers. In Addition, the proposed method requires two differing conceptual formats for business model/motivation and process model. While the business part is expected to be formulated with ArchiMate, process models must be defined with BPMN. This renders the model requirements of this publication to a more complex variant.

5. CONCLUSION

In this study we derived a simple taxonomy based on accepted means in EA research that classifies models according to content and meta-model. For this we adapted and refined the idea of a layered model representation to a unified mean to quantify a model's components. Subsequently, we conducted a systematic literature review and applied this schema to EA model data-sets used in the evaluation of findings in conference publications over the past 8 years. Furthermore, we analyzed 11 papers to answer the question about what requirements for evaluative EA model data-sets exist.

The findings show that EA research and consequently it's evaluation requires very diverse EA model components in terms of scope. Nevertheless, evaluation on all identified components of an EA model rarely happened what gives reason for a federated approach for providing evaluative EA model data-sets via EAMDP. However, this requires further investigation by enhancing the literature review as there exist some limitations. A little number of publications only give implicit insights about the inputs of their evaluative data-set. While this amount was comparable small and any misinterpretation may not bias the results significantly, an extended review would eliminate this. Another limitation refers to the literature review as a single case study for the taxonomy. Since all publications either require no specific meta-model, rather unknown meta-model or the ArchiMate framework we had no verification if our taxonomy conforms with other prominent frameworks. However, we believe this is the case since we adapt approaches which comprises influences from many meta-models and frameworks.

Looking at the big picture of EAMDP, our taxonomy serves as a starting point of future development about the internal organization of this platform. At the same time further work is needed to investigate on technical aspects like model anonymization or model portability to finally lower the barriers of EA model sharing.

6. REFERENCES

[1] S. Aier, C. Riege, and R. Winter. Classification of enterprise architecture scenarios-an exploratory analysis. *Enterprise Modelling and Information Systems Architectures*, 3(1):14–23, 2008.

[2] G. Antunes, J. Barateiro, A. Caetano, and J. L. Borbinha. Analysis of federated enterprise architecture models. In *ECIS*, 2015.

[3] D. Birkmeier and S. Overhage. A semi-automated approach to support the architect during the generation of component-based enterprise architectures. In *ECIS*, page 218, 2012.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[5] L. M. Dooley. Case study research and theory building. *Advances in developing human resources*, 4(3):335–354, 2002.

[6] R. Fischer, S. Aier, and R. Winter. A federated approach to enterprise architecture model maintenance. *Enterprise Modelling and Information Systems Architectures*, 2(2):14–22, 2015.

[7] U. Franke. Enterprise architecture analysis with production functions. In *Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International*, pages 52–60. IEEE, 2014.

[8] U. Franke, D. Hook, J. Konig, R. Lagerstrom, P. Narman, J. Ullberg, P. Gustafsson, and M. Ekstedt. Eaf2-a framework for categorizing enterprise architecture frameworks. In *Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 2009. SNPD'09. 10th ACIS International Conference on*, pages 327–332. IEEE, 2009.

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

[10] D. Őri. Pattern-based misalignment symptom detection with xml validation: A case study. In *Workshop on Enterprise and Organizational Modeling and Simulation*, pages 151–158. Springer, 2017.

[11] S. Oussena and J. Essien. Validating enterprise architecture using ontology-based approach: A case study of student internship programme. In *Proceedings of the 15th International Conference on Enterprise Information Systems - ICEIS*, pages 302–309. IEEE, 2013.

[12] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[13] K. Pessi, T. Magoulas, and M.-Å. Hugoson. Enterprise architecture principles and their impact on the management of it investments. *Electronic Journal Information Systems Evaluation*, 14(1):53–62, 2011.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[15] P. Saint-Louis and J. Lapalme. Investigation of the

lack of common understanding in the discipline of enterprise architecture : A systematic mapping study. *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 1–9, 2016.

[16] A. Santana, D. Simon, K. Fischbach, and H. de Moura. Combining network measures and expert knowledge to analyze enterprise architecture at the component level. In *Enterprise Distributed Object Computing Conference (EDOC), 2016 IEEE 20th International*, pages 1–10. IEEE, 2016.

[17] S. Sousa, D. Marosin, K. Gaaloul, and N. Mayer. Assessing risks and opportunities in enterprise architecture using an extended adt approach. In *Enterprise Distributed Object Computing Conference (EDOC), 2013 17th IEEE International*, pages 81–90. IEEE, 2013.

[18] 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 14âĂŞ21, Nanjing, China.

[19] T.-M. Truong, L.-S. Lê, and L.-P. Tôn. Re-engineering enterprises using data warehouse as a driver and requirements as an enabler. In *Enterprise Distributed Object Computing Conference (EDOC), 2017 IEEE 21st International*, pages 67–72. IEEE, 2017.

[20] J. Venable. A framework for design science research activities. In *Emerging Trends and Challenges in Information Technology Management: Proceedings of the 2006 Information Resource Management Association Conference*, pages 184–187. Idea Group Publishing, 2006.

[21] R. Veneberg, M.-E. Iacob, M. J. van Sinderen, and L. Bodenstaff. Enterprise architecture intelligence: combining enterprise architecture and operational data. In *Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International*, pages 22–31. IEEE, 2014.

[22] R. H. Von Alan, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[23] 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.

[24] A. Xavier, A. Vasconcelos, and P. Sousa. Rules for validation of models of enterprise architecture - rules of checking and correction of temporal inconsistencies among elements of the enterprise architecture. In *Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 3: ICEIS*, pages 337–344. INSTICC, SciTePress, 2017.

Aspects of Software Complexity

Ali Ariff
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
ali.ariff@rwth-aachen.de

Andreas Steffens
RWTH Aachen University
Software Construction
Ahornstr. 55
52074 Aachen, Germany
steffens@swc.rwth-aachen.de

ABSTRACT

Software complexity is vague, hard to grasp and multifaceted nature of complexity leads to no complete methods and techniques to tackle complexity in software systems. It is due to the lack of knowledge of which aspects are existing that could affect complexity. The current existing research on the field is mostly from the architecture and the code point of view, although these two do not cover the whole of aspects. That is why research is needed to get to know and discover which another aspect that could affect complexity to be able to grasp the concept deeper. Moreover, by knowing the other hidden aspects of complexity, it will be easier to calculate the whole complexity. New technique or method such as Microservice and Domain Driven Design are claiming to solve or reduce software complexity emerges in last few years. Hence the question arises, in which aspect that technique or method can solve or reduce software complexity. This question will help to discover the aspect they tackle and also they newly introduce.

In this paper, aspects that are affecting software complexity will be identified. To accomplish this, various technique or method that exists and used in the industry nowadays claiming to reduce or removing software complexity and collect those aspects will be assessed. Hence, an initial overview of each technique or method will be provided, as well as their claim to reduce software complexity in a particular aspect. At the end of the paper, a set of aspects that are affecting software complexity will be given, and the updated definition of complexity based on the research will be defined.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

Keywords

software complexity, microservice, domain driven design, software architecture, devops

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SWC Seminar 2017/18 RWTH Aachen University, Germany.

1. INTRODUCTION

In general, complexity is an interactions or connections between components that are diverse from each other. In the software context, "complexity is the degree to which a system or component has a design or implementation that is difficult to understand and verify" [1].

The definition of software complexity comes from the number of things one should keep in mind when working on a software [18]. This number of things is vague, and this aspect of overall of software complexity wanted to be analyzed and discovered.

One definition of complexity is based on the lines of code which a program is measured based on how big the file size of the program as well how many of lines code the program has [12]. It is stated that the more code of lines the program has, the more complex the program is. However, nowadays, the lines of code aspect is not an aspect anymore to measure the complexity of a program. After all this time, codes (and lines of code) and architecture have always been the aspects of measuring the complexity. However, other unseen things needed to be found to have the complete measurement of complexity.

The other aspect of measuring complexity is the architecture. Based on Lehman's second law of software evolution, it is stated that as an evolving program change, its structure becomes more complicated. Extra resources must be devoted to preserving and simplifying the structure [13]. The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [14]. The architecture of software is one of the most important things to measure the complexity, for the possibility of future maintenance. If the architecture of a program is well designed and implemented, then it will be easier to maintain the program.

The complexity of software is an essential property, not an accidental one. Hence descriptions of a software entity that abstract away its complexity often abstract away its essence [4]. Therefore, it is essential to understand and know more software complexity aspects to be able to define overall software complexity.

Software complexity correlates well with development effort [2].

In the last few years, many techniques and methods are discovered and experimented whether to solve or to reduce software complexity. Among those techniques and methods, the two most renowned techniques and methods will be discussed. The techniques and methods are Microservice and Domain Driven Design. These two are the well-known techniques which are used in the current software engineering industry. More about Microservice will be explained in the section Microservice, and more about Domain Driven Design will be explained in the Domain Driven Design section.

This paper's goal is to comprise provision of initial overview of Microservice and Domain Driven Design; analyzing the techniques and methods to identify aspects of software complexity, and defined a new definition of software complexity.

The paper is structured as follows: in section 2, Microservice and its implication in the aspects of software complexity will be explained. In section 3, Domain Driven Design and its implication in the aspects of software complexity will be elaborated. In section 4, the renewed and updated definition of complexity based on the research will be assembled and defined. Moreover, finally, in section 5, conclusion and future works will be explained.

2. MICROSERVICE

In short, a Microservice architectural style is an approach to developing a single application as a suite of small services, each running in its process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies [9].

A Microservice focus on a single business capability, complete business capability mean that the process can complete without depending on other Microservice. Microservice should have its database and sharing database between Microservice is prohibited, this is to avoid conflict of interest since a Microservice should be able to solve single specific business problem.

Communication in Microservice also need to be stateless, so more copies of Microservice at will can be added. Stateless communication gives Microservice the ability to scale effortlessly, each interaction with our Microservice can be handled by a different instance. Without any state, the communication between the team will be easier to achieve since there is no defined state in the beginning.

Microservice should be able to continue even though another Microservice fails. If the communication with another Microservice cannot be made, the current Microservice should be able to make due by using default value or fail-over strategy.

Microservice can also be made based on the problem that needed to be solved. The service can be created using the related programming language as well as using the right technology based on the problem that needed to be solved.

If we want to talk about the benefit of using Microservice, it will be easier if it is compared to the traditional architecture that creates large monolithic packages. Hence this can be challenging to deploy. Moreover, for the scaling, it needs to scale the whole software by adding more copies.

On the other hand, Microservice is flexible to deploy because of the independent nature of the service, and it can be deployed at any time. For the scaling, Microservice can be scaled horizontally in the specific services that are needed resulting it to save the resources. Furthermore, Microservice can also increase the resiliency because of the each designed service in the isolation.

In Microservice, four aspects that are related to software complexity are identified: architecture, operation, skill, and security.

2.1 Architecture

Cohesion and coupling are the main issues in term of software architecture. These two things are considered as the factor that is affecting architecture complexity. A good software architecture is when it has the high cohesion and the loose coupling.

Microservice is loosely coupled service-oriented architecture with bounded contexts [6]. When services are loosely coupled, a change to one service should not require a change to another. The whole point of a Microservice is being able to make a change to one service and deploy it, without needing to change any other part of the system [17].

Finding boundaries within problem domain help ensure related behavior is in one place, and communicate with other boundaries as loosely as possible [17]. Bounded context from Domain Driven Design helps to separate the problem domain into specific service that has a clear boundary. This makes Microservice have high cohesion because one only needs to change one specific service to change the behavior of the software. Therefore, high cohesion and loosely coupled architecture reduce the complexity, making this aspect one have to think.

2.2 Operation

Operation or DevOps is the combination of cultural philosophies, practices, and tools that increase an organization's ability to delivers applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes [3]. The delivering here is including the process to test, deploy, and monitor the application.

Unit testing in Microservice is more or less the same as in the regular software, but if it is the talk about the integration testing, this will be another thing because Microservice

is a distributed software, the testing remote calls from one Microservice to another Microservice is a hassle.

Microservice deployment usually uses containerization technology, when a Microservice is bundled in a single image within all their library dependencies so it can efficiently distribute or deployed in the production. The management of infrastructure or the cloud technology needs to be done to manage the resources to do that. Automated deployment using pipeline is also significant to make fast and reliable deployment, it needs to be configured in such a way using specific tools to make it possible.

In the production environment itself, there are many things to configure, for example, the service discovery service to detect Microservice which is active and ready to communicate. Load balancing for each Microservice cluster to distribute the request evenly. Fault tolerance is also needed to make Microservice resilience, with a technique such as a circuit breaker will fall back gracefully and stop the cascading failures. If the system using REST API it also need to have API management system to handle thing such as authentication, rate limiting, caching, and others.

Monitoring is also crucial after the software lives into production. The strategy which commonly used to monitor a lot of Microservices is using central logging system. In the system, there is a central service or software which receives or listen to any log from every Microservice and then crunching those data and visualizes it to the user and also alerting system use those data to decide whether something wrong happens in the whole system. It can be using some threshold value to start alerting people or event-based notifier.

2.3 Skill

Advocates of Microservices are quick to introduce the Conway's Law, the notion that the structure of a software system mirrors the communication structure of the organization that built it [10]. Microservice is required to build the team based on the service to reduce the inter-team communications. One service will be handled by one team, and to accomplish that, various skills are needed in the team from back-end engineer, front-end engineer, DBA, QA tester, UI designer, UX researcher, DevOps engineer, and many more. The team should be able to handle their service as a complete product. Therefore, the team consisting people with various skills are important.

Building and managing team with different skill-set is another complexity to think of.

2.4 Security

Security is another important thing that is related to software complexity. Microservice is normally communicating through HTTP API [9]. In the security perspective, when a system has many services and talking to each other, the abstraction of the security is required and needed to be implemented by each of the services.

That service will have different technology stack and different way to configure and hardening. It will also manage the version of each technology stack also needed to make sure the technology stack is kept up to date to the latest version to avoid vulnerability, and on the other hand, it also needs to make sure that the update they have does not have breaking changes.

Complexity in the aspects of security is increasing as long as the service increases because those services need to be hardened and secured.

3. DOMAIN DRIVEN DESIGN

Domain Driven Design (DDD) is an approach to develop software for complex needs by deeply connecting the implementation to an evolving model of the core business concepts [8].

DDD is a technique to help to make architecture design decision to reduce the complexity by separating it on the different domain. It helps to choose how to separate domain based on what one separates it. It will lead to the precise boundaries that separate those domains. DDD is all about solving the problems of an organization, and so the Domain Model is all about understanding and interpreting the essential aspects of a given problem.

Naming is hard; even it is usually wrong. In DDD, the domain expert is used for people that have deep understanding of its subject or topic can bridge term in business and technical.

In DDD, four aspects that are related to software complexity are identified: domain, process, organizational, and code.

3.1 Domain

A domain is a sphere of knowledge, influence, or activity [7]. The domain is the ideas, knowledge, and data of the problem people are trying to solve. The Model of a Domain Driven Designed project is the solution to the problem. The Model usually represents an aspect of reality or something of interest. The Model is also often a simplification of the bigger picture, and so the important aspects of the solution are concentrated on while everything else is ignored [5].

Modeling domain is the critical part of building software, and it is not an easy task, there are many things to consider. The Domain Model is the organized and structured knowledge of the problem. The Domain Model should represent the vocabulary and fundamental concepts of the problem domain, and it should identify the relationships among all of the entities within the scope of the domain. DDD is coming with an offer to modeling domain based on business process and put in in some context [5]. In here they called it Bounded Context a concept when put a clear boundary between context. This boundary is created so people can have isolation and ability to do development without being dependent.

3.2 Process

Building software based on DDD comes with some drawback. Process to create software will be more complex because, to help maintain the model as a pure and helpful language construct, one must typically implement a great deal of isolation and encapsulation within the domain model [16].

This process also requires some stakeholder (for example domain expert, software developer) to collaboratively discuss together and come up with model that is fit.

3.3 Organizational

Communication inter-team is very important to deliver fast good and correct software. Hassle in coordination can be a bottleneck when creating software.

In any organization, limiting the factor on delivering faster will be resulting bottleneck, and to make it productive, one should fix the bottleneck [15]. DDD is coming with the concept of bounded context solve this bottleneck by creating precise boundaries of each context. This clear boundary makes the team member does not depend on each other. Therefore each team is not going to be waiting for each other to do their job because of some task blocked by another team.

3.4 Code

The code structure is one of the most important aspects related to the code's complexity. The bigger the code base is, the more matter that is existed to think, resulting the code to be more difficult to understand for the team member, as well as the coordination needed to maintain and create the code [11].

The other drawbacks from having a big code base are bugs and errors will be likely to happen. When there is only one code base, the whole team have to take care of that only one, making the communication and coordination more difficult and complicated since there are many things to take care. That is why with DDD, the code base will be split into smaller parts. By using this strategy, the parts will be isolated, and the distribution to the team member will be more straightforward, and since the code is split, the complexity will be decreasing.

4. DEFINITION OF COMPLEXITY

In the previous section, aspects from Microservice and DDD have been discussed. In Microservice, four aspects are related to complexity which are architecture, operation, skill, and security while the related aspects of DDD are the domain, process, code and organizational.

High cohesion and loosely coupled architecture reduce the complexity. The more operation existed, the higher the complexity is. The skills from each member of team and also how to manage the team is one aspect of complexity, while in the aspects of security, it is increasing as long as the service increases because those services need to be hardened and secured.

Domain has to be split to reduce the complexity. The process to create software will be more complicated because, to help maintain the model as a helpful and straightforward language construct, one must typically implement a great deal of isolation and encapsulation within the domain model [16]. With an extensive organization and team that is dependent on another team, the complexity will be prominent, and having one code base without splitting it into smaller parts will increase the complexity of software.

Techniques and methods that were claiming to solve or reducing software complexity is shifting the software complexity to another aspect, and it helps us discovering more and more aspects of the overall software complexity.

It is difficult to focus only on one aspect while neglecting the other aspects during the development. Therefore, software complexity is everything related to the development: operation, architecture, skill, domain, security, process, organizational and code. The software complexity is not something that can be measured only using lines of code and architecture in general. All of the aspects have to be considered to measure the complexity.

5. CONCLUSIONS AND FUTURE WORK

Complexity is not measured fixedly based on the few definition that has been stated in the introduction. It is not merely measured based on the number of lines of code where the higher the number, the more complex it is; and although architecture becomes one of the important aspects in complexity, there are more hidden aspects behind it.

Among the existing techniques and methods, two of them which are Microservice's and DDD's implementing effects and drawbacks have been analyzed and discussed in the previous sections since these two are the most well-known techniques that have been used in the current software engineering industry. This side effect help to discover more aspects of overall software complexity that people might be forgotten or put aside. This list of aspect might be increased in the future when there is new technique or method that is claiming can reduce or remove software complexity. After doing the analysis, the list of aspects that affecting software complexity are operation, architecture, skill, domain, process, organizational, security, and code.

The overview of software complexity aspects shown in figure 1 that have been described and discussed in previous chapters. Based on the observation between both of the techniques, a new definition of complexity has been defined. Software complexity is everything related to the development which are but not limited to operation, architecture, skill, domain, security, process, organizational and code.

We have introduced a new definition of complexity, which captures several aspects not included in previous definitions. The preceding chapters have shown that these aspects are essential concerning the evaluation of approaches like DDD and Microservice.

The list helps to have more understanding of overall software complexity. In the future, this list can be used as

Figure 1: Software Complexity Aspects

a metric to measure software complexity when combining with another method. The goal is to use this list and find a formula to calculate each complexity aspect so we can get the overall complexity of the software. It is also expected to discover more hidden aspect of complexity that might be existed yet haven't been thought of.

6. REFERENCES

[1] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[2] M. AlSharif, W. P. Bond, and T. Al-Otaiby. Assessing the complexity of software architecture. In *Proceedings of the 42Nd Annual Southeast Regional Conference*, ACM-SE 42, pages 98–103, New York, NY, USA, 2004. ACM.

[3] Amazon. What is DevOps? `https://aws.amazon.com/devops/what-is-devops/`. Retrieved December 10, 2017.

[4] F. P. Brooks, Jr. No silver bullet essence and accidents of software engineering. *Computer*, 20(4):10–19, Apr. 1987.

[5] P. Brown. What is the Domain Model in Domain Driven Design? `http://culttt.com/2014/11/12/domain-model-domain-driven-design/`. Retrieved December 23, 2017.

[6] A. Cockcroft. Migrating to Microservices. `https://www.slideshare.net/adriancockcroft/goto-berlin`, 2014. Retrieved December 10, 2017.

[7] E. Evans. *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[8] E. Evans. What is Domain-Driven Design? `http://dddcommunity.org/learning-ddd/what_is_ddd/`, 2007. Retrieved December 10, 2017.

[9] M. Fowler. Microservices: A definition of this new architectural term. `https://martinfowler.com/articles/microservices.html`, 2014. Retrieved November 26, 2017.

[10] M. Fowler. Microservice Trade-Offs. `https://martinfowler.com/articles/microservice-trade-offs.html`, 2015. Retrieved December 10, 2017.

[11] H. Graca. Domain-Driven Design. `https://herbertograca.com/2017/09/07/domain-driven-design/`. Retrieved December 23, 2017.

[12] W. Harrison, K. Magel, R. Kluczny, and A. DeKock. Applying software complexity metrics to program maintenance. *Computer*, 15(9):65–79, Sept 1982.

[13] M. M. Lehman and L. A. Belady, editors. *Program Evolution: Processes of Software Change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.

[14] B. Len, C. Paul, and K. Rick. Software architecture in practice. *Boston, Massachusetts Addison*, 2003.

[15] E. M. Goldratt and J. Cox. *The Goal*. The North River Press, 3 edition, July 2004. The first issue was published in 1984.

[16] Microsoft. Architectural Patterns and Styles. `https://msdn.microsoft.com/en-us/library/ee658117.aspx`. Retrieved December 17, 2017.

[17] S. Newman. *Building Microservices*. O'Reilly Media, Inc., 1st edition, 2015.

[18] R. Strukhoff. Reducing Complexity of Software with Domain-Driven Design and Microservices. `https://www.altoros.com/blog/reducing-complexity-of-software-with-domain-driven-design-and-microservices/`. Retrieved December 21, 2017.

Technical Debt Prioritization Approaches for Non-Technical Manager

Jan Uthoff

Full-scale Software Engineering

February 5, 2018

Motivation

Technical Debt

Customer's view

Developer's view

Background

- metaphor origins in financial domain
- use decision-making procedures
- how to handle architectural debt?

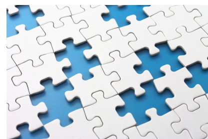
Vocabulary

Principal:

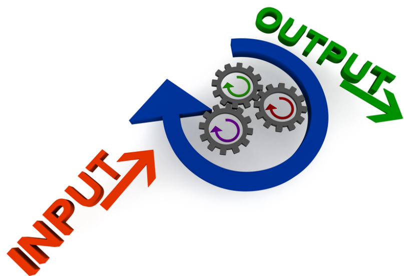
- cost to complete the task

Interest:

- cost if task remains undone



Comparison Framework



Comparison Framework

Procedure

Challenges

Technical debt examples

Documenting code:

- principal: 1h
- interest: 3h

My Project

Main Page | Classes | Files | Search

Class List | Class Index | Class Members

Foo 〉 Foo 〉

Public Types | List of all members

Foo::Foo Class Reference

Public Types

enum Foo
Foo enum, possible ways to foo.

The documentation for this class was generated from the following file:

- Foo.h

Generated on Thu Dec 6 2012 15:38:12 for My Project by doxygen 1.8.2

Adding new feature:

- principal: 2h
- interest: 1h

Approach: Cost-Benefit Analysis

Goal:

- sort alternatives

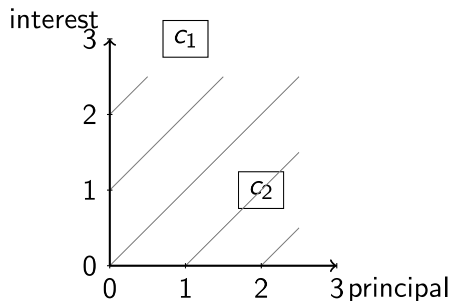
Input:

- estimated cost to complete the task
- estimated price if task remains undone

Approach: Cost-Benefit Analysis

Procedure:

- sort items by principal-interest ratio

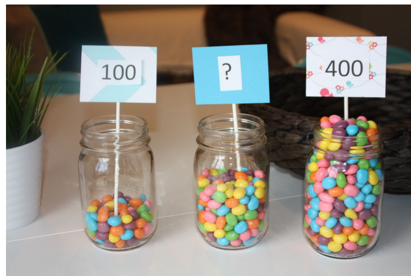


c_1	Documenting code
c_2	Adding a feature

Approach: Cost-Benefit Analysis

Challenge:

- estimate the input



Approach: Analytic Hierarchy Process

Goal:

- choose an alternative that maximizes/minimizes an objective function

Input:

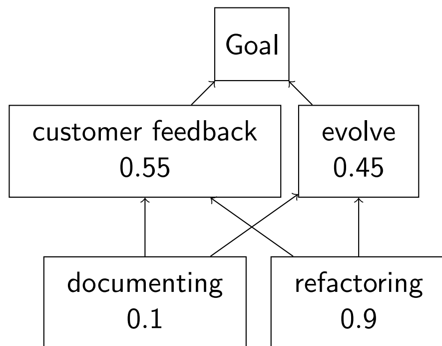
- criteria
- weights for each criterion

$$f(x) = w_1 \cdot c_1(x) + w_2 \cdot c_2(x) + \dots$$

Approach: Analytic Hierarchy Process

Procedure:

- pairwise comparison



$$
\begin{aligned}
f(i_{doc}) &= 0.55 \cdot c_1(i_{doc}) + 0.45 \cdot c_2(i_{doc}) = 15 \\
f(i_{feat}) &= 0.5 \cdot c_1(i_{feat}) + 0.45 \cdot c_2(i_{doc}) = 13
\end{aligned}
$$

Approach: Analytic Hierarchy Process

Challenges:

- identify criteria and weights
- estimate the criteria function

$$c_1(x) = \quad ?$$
$$c_2(x) = \quad ?$$
$$\ldots$$

Approach: Local Manager

Goal:

- satisfy recurrent customers

Input:

- location of the technical debt
- plan of the upcoming product requests

Approach: Local Manager

Procedure:

- dividing product into parts
  two parts
- which product must be completed next

Delivery Dates

- i_{doc}: in 5 weeks
- i_{feat}: in 3 weeks

Program part

- i_{doc}: core
- i_{feat}: feature

Approach: Local Manager

Challenge:

- the costs

Discussion

- precision
- performance
- cost-Effectiveness

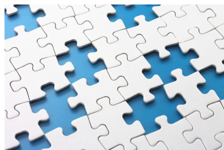
Research

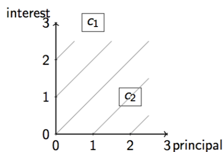
- **researches required!!!**
- case studies
- historical data

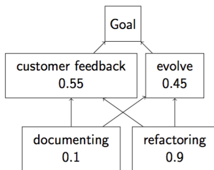
Summary

- why deal with technical debt
- introduced three approaches
- no concrete comparision possible

Slide overview







Thank you

Towards a Quality Model for DevOps

Leon König
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
leon.koenig@rwth-aachen.de

Andreas Steffens
RWTH Aachen University
Software Construction
Ahornstr. 55
52074 Aachen, Germany
andreas.steffens@swc.rwth-aachen.de

ABSTRACT

Continuous Delivery is widely considered to be a critical requirement in the context of software engineering. The DevOps approach aims to bring together the opposing interests of software development and IT operations to improve the efficiency of development as well as the deployment process.

Whereas the goal of DevOps is defined pretty clearly, no convincing quality model to state the maturity of a certain approach in all its facets could be found. As DevOps is considered to be a mindset and thus a very heterogeneous environment, existing quality models usually focus on a subset of aspects and stakeholders.

This paper intends to suggest a quality model covering the DevOps process along with the technical aspects of the pipeline model and the underlying pipeline system. Therefore, a set of quality attributes along with adequate metrics has been identified.

Keywords

DevOps, Continuous Delivery, Continuous Deployment, quality model

1. INTRODUCTION

DevOps is a term that was shaped over time. Therefore, a clear definition of the parts included in DevOps does not exist. In a historical context, the separation of development and operations teams led to conflicts. A development team is considered to be successful when it is able to develop and deliver new features in a short period of time. Operation teams on the contrary, have to keep their servers up and thus do not want to change anything on a running system to gain a certain stability [6].

DevOps is the result of an integration of movements like lean manufacturing, agile transformation and continuous delivery. Derived from the manufacturing value stream of Lean, a technology value stream which converts a feature requests to value for the customer should be established. Small teams implement these features independently from each other. Automated testing and telemetry in code and production environment foster fast feedback loops to detect and fix problems as early as possible which also supports a continual and genuine learning process. As deployments are automated releases become predictable and reversible. However, monitoring and measurement is an essential part of DevOps and enables teams to treat "product development and process improvement as experiments" [9]. A collaborative culture within and between teams is needed to achieve the shared goals, allow a collective ownership and share the resulting values [6].

DevOps originated from the conflicts between separated teams, hence there is especially no distinct team responsible for the implementation of DevOps as Humble stated in [5]. There could be a team supporting development teams in this, but all in all the development team should develop and own this process.

The State of DevOps Report 2017 by Puppet and DORA mentioned DevOps to be a mindset many enterprises currently try to implement [13]. Whereas DevOps methodologies are widely discussed regarding DevOps culture and available tools, no suitable quality model for DevOps approaches which is widely accepted can be found. During research a bunch of papers addressing this gap had been identified. As stated above DevOps is a very heterogeneous environment hence most of the papers only address a part of the DevOps approach. This paper aims to integrate and supplement the existing concepts for quality models. The resulting DevOps quality model focuses on non-technical as well as technical aspects, which there are the DevOps process and culture, the pipeline model and its underlying pipelining system as these can be considered as the substantials of a successful DevOps implementation.

Therefore, this paper firstly discusses the related work in section 2. The following section 3 presents the DevOps quality model. After that, section 4 critically discusses the suggested quality model. The last section 5 summaries the work and discusses future work to be done for defining an entire DevOps quality model.

2. RELATED WORK

During literature based research, a number of related publications was found. Only a few of them describe a proper quality model for DevOps. Other papers state core components and capabilities of a successful implementation which

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SWC Seminar 2017/18 RWTH Aachen University, Germany.

can also be seen as part of a quality model. However, many quality models rather focus on continuous delivery than on DevOps itself as this concept was described earlier. Nevertheless, as this is also seen as an essential part these are also included.

Juner and Benlian's work [8] based on information gained from five case studies in different enterprises identified five capability models. Like [14] they also identified fast feedback-loops as an important feature to foster a continuous value stream in the organization. Similarly, infrastructure as a code leads to an deterministic configuration of software and infrastructure which only then allows a continuous deployment when it is applied as the single source of truth and no manual changes are applied. To achieve these automatic deployments, the deployment pipeline has to enable these builds, tests and deployments in an auditable and fast way. A variety of QA arrangements, like staging environments, unit and integration tests, or code analysis ensure product quality and can also be seen as part of the feedback-loops mentioned above. In the same way, metrics monitoring systems and software enable developers to gain faster feedback on their doings.

Riley went further to define metrics measuring DevOps quality for example by deployment / change frequency, change lead time and failure rate or mean time to recover. These metrics allow an evaluation of response times, development tool efficiency, team capabilities and overall efficiency of the process [16].

While [10] also defined metrics these concentrate on data that can be retrieved from the continuous delivery and the deployment pipeline, which are also a central parts of a DevOps implementation. The authors mentioned development, deployment and activation time to be useful to state quality of the process. Focusing on continuous deployment, they also defined an *oldest done feature* metric to measure the time a feature needs at most to be deployed. On the pipeline level, metrics like features or releases per month and the fastest possible lead time give suitable insights.

Mills developed a matrix to state maturity on the pipeline level. He validated whether in his view important steps are covered by the pipeline and derives thereby a simple maturity for the examined continuous delivery pipeline. He moved from continuous integration builds with unit tests and static code and coverage analysis to an automated deployment of the generated artifact to an infrastructure that was provisioned automatically. On this system, he then suggests to carry out various tests including functional, cross-version, performance and security tests. As a last step, he described some release management activities like release tagging, production deployment but also roll-back abilities of the pipeline [11].

Forester Consulting developed a maturity model for continuous delivery capabilities divided into five levels reaching from an initial to an optimizing level. On the first initial level, the delivery process relies on manual activities leading to ad hock deployments. The managed level already requires some automation in provisioning and acceptance testing, an iterative development process and clear responsibilities to enable an adaptive delivery process with planned releases. Running continuous integration tests and considering work only as done when it passes these, the next maturity level is reached allowing regular releases. The upper two levels focus on release on demand and deployment capabilities to meet business needs by suitable deployment pipelines, cross-functional teams, monitoring and an architecture which also keeps continuous delivery approaches such as dark launching in mind [3].

Trienekens on the other hand, based his research on the three principles system thinking, amplifying feedback loops and a culture of continuous experimentation and learning also described by Kim et al. in [9]. The measurements he suggests focus on current states of these three principles. Furthermore, he evaluates the effect of process changes to the overall performance of the process by comparing f.e. lead times or number of incidents before and after this change. When considering feedback loops he mainly focuses on feedback given by customers but also how well the process can respond to it. For the last principle of learning and experimentation the amount of time for knowledge storage and retrieval and the teams reflection on their work and learning process are mentioned. Furthermore, he also measures their ability to detect faults which may be introduced deliberately to foster their ability to react on real failures [18].

Since the Capability Maturity Model Integration (CMMI) is a widely accepted model to determine the maturity of a certain process or organization, Rong, Zhang and Shao rested their research upon a combination of CMMI-DEV and CMMI-SVC to appraise the maturity of a certain DevOps implementation. In their case study, they performed interviews with one manager each from project, development, testing / operation or process improvement and conducted an appraisal based on this combination. They concluded that a combination of CMMI-DEV and CMMI-SVC to evaluate a DevOps implementation is basically possible but also leads to some difficulties when classifying observations as high or low maturity, or when determining where to improve a process [17].

Gupta with [4] and Rehn, Palmborg and Boström with [15] both developed a continuous delivery maturity matrix using the five levels of CMMI and segregate each of them into certain areas.

Gupta separated into the six areas culture & organization, build & deploy, release, data management, test & verification and information & reporting in detail. For each level he suggested certain capabilities and practices. As always, the initial stage does not assume any automation while higher stages require continuous integration, scripted or orchestrated deployments and provisioning, automated acceptance tests or automatically applied changes to datastores. His vision of a process reaching the optimizing level consists of cross functional teams, deployments that do not need any human intervention and a release process that is always rolling forward to fix defects. The information gathered during this process should be accessible to any member across organizational boundaries [4].

In comparison, Rehn et al. left out the release and data management areas to add an area called design & architecture. This area contains database, library and API management on the lower levels moving on to configuration as code, branching techniques, component based architecture and infrastructure as code. Similar to Gupta they suggested the removal of team boundaries moving to cross functional teams responsible for the whole process. A continuous im-

provement motivated from Kaizen is also seen as part of a continuous delivery culture [15].

Furthermore, Solinea developed a DevOps maturity model which is described by Parks. The model also uses five maturity levels derived from CMMI. The areas process, people, technology and culture are considered to be the "Four Pillars of DevOps". The items described for each level are similar to the ones mentioned by Gupta or Rehn et al. for a continuous delivery maturity model [12].

In his master thesis on the adoption of DevOps in software product organizations, de Feijter developed a model to evaluate the maturity of a certain development process regarding DevOps concepts. He suggested 16 focus areas for which different capabilities to reach one of the eight maturity levels are stated. In addition to the continuous delivery maturity models described previously, he added continuous deployment competences and also suggested recoverability and resilience tests on production systems. As a interesting point he proposed gated check-ins to version control systems which merge changes with the mainline, carry out tests and reject commits if they do not pass these tests to keep the mainline buildable [2].

Unfortunately, the DevOps maturity model of Solinea is only described briefly with no more information supplied for the items claimed to gain a certain maturity. The maturity model described by de Feijter is very complicated to read as he describes two to five different capabilities for each focus area on roughly 20 pages. Later on he then sorts all these capabilities into one of the eight maturity levels. Most of the other maturity or quality models mentioned above only consider capabilities and changes that come along with continuous delivery. As DevOps aims to deliver value to the customer as fast as possible, changes have also to be deployed to production and not only to staging environments. Consequently, a DevOps requires continuous deployment which is going further than continuous delivery.

3. A DEVOPS QUALITY MODEL

The new DevOps quality models aims to integrate the above quality models for DevOps and continuous delivery to design a potential quality model. Furthermore, the quality model should distinguish between the process and the technical implementation by a continuous deployment pipeline.

The quality model will be developed in an GQM-like (Goal-Question-Metric) approach according to [1].

1. In GQM a goal should always include the entire goal, the purpose of this goal and also a viewpoint from which one will analyze this goal.

   Therefore, two conceptual goals are described. As one would state the quality of an existing process the intention of all these goals is to analyze the given circumstances.

2. The questions purpose is to describe the goal and its assessment in a more detailed way.

   Hence, multiple questions are specified to emphasize the two conceptual goals.

3. To answer the questions, one uses metrics to meassure them in a quantitative way.

   Consequently, the metrics which help to evaluate the questions and goals are derived from the literature described above.

Due to page restrictions and time limiations the resulting quality model will not be verified by a case study as suggested by GQM. Consequently, the first quality model can also not be adapted to the results of such an evaluation.

3.1 Goals

The quality model developed in this paper intends to analyze the DevOps process. Parts of the DevOps process are the whole development and deployment process as well as the way team members and teams work together to generate value to the customers. On the other hand, the development and deployment pipeline is a central component and the technical implementation of this process without which one will never reach desired performance and flexibility. As the success of a DevOps implementation is mainly dependent on this component it should also be covered by this quality model.

Consequently, the following two conceptual goals can be identified:

1. Analyze the DevOps processes of an organization to foster a continual and efficient flow of value to the customer from the viewpoint of

   a. the teams associated with the development and deployment process (an interdisciplinary team or development and operation teams)

2. Analyze the technical implementation of the DevOps process to improve the flow of feedback from the viewpoint of

   a. the teams associated with the development and deployment process (an interdisciplinary team or development and operation teams)

Within a DevOps environment the teams are central. In an ideal setting the teams develop and improve their way of working in an continual way. Hence, both goals analyze from the viewpoint of the teams associated with the whole process.

3.2 Questions

Kim et al. described three ways of a successful DevOps process [9].

The first way and the principles of flow originated from the lean movement. Traditionally, the work flows from development to operations until it generates value to the customer. This flow of value should be accelerated by making work is made visible, limit work in process, reducing the number of handoffs and batch size, eliminate waste of any kind and claim way of thinking on the whole system.

The principles of feedback are supposed to create a flow of feedback from right to left. Thereby, one can detect problems earlier, fix them before they get catastrophic and react faster to changing customer needs.

The last principles of continual learning and experimentation depicts a culture of high-trust, where learning from success and mistakes of each other and sharing of knowledge globally is part of daily work. Going further, one can even think of injecting failures to production systems to enforce employees to validate and improve their capabilities to react.

As these three ways are widely accepted and seem appropriate, an existing DevOps process will be analyzed along the principles of these ways and used as questions to specify the first goal of analyzing the DevOps process.

The development and deployment pipeline is a product of a seperate development process with the goal of allowing a continuous way of building, testing, analyzing and deploying software to different environments.

Consequently, the analysis of the technical implementation of DevOps including pipeline model and pipelining system will mainly rely on a subset of the product quality characteristics suggested by ISO 25010, the standardization of software quality models for system and software quality requirements and evaluation [7]. In particular, the analysis will rely on the characteristics functional suitability, performance efficiency, compatibility, usability, reliability, security and maintainability.

3.3 Metrics

When defining metrics for DevOps, one needs to be very careful considering a possible fallback. Especially inter- and intra-team metrics should not be used as this may lead to revealed teams and thus inhibit collaboration. Furthermore, traditional metrics like mean time to failure are at least questionable if they are not considered in reference to other indicators as deployment quantity, especially when a team just starts to implement DevOps. In addition, vanity metrics as lines of code are also only useful if one also considers validation and test coverage of this piece of code [14].

In the following, the tables containing metrics derived from literature are described and examples on how these have been developed are given.

Table 1 contains the questions and suitable metrics for the first goal of analysing the DevOps process. Moreover, the last column includes the sources of these metrics or the idea behind these.

Riley described metrics like change lead times, change failure rates or mean times to recover in [16]. Considering they are already described in a measurable way, they can simply be added to the first section meassuring the flow of value to the customer as changes should always deliver new features to the customer or at least recover them after failures.

In [8], Juner and Benlian described five capability areas. One of their suggested competences focused on metrics available in the process. For example, they brought up appropriate logging, montoring and alerting capabilities. Since monitoring and logging systems provide feedback to developers on their implementations, this capability is mentioned by a metric asking for the availability of such systems to specify the question for flow of feedback. On the other hand, they described a knowledge exchange by code reviews. This is represented by the suitable metric in the continual learning and experimentation section.

For the pipeline view, Table 2 contains the product chararcteristics of ISO 25010 instead of questions. The following columns are structured the same way like in the previous table.

The continuous delivery maturity matrix described by Gupta in [4] contains many capabilities that can be implemented by a continuous delivery pipeline. For example, artifact tagging, static code metric generation or zero-downtime deployments can be seen as features of the pipeline and are thereby considered when evaluation the functional suitability characteristic of the pipeline. Moreover, equal and orchestrated deployments to different environments can be seen as compatibility features as they allow a certain flexibility on deployments. Numerous tests on build, integration or performance level ensure the quality of deployments and thereby generate reliability on the pipeline and its deployments. The classification of Infrastructure as Code (IaC) is not that clear. On the one hand, IaC generates compatibility and flexibility when deploying to different systems. On the other, IaC also supports maintainability as all systems are provisioned in an equal way. Nevertheless, IaC here is only mentioned in the reliability section as an equal provisioning allows other pipeline steps to rely on this infrastructure and mentioning IaC and other similar cases in every section would blow up the table.

4. DISCUSSION

Table 1 brought up many metrics to cover the three ways of DevOps. For the flow of value to the customer it is fundamental to deliver developed features to the customer as fast as possible. Consequently, the metrics focus on the number of features deliverd, the time a feature needs to be delivered or the number of releases to deliver these features.

The flow of feedback mainly focuses on available metrics and monitoring systems covering test and production environments. Furthermore, the flow of feedback from customer to development is mentioned.

For the last point of continual learning and experimentation the metrics assess the teams reflection, adaption and reaction capabilites.

Observing Table 2 one notes a lack of metrics in the areas usability and security.

First of all, the development and deployment pipeline is normally a tool that is only used internal. Thus, it seems that only few pay attention to usability characteristics of it. At the moment the teams seem to pay most attention to functional capabilities, reliability and performance issues to even reach a continuous deployment environment. Hence, usability is a point that should be observed in the future.

Secondly, the security of these pipeline systems is not mentioned that often. The write access to version control systems should normally be restricted to people involved in the project and thereby cover integrity, accountability and authenticity of changes. Commit hooks are thereby also restricted. However, it is not said if the access to pipeline systems and their capabilities to change production environments is restricted.

A lack of metrics on the compatibility characteristic may also be observed but as described earlier, this may be solved by e.g. adding metrics considering Infrastructure as Code to multiple characteristics in the table.

5. CONCLUSION

First of all, this paper analyzed existing literature on quality models covering DevOps, Continuous Delivery and Continuous Deployment based on different meta models. The suggested quality model of this paper thereby resulted from an aggregation of numerous existing quality models or at-

Table 1: DevOps process quality

Question	Metrics	Source
What is the current performance of the development and deployment process regarding flow of value to the customer?	What is the number of releases per month?	[10]
	How many features are released per month?	[10]
	What are the costs per release?	[14]
	What is the average lead time of a feature?	[6]
	What is the average cycle time of a feature?	[18]
	What is the average change lead time?	[8, 16]
	What is the fastest possible feature lead time?	[10]
	What is the current revenue per user story?	[14]
	How old is the oldest done feature which is already developed and tested but not deployed to production yet?	[10]
	What is the mean time to repair a detected failure?	[16]
	What is the current change failure rate?	[16]
	What is the average badge size?	[6]
	Are teams organized around products or KPIs?	[12]
What is the current performance of the development and deployment process regarding flow of feedback?	What is the mean time to detect a failure?	[6]
	Does the development team have access to logs and stacktraces of the production systems for debug purpose?	[8]
	What is the number of incidents as a result of a feature release?	[18]
	Are metrics on system, availability and performance available?	[8]
	Does the monitoring system alert appropriate persons on failures?	[8]
	Does every person in the organization have access to visualized feedback and metrics of all systems?	[8, 4, 15]
	What is the average and maximum time between customer touch points?	[18]
What is the current performance of the development and deployment process regarding continual learning and experimentation?	Are code reviews performed to gain a uniform understanding?	[8]
	How much time is spent on rework after a feature release?	[18]
	What amount of time is used to store new knowledge?	[18]
	What amount of time does it take to retrieve knowledge acquired earlier?	[18]
	How much time does the team spend on reflection of their work after a project or sprint?	[18]
	How much time does the team spend on reflection of their working process?	[18, 15]
	Which percentage of deliberate introduced faults for experimentation does the team discover?	[18]
	Do people collaborate across team borders?	[4, 15]

tributes of the three topics mentioned. In order to cover the DevOps idea as good as possible the model considers the process view as well as the technical implementation of DevOps processes. The process view mainly focuses on the way developers work together to deliver value to the customer, whereas the technical implementation considers the pipeline model and tools in general. However, these two views are not seperated as they affect each other.

As the metrics of this model are still very diverse, this also shows the difficulties when clearly defining the DevOps approach again.

As the model is not yet verified this is some point to focus on in further work. To complete the GQM-approach one will have to apply the quality model in a case environment, collect data and interview material of involved stakeholders. Afterwards the model may be adapted according to the results of this case study.

6. REFERENCES

[1] V. R. Basili, G. Caldiera, and H. D. Rombach. The Goal Question Metric Approach. 1994.

[2] R. de Feijter. *Towards the adoption of DevOps in software product organizations: A maturity model approach.* Master Thesis, Universiteit Utrecht, Utrecht, 23.05.2017.

[3] Forester Consulting. Continuous Delivery: A Maturity Assessment Model. 2012.

[4] A. Gupta. Continuous Integration, Delivery, Deployment and Maturity Model, 2015. `http://blog.arungupta.me/continuous-integration-delivery-deployment-maturity-model/`.

[5] J. Humble. There's No Such Thing as a Devops Team, 2012. `https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/`.

[6] M. Hüttermann. *DevOps for developers.* Books for professionals by professionals. Apress, New York, NY, 2012.

[7] ISO/IEC. Systems and software engineering —

Table 2: Technical implementation quality

Characteristic	Metrics	Source
functional suitability	To what degree does the pipeline support a fully automated deployment?	[8]
	Are artifacts tagged and managed appropriate?	[11][4, 15]
	Are static code metrics (CheckStyle, Sonar, FindBugs) generated?	[8][11][4][15]
	Does the pipeline allow deployment roll-backs?	[11]
	Does the architecture support zero-downtime deployments?	[8][3][4][15]
	Are deployments disconnected from releases?	[4, 15, 6]
	Does the pipeline support self-healing mechanisms on failures?	[12]
performance efficiency	What is the average lead time of a feature?	[6]
	What is the average cycle time of a feature?	[18]
	What is the fastest possible feature lead time?	[10]
	What is the mean time to repair a detected failure?	[16]
	How long does it take to deploy a new feature?	[8]
compatibility	Are artifacts deployed to any environment in an equal way?	[4, 15]
	Are deployments orchestrated?	[4][15]
usability	Are testresults and metrics illustrated in a graphical way?	[8, 15]
reliability	What is the current change failure rate?	[16]
	Are database changes automated and versioned in an auditable way?	[3, 4]
	Are automated tests performed on build level in a continuous way?	[8]
	Are integration tests run in a continuous way?	[8, 4, 15]
	Are performance tests run in a continuous way?	[8, 4]
	Are deployments scripted and automated?	[8]
	Are system tests performed after every deployment automatically?	[8]
	Is the infrastructure provisioned by versioned *Infrastructure as Code*?	[8, 4]
security	Are changes to sourcecode or Infrastructure as Code auditable?	[8]
	Are pipeline runs traceable and accountable?	[8][4, 15]
maintainability	Is the architecture component based or orchestrated?	[4, 15]
	Is an appropriate library and API management used?	[15]

Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models, 01.03.2011.

[8] C. Juner and A. Benlian. Praxisbasierte Capability-Modelle für DevOps-Einsätze in Unternehmen. *HMD Praxis der Wirtschaftsinformatik*, 54(2):230–243, 2017.

[9] G. Kim, P. Debois, J. Willis, J. Humble, and J. Allspaw. *The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations.* IT Revolution Press, Portland, OR, first edition edition, 2016.

[10] T. Lehtonen, S. Suonsyjä, T. Kilamo, and T. Mikkonen. Defining Metrics for Continuous Delivery and Deployment Pipeline. *SPLST*, 2015, 2015.

[11] R. Mills. A Maturity Matrix for Continuous Delivery Pipelines, 2014. https://www.coveros.com/a-maturity-matrix-for-continuous-delivery-pipelines/.

[12] J. Parks. The Solinea DevOps Maturity Model, 2016. https://solinea.com/blog/solinea-devops-maturity-model.

[13] Puppet + DORA. State of Devops Report 2017. 2017.

[14] A. Ravichandran, K. Taylor, and P. Waterhouse. *DevOps for Digital Leaders.* Apress, Berkeley, CA, 2016.

[15] A. Rehn, T. Palmborg, and P. Boström. The Continuous Delivery Maturity Model, 2013. https://www.infoq.com/articles/Continuous-Delivery-Maturity-Model.

[16] C. Riley. Metrics for DevOps. https://devops.com/metrics-devops/.

[17] G. Rong, H. Zhang, and D. Shao. CMMI Guided Process Improvement for DevOps Projects: An Exploratory Case Study. In D. E. Perry and D. Raffo, editors, *Proceedings of the International Workshop on Software and Systems Process - ICSSP '16*, pages 76–85, New York, New York, USA, 2016. ACM Press.

[18] J. Trienekens. Towards a Metrics Model for DevOps: Results of a Case Study in an Industiral Company. In IARIA, editor, *FASSI 2015*, pages 1–6, 2015.

Tool Support for Collaborative UML Modelling

Nina Rußkamp
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
nina.russkamp@rwth-aachen.de

Ana Nicolaescu
RWTH Aachen University
Software Construction
Ahornstr. 55
52074 Aachen, Germany
ana.nicolaescu@swc.rwth-aachen.de

ABSTRACT

For a couple of years, collaboration has been taking an increasing part in software engineering. By time, a considerable number of digital tools, claiming to support collaborative software engineering in a natural way, has been developed. However, it is questioned whether those tools take into account the requirements and characteristics of collaborative work and facilitate software engineering effectively. In this paper, contribution and benefits of collaborative software tools are investigated, focusing on collaborative model creation. Comparing collaborative UML modelling tools with each other by means of the concepts of collaboration and UML modelling, it turns out that they do not follow a holisitic apporach, but only provide restricted support regarding collaboration types, levels of formality, model types (and therefore stages of software desing) and input modalities.

Categories and Subject Descriptors

D.2. [**Software**]: Software Engineering

Keywords

collaborative software engineering (CoSE), collaboration tools, software design, UML

1. INTRODUCTION

By time, collaboration has established as an essential part of software engineering [1]: It serves not only as means to face the issue of increasing complexity of software systems [4], but also provides validation for software solutions in process [17]. To turn all benefits of collaboration to account, various software tools have been developed, claiming to support collaborative software engineering in a natural way. However, it is questioned whether those tools take into account the requirements and characteristics of collaborative work and facilitate software engineering effectively [21].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SWC Seminar 2017/18 RWTH Aachen University, Germany.

Therefore, the objective of this paper is to examine how far existing collaborative UML modelling tools contribute to software engineering. Although collaboration is applicable to every stage of the software development lifecycle, we concentrate on collaborative software design, especially on collaborative modelling because modelling a software problem is at the core of finding an appropriate solution for it [30]. Additionally, modelling in general and UML modelling in specific have proved to contribute to both the productivity of software engineers and the quality of software solutions [10], [23].

The paper is structured as follows: In the next section, we will first define collaboration, clarify different types of collaborating and characterise the process of modelling. Ensuing, OctoUML is presented as an example of existing collaborative UML modelling tools and compared with alternative tools on the basis of the presented collaboration and modelling characteristics. Afterwards, the results are concluded.

2. COLLABORATIVE MODELLING

As the term *collaborative modelling* suggests, it consists of *collaboration* and *modelling*. That is the reason why both concepts are explained separately first and then brought together to give an understanding of collaborative modelling in software engineering.

2.1 Collaboration

First, we define collaboration by explaining the etymology of the term *collaboration*. Then the concept of collaboration is considered in detail describing its characteristics and appearances.

2.1.1 Etymology

The term *collaboration* comes from the Latin expression *cum laborare* which means *to work along side one another* [20]. Nowadays, English dictionaries provide different meanings of collaboration: The concept of collaboration is either understood as a *process* or a *product* or a *state*. Collaboration as a *process* terms the action of working together, whereas collaboration as a *product* denotes the outcome of such an activity. Collaboration as a *state* refers to the mind setting of people, e.g. their willingness to pursue a shared goal [17].

2.1.2 3C Model

According to the 3C collaboration model [13], collaboration is based on the concepts of *communication*, *coordination* and *cooperation*: *Communication* serves the exchange of in-

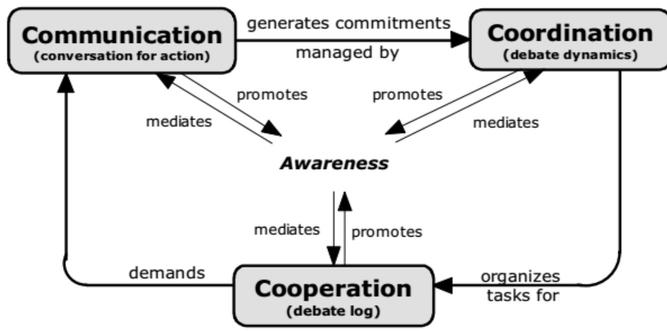


Figure 1: Inter-relationship between the 3Cs [15]

formation whereby information are understood as contextualised data [19]. *Coordination* is necessary to organise the collaborative work by dividing tasks into subtasks, assigning these subtasks to different workers, managing the time schedule, and collecting results [27]. *Cooperation* describes the phase of advancing the software solution via a shared workspace [15].

The three concepts are interdependent which means that there are both inter- and intra-relationships between them. The intra-relationship describes for each C how all three Cs are represented inside one C interacting within themselves, for instance, in how far a communication unity of a collaboration system also supports coordination and cooperation. The inter-relationship is depicted by the example of collaborative group work (see figure 1). To start working together in a group, it is necessary that the group members get in contact with each other (*communication*). During their conversation the group members start a discussion how to organize their task (*coordination*) resulting in joint activities (*cooperation*), that require another communication event to eleminate uncertainties and so on. Thus, the inter-relationship between communication, coordination and cooperation describes the cycle of their resultant sequences. Consequently, collaboration is inherently iterative [15].

2.1.3 Definition of Collaboration

In addition to these etymological definitions, creating a value as a group that could not be accomplished by an inidvidual person is emphasised to be the unique selling point of collaboration [26], [20]. Therefore, we define collaboration as an iterative process of organisational and productive working stages according to the 3C model at least two persons are involved in to achieve a joint goal that a single person is not able to reach due to immutable limitations (such as time).

2.1.4 Classification of Collaboration Types

By reference to the main distinguishing characteristics, different collaboration types can be identified. Maher et al. [22] define three types of collaboration to specify how many software designers are working at a time and if they are contributing to the same or distinct subtasks. People *collaborate mutually* if they are working on the same task simultaneously. Working alone on different subtasks (except the case one needs advice to continue) is called *exclusive collaboration*. *Dictator collaboration* is applied whenever a working group determines one person who takes over control of the working process [20].

Hildenbrand et al. [17] found five factors influencing collaborative software engineering including *geographical* and *temporal* distribution. Regarding the *geographical* distribution, collaborative software engineering teams work either *distributed* or *co-located*. The *temporal* distribution describes whether the team members work simultaneously (*synchronous* collaboration) or independently of time (*asynchronous* collaboration) [17].

2.2 Modelling

Modelling is a complex process of adressing software problems and drawing near an appropriate software solution iteratively. Thus, creating models contributes to bring up a solution approach that is refined step by step and therfore bridges the gap between a problem statement and the final software product [30]. Software models can be classified on the basis of the stage of development, the abstraction level and the level of formality: As modelling is used in different phases of the software engineering lifecycle, models can be classified by the stage of development they are created in [9]. To solve software problems, software designers structure and divide their task, operating on different abstraction levels which leads to models of different levels of detail [32], [16]. According to the stage of development and the abstraction level, software designers apply informal or formal notations. In the beginning, software designers often deal with a modelling task by free-hand sketching while they use the UML standard [28] later on [11]. Consequently, models need to be edited again and again during their creation process, whereby quite often many software designers participate in modelling simultaneously [10].

2.3 Collaboration Connected with Modelling

Collaborative UML modelling is defined as an iterative process including communication, coordination and cooperation at least two persons are involved in to create a UML model and pursue a joint goal.

3. COLLABORATIVE UML MODELLING TOOLS

Referring to Briggs et al. [7], we define collaborative modelling tools as artifacts that support a collaborative team in the joint modelling process and contribute to the achievement of shared goals.

Existing collaboration tools in (model-driven) software engineering have already been classified. To the best of our knowledge, Whitehead [31] was the first who classified collaboration tools. He identified four categories, i.a. covering model-based collaboration tools. Further studies followed in the last ten years (see [21], [25], [12]).

Recently, Franzago et al. [14] have published an extensive classification map of model-driven collaboration tools classifying those i.a. by means of collaboration characteristics.

In contrast to the preceded studies, we do not aim at giving a broad overview of collaboration tools but limit our object of research to collaoborative tools that provide the possibility to create UML models. Thereby we intruduce the main features of the modelling tool OctoUML and competing tools.

3.1 OctoUML

OctoUML claims to support collaborative UML modelling

during any phase of the software design process. To meet the requirements of all modelling phases, OctoUML allows using both formal and informal notations to create diagrams. Formal notations are computer-drawn diagram elements corresponding to the UML standard. Currently, creating models with the formal notation is limited to UML class and sequence diagrams. Informal notations denote hand-drawn sketches that are not subjected to any restrictions of the UML modelling language. As OctoUML provides different visual layers for each type of notation, the user can choose whether formally and informally notated models are shown simultaneously or separately. Furthermore, it is possible to transform informal notations to formal diagrams (see figure 2). OctoUML can be run on different devices such as desktop computers, touch screens and interactive whiteboards. To interact with the modelling environment, users can use multiple input modes: mouse, keyboard, touch/multi-touch using finger and styluses, sketching and voice commands. The multi-touch input modality is a key factor for in-situ modelling as it enables different users to work on the same model simultaneously. It is also possible to work in distributed teams, using a session-based client-server paradigm. Referring to the collaboration types defined in section 2.1.4, OctoUML supports the following collaboration types: Concerning the graphical distribution, OctoUML enables collaborative teams to work either co-located or distributed. In each case, all team members are supposed to work synchronously.

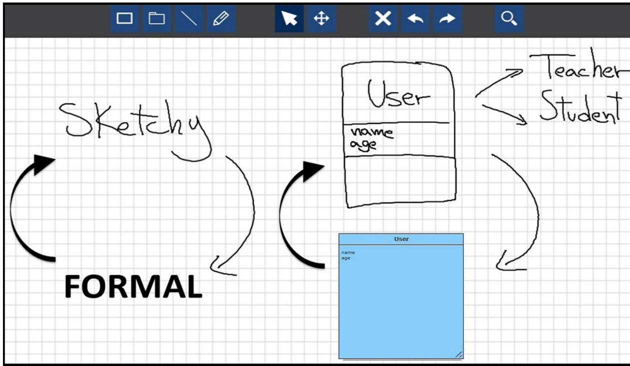


Figure 2: OctoUML: formal and informal notations [30]

3.2 Alternatives to OctoUML

As alternatives to OctoUML not only tools that seem to be similar to OctoUML were chosen, but also tools that highly differ from OctoUML are contrasted to it. Table 2 gives an overview of OctoUML and its alternatives. The distinctive features of the alternative tools are named as well as assigned to the collaboration types as defined in section 2.1.4.

Apart from OctoUML, at least one more collaborative UML modelling tool using multi-touch as interaction modality has been presented recently [6]. This tool - called MT-CollabUML - has a smaller range of functions than OctoUML as it does not allow users to collaborate remotely, viz. synchronous co-located collaboration is supported while asynchronous and/or distributed collaboration is not.

COLLECT-UML [3] is an educational modelling tool for UML class diagrams that first let students design a model solution on their own before joining the collaborative modelling workspace. In contrast to OctoUML and MT-CollabUML, COLLECT-UML can only support distributed collaboration to make sure that every student comes up with an own solution before starting the collaboration session. Collaborative sessions follow the principal of dictator collaboration: At one and the same time, there is only one user who has the pen and is allowed to edit the model. As input modalities mouse and keyboard are used. Furthermore, the tool supports communication and coordination: Users can communicate via chat. To organize the chat protocol, all messages have to be assigned to a communication category before they are sent. The coordination of the collaborative sessions is is done by showing which users have already joined the session and who owns the pen at the moment. The interface of the tool is divided into three parts: a private workspace, a shared workspace and a chat window [2].

Table 1: Graphical and Temporal Distribution

Tool	*Graphical Distribution (c, d)*	*Temporal* Distribution *(s, a)*
OctoUML	(yes, yes)	(yes, no)
MT-CollabUML	(yes, no)	(yes, no)
CAMEL	(no, yes)	(yes, no)
COLLECT-UML	(no, yes)	(yes, no)

Abbreviations: c: co-located collaboration, d: distributed collaboration, s: synchronous collaboration, a: asynchronous collaboration

Similar to COLLECT-UML is CAMEL [8]; CAMEL allows distributed and dictator collaboration (except from the whiteboard are where all users can sketch free-hand diagrams simultaneously), too. In contrast to COLLECT-UML, CAMEL does not require that the users first solve modelling tasks on their own. Instead, CAMEL uses the private workspace as a local copy of the shared workspace to enable the users to view any part of the model at any time. Both formal notations (namely UML elements of class, sequence, state machine, activity and use case diagrams) and informal notations (free-form sketches) via mouse and keyboard are accepted as input. In addition, CAMEL enables collaboration teams to coordinate the modelling process; there is a list of all session members. Furthermore, different colours are assigned to the members to show which part of the model has been changed by whom. The session members can communicate via chat during the modelling process as well as record and playback the modelling process. The interface of CAMEL is shown in figure 3. Table 1, table 3 and table 4 show which characteristics are covered by the different tools. The inter- and intrarelationships between communication, coordination and cooperation (as defined in section 2.1.2) of the tools are depicted in figure 4.

4. DISCUSSION

In this section it is discussed how far the investigated tools support the process of collaborative UML modelling. Thereby we apply different criteria, similar to the collaboration characteristics of Franzago et al. [14]. Most of the criteria are derived from section 2.1.4 and section 2.2. Ad-

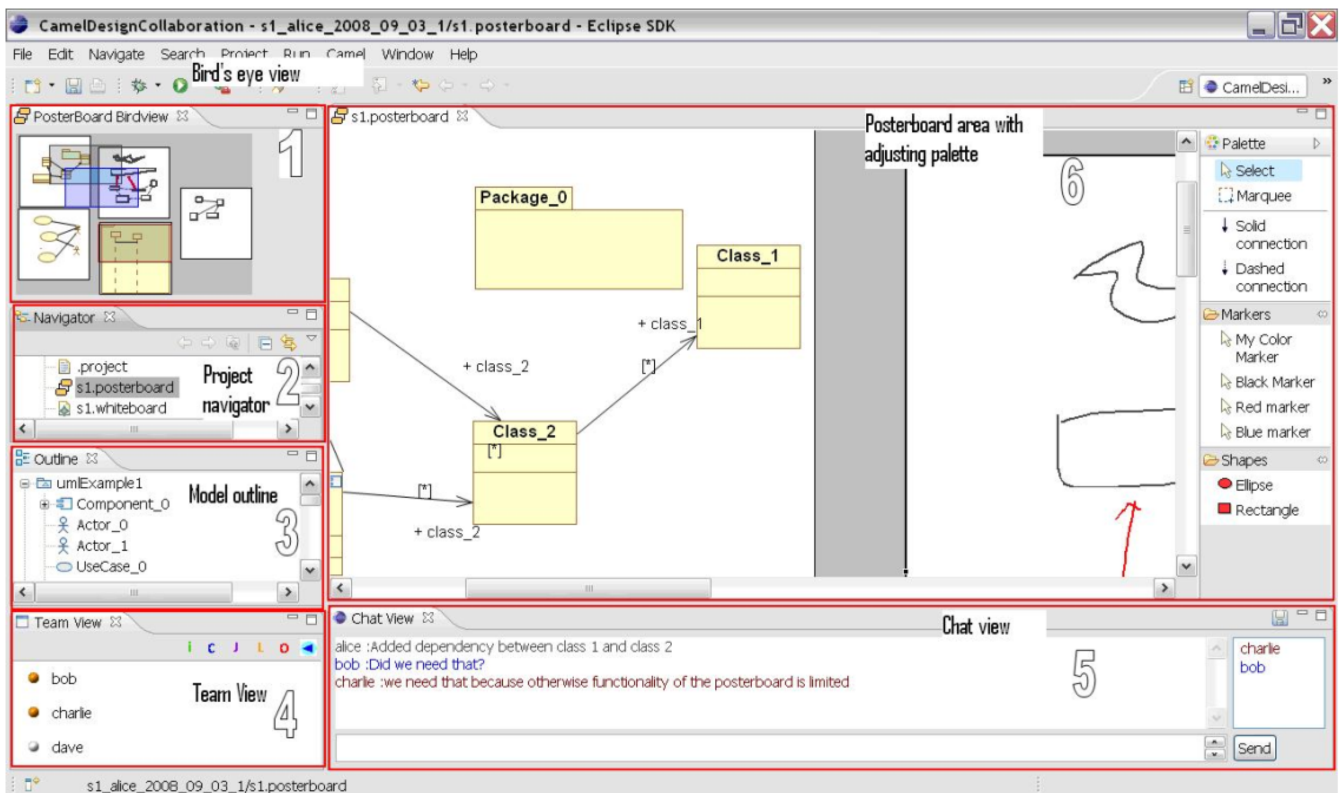


Figure 3: Interface of CAMEL [8]

Table 2: Collaborative Modelling Tools

Tool	*Year*
OctoUML [30]	2017
MT-CollabUML [6]	2012
CAMEL [8]	2009
COLLECT-UML [3], [2]	2005, 2007

Table 3: Interaction Modality User - Tool

Tool	*Interaction Modality*
OctoUML	mouse, keyboard, single touch, multi-touch, voice, sketch
MT-CollabUML	mouse, keyboard, single touch, multi-touch
COLLECT-UML	keyboard, mouse
CAMEL	keyboard, mouse, sketch

ditionally, we regard the interaction mode between user and tool, meaning which input modalities the user can use to operate the tool. This characteristic is chosen because multi-touch has recently gained interest in the context of collaborative modelling tools. Looking at the temporal distribution, none of the investigated tools provide asynchronous collaboration (except the pre-phase in which students work on their own using COLLECT-UML or CAMEL). Applying synchronous collaboration, the tools offer different options regarding the geographical distribution. OctoUML is the only tool that provides an environment for both distributed and co-located collaboration whereas MT-CollabUML is restricted to co-located collaboration. Altough it is possible to overcome spatial distances, the qualification of OctoUML for international software projects is restricted as time shifts complicate the appointment for synchronous work of spatially distributed teams.

Furthermore, the remote version of the tool to be used by distributed teams is realized by a client-server-paradigm operated with mouse and keyboard or single-touch whereas co-located collaboration is done by using touch screen devices or interactive whiteboards allowing multiple users to sketch with their fingers or a stylus simultaneously on the same canvas and use voice commands. To benefit from the wide range of input modalities, it seems to be more likely to use OctoUML for co-located collaboration.

In contrast, COLLECT-UML and CAMEL are only designed for distributed collaboration. For these tools the choice of distributed collaboration is reasonable as the learning concept intends students to first solve a modelling task on their own (the co-location would be obstructive in this case) and then use dictator collaboration which does not require co-location and maybe is even easier to be conducted in geographically distributed teams as the system locks every unauthorized intervention (which cannot be quaranteed in natural interaction situations).

In addition, Basheri et al. [5] compared a PC-based version with mouse and keyboard as input devices with the multi-touch version of their tool. They found out that group members participate more equally in modelling tasks when using multi-touch screens instead of PCs.

Table 4: Concurrency

Tool	*Concurrency (m, e, d)*
OctoUML	(yes, no, no)
MT-CollabUML	(yes, no, no)
COLLECT-UML	(no, no, yes)
CAMEL	(no, no, yes)

Abbreviations: m: mutual collaboration, e: exclusive collaboration, d: dictator collaboration

Regarding OctoUML and MT-CollabUML, it can be observed that multi-touch input seems to co-occur more likely with mutual collaboration than with exclusive or dictator collaboration as team members do not need to exchange input devices before working on a modelling task. Using keyboard and mouse, however, hinders mutual collaboration. Instead, the members of collaborative teams use dictator collaboration to work with mouse and keyboard on modelling tasks. Although dictator collaboration is more suitable to PC-based modelling situations than mutual collaboration, the combination of dictator collaboration and a PC-based setting leads to a lower productivity during the collaboration process than the combination of a multi-touch device and mutual collaboration[5]. Contrary to this observation, Rummel et al. [29] claim turn-taking (as called dictator collaboration in this article) to be a characteristic of good collaboration.

Relating to the classification of the tools based on the 3C model, it becomes obvious that the collaborative UML modelling tools differ in the weighting of communication, coordination and cooperation. OctoUML and MT-CollabUML are limited to cooperation as communication and coordination can be done by face-to-face interaction during the synchronous and co-located collaboration. It is planned to integrate new features for communication and coordination to improve distributed collaboration using OctoUML [18]. CAMEL and COLLECT-UML also include coummunication and coordination (see figure 4).

Olsen et al. [24] state that almost 60% of collaboration time is spent on coordination. The study of Basheri et al. [6] verifies this and, beyond that, provides study results proving that in multi-touch collaboration sessions less time is spent on discussing than in PC-based collaboration sessions. So maybe face-to-face coordination is more effective than coordination guided by the tool.

Looking at the process of modelling, only one tool (CAMEL) supports all UML diagram types while the others only offer the creation of class (and sequence) diagrams. As many users claim the importance of informal sketching in addition to UML standard notation [18], this feature is proposed by more than one tool.

5. CONCLUSIONS AND FUTURE WORK

As the result of our discussion, we identify two different types of tools providing a coherent set of collaboration types and input modalities: The first type of tools supports co-located users working synchronously and mutually together on multi-touch devices. This type of tool does without any additional communication or cooperation component

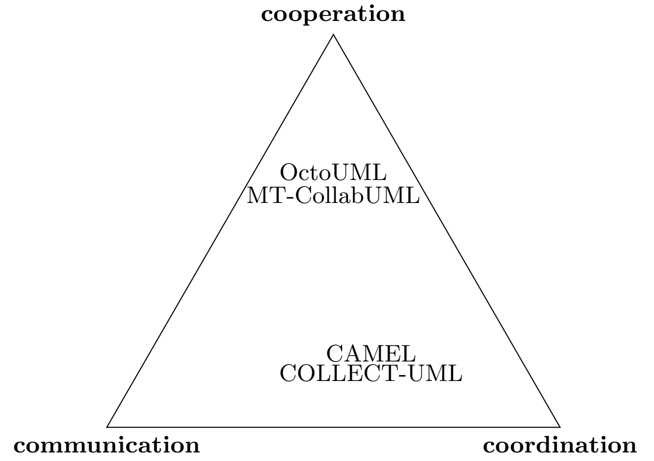


Figure 4: Tool classification based on the 3C model

as users are supposed to organize their work in a natural face-to-face meeting. The second type of tools is aimed at distributed collaboration providing communication and coordination structures to guide the users over the course of their collaboration process. Due to its technical support of dictator collaboration, this type of tool can be used for educational purposes.

These two examples show that collaborative UML modelling tools still do not follow a holisitic apporach, but only provide restricted support regarding collaboration types, levels of formality, model types (and therefore stages of software desing) and input modalities. Nevertheless the two classes of tools can be convenient for particular purposes, depending on the application domain.

All in all, collaboration is a complex concept that has the potential to improve both the process and the product of model creation in software design. The process of collaborative modelling underlies numerous varying factors. Designing collaborative UML modelling tools, these characteristics and their interdependencies have to be taken into account to make the tool useful. In future research, the application domain (e.g. industry and education) of the collaborative UML modelling tools has to be considered as influencing factor of collaboration characteristics.

6. REFERENCES

[1] N. Ahmadi, M. Jazayeri, F. Lelli, and S. Nesic. A survey of social software engineering. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 1–12, Piscataway, NJ, 2008. IEEE.

[2] N. Baghaei. A collaborative constraint-based intelligent system for learning object-oriented analysis and design using UML. 2007.

[3] N. Baghaei and A. Mitrovic. COLLECT-UML: Supporting individual and collaborative learning of UML class diagrams in a constraint-based intelligent tutoring system. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 458–464, 2005.

[4] C. Barlelt, G. Molter, and T. Schumann. A model repository for collaborative modeling with the jazz

development platform. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–10, 2009.

[5] M. Basheri, L. Burd, and N. Baghaei. A multi-touch interface for enhancing collaborative UML diagramming. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, pages 30–33, 2012.

[6] M. Basheri, L. Burd, and N. Baghaei. Collaborative software design using multi-touch tables. In *Engineering Education (ICEED), 2012 4th International Congress on*, pages 1–5, 2012.

[7] R. O. Briggs, G. Kolfschoten, G.-J. d. Vreede, C. Albrecht, D. R. Dean, and S. Lukosch. A seven-layer model of collaboration: Separation of concerns for designers of collaboration systems. *ICIS 2009 Proceedings*, page 26, 2009.

[8] M. Cataldo, C. Shelton, Y. Choi, Y.-Y. Huang, V. Ramesh, D. Saini, and L.-Y. Wang. CAMEL: A Tool for Collaborative Distributed Software Design. In *2009 Fourth IEEE International Conference on Global Software Engineering*, pages 83–92. IEEE, 2009.

[9] M. R. V. Chaudron, A. Fernandes-Saez, R. Hebig, T. Ho-Quang, and R. Jolak. Diversity in UML Modeling Explained: Observations, Classifications and Theorizations. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 47–66, 2018.

[10] M. R. V. Chaudron, W. Heijstek, and A. Nugroho. How effective is UML modeling? *Software & Systems Modeling*, 11(4):571–580, 2012.

[11] M. R. V. Chaudron and R. Jolak. A Vision on a New Generation of Software Design Environments. In *HuFaMo@ MoDELS*, pages 11–16, 2015.

[12] K. Dullemond, B. van Gameren, and R. van Solingen. Collaboration Spaces for Virtual Software Teams. *IEEE Software*, 31(6):47–53, 2014.

[13] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):39–58, 1991.

[14] M. Franzago, D. Di Ruscio, I. Malavolta, and H. Muccini. Collaborative Model-Driven Software Engineering: A Classification Framework and a Research Map. *IEEE Transactions on Software Engineering*, page 1, 2017.

[15] H. Fuks, A. Raposo, M. A. Gerosa, M. Pimentel, D. Filippo, and C. Lucena. Inter-and intra-relationships between communication coordination and cooperation in the scope of the 3C Collaboration Model. In *Computer Supported Cooperative Work in Design, 2008. CSCWD 2008. 12th International Conference on*, pages 148–153, 2008.

[16] R. Guindon. Designing the design process: Exploiting opportunistic thoughts. *Human-Computer Interaction*, 5(2):305–344, 1990.

[17] T. Hildenbrand, F. Rothlauf, and A. Heinzl. Ansätze zur kollaborativen Softwareerstellung. 2006.

[18] R. Jolak, B. Vesin, M. Isaksson, and M. R. V. Chaudron. Towards a New Generation of Software Design Environments: Supporting the Use of Informal and Formal Notations with OctoUML. In *HuFaMo@ MoDELS*, pages 3–10, 2016.

[19] T. Karle and A. Oberweis. Unterstützung von Kollaboration im Rahmen der Softwareentwicklung auf Basis Service-orientierter Architekturen. In *EMISA*, pages 77–90, 2006.

[20] T. Kvan. Collaborative design: What is it? *Automation in Construction*, 9(4):409–415, 2000.

[21] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino. Collaboration Tools for Global Software Engineering. *IEEE Software*, 27(2):52–55, 2010.

[22] M. L. Maher, A. Cicognani, and S. Simoff. An experimental study of computer mediated collaborative design. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1996. Proceedings of the 5th Workshop on*, pages 268–273, 1996.

[23] A. Nugroho and M. R. V. Chaudron. A survey into the rigor of UML use and its perceived impact on quality and productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 90–99, 2008.

[24] G. M. Olson, J. S. Olson, M. R. Carter, and M. Storrosten. Small group design meetings: An analysis of collaboration. *Human–Computer Interaction*, 7(4):347–374, 1992.

[25] J. Portillo-Rodríguez, A. Vizcaíno, M. Piattini, and S. Beecham. Tools used in Global Software Engineering: A systematic mapping review. *Information and Software Technology*, 54(7):663–685, 2012.

[26] N. Randrup, D. Druckenmiller, and R. O. Briggs. Philosophy of Collaboration. In *System Sciences (HICSS), 2016 49th Hawaii International Conference on*, pages 898–907, 2016.

[27] A. Rashid, A. Behm, M. Geisser, and T. Hildenbrand. Kollaborative Softwareentwicklung–zum Kollaborationsbegriff. 2006.

[28] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language reference manual ; [UML]*. /The Addison-Wesley object technology series]. Addison-Wesley, Boston, 2. ed. edition, 2005.

[29] N. Rummel and H. Spada. Learning to Collaborate: An Instructional Approach to Promoting Collaborative Problem Solving in Computer-Mediated Settings. *Journal of the Learning Sciences*, 14(2):201–241, 2005.

[30] B. Vesin, R. Jolak, and M. R. V. Chaudron. OctoUML: An environment for exploratory and collaborative software design. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 7–10, 2017.

[31] J. Whitehead. Collaboration in Software Engineering: A Roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 214–225, Washington, DC, USA, 2007. IEEE Computer Society.

[32] C. Zannier, M. Chiasson, and F. Maurer. A model of design decision making based on empirical results of interviews with software designers. *Information and Software Technology*, 49(6):637–653, 2007.

Does Agile Software Development need Architecture?

Lukas Schade
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
lukas.schade@rwth-aachen.de

Christian Plewnia
RWTH Aachen University
Research Group Software Construction
Ahornstr. 55
52074 Aachen, Germany
christian.plewnia@swc.rwth-aachen.de

ABSTRACT

The advent of agile software development has changed the way we create software. Ideally, it helps to ship software faster and makes it easier to react to changes in the requirements. However, it is often misunderstood how to apply agility to software architecture. The lack of a common understanding of software architecture plays a role here. This can lead to confusing and unmanageable architecture, which reduces the advantages of agile software development.

This paper pursues the question, whether software architecture is a concern that needs to be considered when developing software in an agile way. Furthermore, we investigated whether there are special characteristics of architecture which are especially supportive in agile development. In sum, the result of our research is that it does not need a big planning phase upfront. Instead architecture is developed iteratively and emerges during development. For this, the architecture has to fulfill some requirements.

Keywords

software architecture, agile software development

1. INTRODUCTION

In this paper we want to research the role of software architecture in agile software development. The role of software architecture in agile software development in not as clear as it might seem to be at the first sight. There are many different definitions for software architecture. Often it seems to be related to big design upfront (BDUF). This does not fit with the agile principles. As the agile manifesto [5] says that working code is more important than comprehensive documentation, an upfront designed blueprint of the software architecture is incompatible with agile software development. But there are developers who say software architecture is just as important in agile projects than in non-agile projects [10]. How does this fit together?

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SWC Seminar 2017/18 RWTH Aachen University, Germany.

In the first part we deal with the question if software architecture plays a role in agile software development. We want to find out what the problems are, that lead to the assumption software architecture is BDUF and therefore incompatible with agile software development and if there are solutions for them. For this we take a closer look at what is understood by software architecture in general. We will see, that there are many different understandings. This may be one of these problem.

Assuming software architecture plays a role in agile software development projects, we are interested in whether there are *architectural properties* that support or impede agile software development. By architectural properties we mean specific characteristics or attributes of the architecture that have an impact on the compatibility with agile software development. This way, they affect how the architecture reduces or supports the advantages of agility. To find such properties, we first study what to take care of and what is important in agile software development with regard to software architecture. We use the results to find characteristics that are responsible for the compatibility with agile software development.

The results in this paper are based on a literature research. First, we collect information about the context of the research question from various sources. Resources like scientific papers, but also experience reports and tips from developers regarding software architecture in agile software development are interesting here. Then, the gathered material is analyzed and discussed. In doing so, we connect and compare different sources. We check if the opinions are consistent or if there are maybe contrary positions.

Section 2 starts with a list of different definitions of software architecture. We will see that there is no common understanding what software architecture is and what it is not. Out of these definitions, we formulate two different understandings. In the sections 2.1 and 2.2 we then study the first research question, each section depending on one of the two understanding of architecture.

In section 3 we assume software architecture plays a role in agile software development. We investigate if there are any properties of architecture that support agile development. If there are any, we want to work these properties out.

Finally a conclusion is given. We critically discuss the value of the results and give suggestions for further research.

2. SOFTWARE ARCHITECTURE VS. AGILITY

Is software architecture compatible with agile software development? That is the question we want to answer in this section. The reason why this question has to be asked is that software architecture can be brought in connection with big design upfront. A big design phase upfront is generally undesired in agile software development.

Before we start trying to answer the question we have to find out what developers understand by software architecture. Another point to research is if there is a different understanding in agile and non-agile projects. There are many different definitions:

- The **International Organization for Standardization (ISO)** defines software architecture in ISO/IEC/IEEE 42010 as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". [1] Further, the ISO also gives a definition of "architecting": "process of conceiving, defining, expressing, documenting, communicating, certifying proper implementation of, maintaining and improving an architecture throughout a system's life cycle" [1]. So in their definition this process it not part of software architecture itself.

- According to the **Microsoft Application Architecture Guide** software architecture also contains a process. "Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability." [16]

- **Martin Fowler** gave a quite general definition of software architecture. He noticed that there are many different definitions and noted that all have two common elements: "the highest-level breakdown of a system into its parts" and "the decisions that are hard to change" [9].

- **Grady Booch et al.** derived a quite comprehensive definition of software architecture based on work from Mary Shaw and David Garlan [12]:

  > Software architecture encompasses the significant decisions about
  >
  > - the organization of a software system,
  > - the selection of the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements,
  > - the composition of these elements into progressively larger subsystems,
  > - the architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition.
  >
  > [14]

- The **Software Engineering Institute** of the Carnegie Mellon University provides a wide-ranging list of definitions resp. understandings of software architecture, most of them submitted by people active in fields related to software development. [19]. That makes the list a good starting point to find out what developers understand by software architecture. Some definitions are described in detail, others are kept very short, but overall each is similar to at least one of the definitions given above. Additionally, some contributions already notice that there are many different definitions.

There are even more definitions, the provided list is of course not conclusive, but it already shows that there are many different understandings. Martin Fowler remarked this problem already in 2002 by saying: "'Architecture' is a term that lots of people try to define, with little agreement." [9] Even before Fowler, David Garlan reminded to avoid to "dilute the term 'architecture' by applying it to everything in sight" [11]. This mass of definitions makes communicating about software architecture more difficult and complicates the question if agile software development is compatible with software architecture. It is also a reason why this question needs further investigation. When it is not even clear how to understand the term software architecture, it is impossible to generally say whether software architecture plays a role in agile software development. Another problem is, that in some definitions software architecture seems be incompatible with the agile principles. For example, in Microsoft's definition software architecture is a big process. But, as already noted, one big design process is generally undesirable in agile software development.

P. Abrahamsson et al. and D. Falessi et al. name another possible reason for the problem architecture vs. agility. They say it results, at least partly, from a misunderstanding of what software architecture means in agile software development and that architecture in agile is as important than in classical waterfall. [2] [8] What they say is that software architecture has to be treated differently in agile projects than in non-agile projects that, for example, follow the waterfall model.

For our further research we focus on two understandings of software architecture: First, "**architecture as the implicit structure of a software system**". That is basically a slightly simplified version of the definition in ISO/IEC/IEEE 42010. This definition is discussed in section 2.1.

Second, "**software architecture as the result of a design process**", for example a blueprint in form of UML diagrams. How and when the process is fulfilled is not defined. We will see that this has an influence on how the research question is answered. This definition is subject of section 2.2.

2.1 ARCHITECTURE AS STRUCTURE OF SOFTWARE SYSTEMS

In this section we investigate the first research question with the understanding that software architecture is the structure of a software system. This does not include any documentation or similar. It is just the structure that is implicitly given by the current state of the software. First we look up what literature says about architecture in agile software development.

Grady Booch says architecture can be "accidental" [6]. He describes such an architecture as an architecture that emerges from design decisions that are made during development. In his eyes "having an accidental architecture is not necessarily a bad thing", as long as the design decisions are

communicated in some way and the important ones are made visible and remain visible to the other developers. Michael Waterman, James Noble and George Allan (later referred as Waterman et al.) use this term mostly in a negative way in their paper "How Much Up-Front? A Grounded Theory of Agile Architecture" [20]. An accidental architecture is described as not being "carefully thought through". Also Christine Miyachi regards "accidental architecture" as rather bad practice in her paper "Agile Software Architecture" [17]. She says that "a well thought out" architecture is usually better.

Scott Ambler clearly says that there is always an architecture. "Every system has an architecture. BUT [sic], it may not necessarily have architectural models describing that architecture." [3] Architectural models are here understood as some sort of documentation of the architecture. It is said, this works because communication aspects have a high importance in agile development. But the author also indicates that in rather large teams it might not be possible to get along without any documentation, because the overhead of direct communication is too high. Myachi supports the statement that every software has an architecture: "No matter what a team does, an architecture is created." [17]

Kent Beck emphasizes the importance of software architecture in projects using the Extreme Programming methodology: "architecture is just as important in XP projects as it is in any software project" [4].

In the paper "How Much Up-Front? A Grounded Theory of Agile Architecture" [20] by Michael Waterman, James Noble and George Allan a requirement architectures needs to fulfill to be more compatible with the agile principles is mentioned. In summary, they say that software architecture needs to be adaptable and in general open for changes. This is achieved by delaying decisions. By this, developers can gain more information on the requirements, which reduces the probability that design decision are wrong and may unnecessarily restrict the architecture.

In the next step we want to discuss these statements about software architecture in agile software development. At the first glance, the opinion of Grady Booch about "acceidental architecture" seems to be contradictory to the opinions of Waterman et al. and Myachi. Booch says accidental architecture is not necessarily bad, but Waterman et al. and Myachi warn of accidental architecture. But what is understood by the term accidental architecture differs. In Booch's paper an accidental architecture is an architecture, that is not the result of a design phase upfront, but an architecture that evolved during development. That does not mean that it was developed without any plan, but it was developed iteratively in small steps. Waterman et al. also seem to be a bit skeptical about Booch's opinion about "accidental architecture". Although they reference his paper and do not give an own definition, they describe an "accidental architecture" as not being "carefully thought through" [20] and say it "can potentially lead to gradual failure of the project" [20]. Also Myachi kind of ignores some parts of Booch's description of accidental architecture. She says an accidental architecture "isn't as good as a well thought out one" [17]. But in Booch's definition it is not necessarily an architecture that has not been thought out well, it is just an architecture that "emerges from the multitude of individual design decisions that occur during development" [6]. He also states that it is not a bad thing as long as these decisions are made manifest and the important ones are "made visible as soon as they are instituted" [6].

But in one point these opinions actually point in the same direction. They all say that architecture should not be ignored and that big design upfront does not fit into the agile world. So, architecture always has to be kept in mind during development because there is no big plan or similar to follow until the software is finished. Architecture is not developed on the basis of a big blueprint, but in many small steps and continuously thinking about changes of the architecture.

From the paper "How Much Up-Front? A Grounded Theory of Agile Architecture" [20] we learned that architecture needs to fulfill some requirements to be agile. This means architecture needs to be respected in agile software development. Otherwise, there is a risk of creating an architecture that does not compete well in an agile environment. For example, the architecture may not be open for changes. This possibly makes it harder to react to changes, which is an important advantage of agile software development. So the advantages of agile software development can be lessened by the flaws of the architecture.

Given this we conclude that Kent Becks sentence "architecture is just as important in XP projects as it is in any software project" [4] can be applied to agile methods in general. Architecture is just as important in agile projects as it is in any software project.

Another way to answer the research question is by just looking at the definition of software architecture that is used in this section. Then it becomes clear that the question is more a rhetorical question. It even does not make sense here to say software architecture in general does not or does fit with the agile principles. Every software system has some kind of structure and consists of parts that have dependencies between each other, so every software has an architecture. Myachi [17] and Ambler [3] also clearly state that every software has an architecture.

Another result of this section is that the demands on architecture in agile projects are different than in non-agile projects. One reason for this is that there is no big planing phase upfront that yields a blueprint of the architecture as an artifact. Instead, it is a result of continuous planning and development. So, software architecture emerges during development. For this it has to fulfill some requirements. For example, agility requires a higher flexibility in changing parts of the architecture during development without requiring to rewrite the whole system.

To sum up, with the understanding that architecture is the explicitly given structure of a system, the concept software architecture is not something that could fit or not fit with with agility. Instead, it is the way how architecture is developed and how the specific architecture looks like that can be contrary to the agile principles.

2.2 ARCHITECTURE AS RESULT OF A DESIGN PROCESS

In this section we investigate the first research question with respect to the understanding that software architecture is the result of a design process. One example for such a result are UML diagrams that serve as blueprint for the subsequent software development. But it could also be something completely different. We are interested in whether it is compatible with agile principles to design architecture arti-

facts. With this in mind we start with a literature research. After that we discuss the results and give an answer to the research question.

Scott Ambler says that there is not necessarily a documentation of the architecture: “Every system has an architecture. BUT [sic], it may not necessarily have architectural models describing that architecture.” [3] Architectural models here are some kind of documentation or description of the architecture. Having no architectural models can work, because direct communication plays an important role in agile development. [3] The Agile Manifesto expresses the importance of communication by the principle: “Individuals and interactions over processes and tools” [5]. Further Ambler argues, that having little to no documentation is particularly possible in small teams that work together in the same room, but large teams might have to create architectural models to overcome communication challenges.

The Agile Manifesto also says something about documentation: “Working software over comprehensive documentation” [5]. This does not addresses software architecture directly, but related to architecture as result of a design process this means that a comprehensive blueprint is undesirable.

Michael Waterman, James Noble, George Allan deal with the question of how much should be done upfront in their paper “How Much Up-front?: A Grounded Theory of Agile Architecture” [20]. They derived a “theory that explains how teams determine how much architecture to design upfront” [20]. This theory consists of forces and strategies. The forces are circumstances that make up the project’s context. The strategies determine how much should be done upfront. Which strategy to choose depends on the manifestation of the forces.

One of these strategies is “Big Design Up-Front”. By using this strategy a blueprint of the full architecture is designed at the beginning of the project. But it is said that this is generally undesirable in agile development, because it “will compromise their ability to be agile” [20]. It should only be used if the context of the project requires it.

In contrast to “Big Design Up-Front”, the strategy “Respond to Change” says that only as much as needed should be designed upfront. This means that there is no full blueprint of the architecture. Instead, design decisions are delayed “until sufficient information on the requirements is known” [20]. By this, the probability that theses decisions need to be changed in the future is decreased. All this increases the project’s agility.

In the strategy ”Emergent Architecture” only a minimum of architectural decisions are made upfront, e.g. the selection of the technology stack”. This way, development starts with an architecture that is kept as simple as possible. It is said that this strategy is explicitly useful when a running software is already needed at an early stage.

Alistair Cockburn presents a principle called “walking skeleton” [7] which heads into a similar direction as the strategies “Respond to Change” and “Emergent Architecture”. A walking skeleton is defined as “a tiny implementation of the system that performs a small end-to-end function” [7]. It is a skeleton, because it connects the main architectural components. The most application functionality, here the flesh, is missing. The skeleton is walking, because it is already a running software. So the architecture can be tested at an early stage in the project. The skeleton provides some kind of outlook of what the final architecture will look like. Architecture and functionality are developed in parallel. By following this strategy the upfront effort is reduced to a minimum and design decisions that do not need to be made at the beginning of the project are delayed.

The concept of architecture that is evolving during development is backed by other papers. For example, Miyachi says that software architecture has to be seen as iterative process [17] and Fraser says that there have to be “ongoing architecture evolution activities” [10] during development. Grady Booch’s definition of “accidental architecture” [6] is heading in a similar direction as well.

We can summarize that big design upfront is generally undesirable in agile software development. When the complete architecture is designed upfront, all design decisions are made at the beginning of the project. The problem by this is that it makes reacting to changes costly when parts of the architecture need to be changed. The time in which these parts of the architecture have been designed was wasted. Additionally some design decisions might prevent changes on the architecture, because they are too restrictive. [20] All this reduces the project’s agility.

So what we can say with respect to architecture as result of a design process is, that the architecture does not result from only one big design process. Instead, it is defined by rather small design decisions that are made during development. As the strategy “Emergent Architecture” says, only key architectural decision are made at the beginning. Delaying decisions is also an important principle here. By this, the probability that time is wasted by designing too much upfront is reduced. Additionally architectures adaptability is increased by not making decisions that might prevent changes and keeping the architecture open for modifications.

Another result of the literature research is that architecture as the result of a design process, is not necessarily a description or plan of an architecture in form of a written documentation. Instead, the resulting architecture of the initial design phase can be represented by code, for example in form of a “walking skeleton” [7]. This realizes the principle of the Agile Manifesto “Working software over comprehensive documentation” [5]. The code itself represents architectural decisions.

How much documentation of the architecture in form of text or design documents is needed depends on the teams size and structure. While small teams need little to no documentation, larger teams, where developers may be distributed over several places, need documentation to overcome communication challenges. [3]

3. ARCHITECTURAL PROPERTIES

This section focuses on properties an architecture in agile projects needs to have. These properties are needed because the architecture has to fulfill some requirements to be agile. By architectural properties we mean specific characteristics or attributes of the architecture. We are looking for architectural properties that have an impact on the compatibility with agile software development. So they affect how the architecture weakens or supports the advantages of agility.

It is hard to give concrete architectural properties, because the impact of the project’s context, e.g. the chosen agile method, on architecture is too high. The properties would

be to specific, so that they could be applied in only few cases. Instead, we give more general characteristics that help making the architecture to be compatible with agile software development. These characteristics may then be used in a project to derive concrete architectural properties that fit in the project's context.

First, we want to find out what these requirements are by looking up advices on architecture in agile software development in a literature research. Then we use the results to find and formulate characteristics that help to make the architecture compatible with agile software development. With this goal we start with the literature research.

We start with the previously referenced paper by Michael Waterman, James Noble and George Allan "How Much Up-Front? A Grounded Theory of Agile Architecture" [20]. It gives some interesting aspects about software architecture in an agile environment. Although the paper is more about the question how much upfront effort is needed, the presented strategies help to understand what the requirements to an architecture in agile environments are. The strategy "Respond to Change" gives advices on how to increase the architectures modifiability. In order to achieve this, the architecture's design needs to be kept simple. This means that it is only designed what is needed at the moment and not what might be needed in the future. By delaying design decisions, developers gather more information on the requirements, which lowers the probability the decisions have to be changed later. Another important aspect is, that decisions that are made too early, have a high risk to be unnecessarily constrained and thus, impede future development. [20]

Another strategy they present is called "Emergent Architecture". In this strategy the upfront effort is reduced to a minimum. Only key architectural decisions are made at the beginning, e.g. the selection of the technology stack. This helps to start with an architecture that is as simple as possible. It is said that this strategy is explicitly useful when a running software is already needed at an early stage.

Bob Martin presents the "Dependency Rule" in his blog article "The Clean Architecture" [15]. He says that source code dependencies should only point inwards. This means parts of the software only have dependencies on more general parts and never on specific parts.

The paper "Architecture-aware Programming in Agile Environments" [13] by Thorsten Keuler, Steffan Wagner and Bernhard Winkler describes the experiences of a company on how to handle architecture in agile software development. These experiences where gathered while introducing agile development methods. It is said that "architecture needs to be explicit so that it is easy to follow but hard to break" [13]. Another point mentioned is that "scalability of teams heavily depends on the scalability of the software structures that the systems are made of." [13] Scalability of the software structure here means the ability of the software structure to grow and to get extended. This is brought in connection with Conway's Law: "Organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations" [18]. By this they make the hypothesis that "overall productivity scales with the ability to decouple code units from one another" [13].

Given these advices and experiences on software architecture in agile software development, we now want to derive some characteristics of the architecture that are especially helpful in an agile environment. In the previous sections we argued that an architecture in agile software development is not developed by a big blueprint, but it emerges in rather small steps. The strategy "Respond to Change" by Waterman et al. [20] defines characteristics of architectures in agile software development that enable the architecture to emerge. This is achieved by keeping the design simple and delaying decisions. All this increases the architectures adaptability. A high adaptability is important because it allows making changes without requiring to rewrite the whole system. So, it helps to develop an emerging architecture.

The strategy "Emergent Architecture" has a similar goal, but gives advices on how to start. It says, the initial architecture should be kept as simple as possible. This creates an architecture than can easily emerge during development, because it contains as less restrictive decisions as possible. The architecture is kept open for changes. So, high adaptability is also an desired characteristic here.

Thorsten Keuler et al. say that scalability of the software system is important because it has an high impact on the scalability of the team. Organizations, e.g. companies, generally want to get bigger, i.e. they want to scale. For this, most times the teams also need to grow. So, according to Keuler et al., scalability of the architecture is an important aspect. To be scalable, an architecture needs to be adaptable and open for changes. So we do not see scalability as an extra characteristic, as it follows by adaptability. Additionally, this aspect is always important, not only in agile software development.

The "Dependency rule" helps to prevent confusing and complicate dependencies that make it hard to interchange parts. This also helps to increase the adaptability of the architecture.

Overall, the most important aspect of software architecture in agile software development is, that it needs to be able to emerge. For this it needs to be adaptable and open for changes. To reach this, we found some advices like, delaying decisions, keeping the design simple and preventing constraining dependencies. All this advices lead to an adaptable architecture that is open for changes. So, we can say that adaptability and being open for changes are the most important characteristics of software architecture in agile software development.

4. CONCLUSION

Software architecture can be understood in many different ways. In some definitions, software architecture and agile software engineering seems to work together with no major problems. But in others, it seems to be incompatible with the agile principles on the first sight. To be able to answer the research questions we formulated two definitions of software architecture. In the first, software architecture is the implicitly given structure of a software system. In the second, software architecture is the result of a design process.

Overall, we found out agile software development needs software architecture. It is not less important in agile projects than in other projects. But the more interesting question was why agile needs architecture. The difference is how architecture is understood and handled. Architecture in agile software development is developed in small iterative steps

and not by strictly following a plan or blueprint resulting from a design phase at the beginning of the project. To create an architecture that can emerge during development, it needs to full some requirements. In general, it must be adaptable and open for changes at any time. This way, it is possible to extend and change the architecture at any time. As reacting to changes fast is a big advantage of agile software development, this is an important requirement for software architecture.

Strategies like "Emerging Architecture" or concepts like "Walking Skeleton" can be used to create an initial architecture on which further development can be built on. They reduce the upfront effort and make it possible to evaluate the software at an early stage.

How much architectural activity a project needs, can not be answered in general. It highly depends on the project's context. [2] The project's context includes for example the chosen programming language, the organization and the domain. This leads to the conclusion that the result presented in this paper always have to be seen in the project's context.

This is especially true for section 2.1. Some of the discussed characteristics might not be important in some projects, but more important in other projects. Additionally, it was totally left out how the named requirements that enable an emerging architecture can be implemented. This would have been too specific for this paper, because it highly depends on the project's context.

Another problem is that the architectural properties we found are not proven to be right or even complete. The actual impact to real projects has to be researched in future work. This research could be done in connection with the question how the needed requirements can be implemented in practice. Both would probably be highly dependent on aspects like the programming language or the agile method.

As the existence of various understandings of software architecture is a problem in research about it, we now want to close this paper with the appeal not to "dilute the term "architecture" by applying it to everything in sight" [11].

5. REFERENCES

[1] 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.

[2] P. Abrahamsson, M. A. Babar, and P. Kruchten. Agility and architecture: Can they coexist? *IEEE Software*, 27(2):16–22, 2010.

[3] S. W. Ambler. Agile Architecture: Strategies for Scaling Agile Development. `http://www.agilemodeling.com/essays/agileArchitecture.htm`, accessed: 19.01.2018.

[4] K. Beck. *Extreme programming explained: embrace change.* addison-wesley professional, 2000.

[5] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for agile software development. 2001.

[6] G. Booch. The accidental architecture. *IEEE Softw.*, 23(3):9–11, May 2006.

[7] A. Cockburn. *Crystal clear: a human-powered methodology for small teams.* Pearson Education, 2004.

[8] D. Falessi, G. Cantone, S. A. Sarcia, G. Calavaro, P. Subiaco, and C. D'Amore. Peaceful coexistence: Agile developer perspectives on software architecture. *IEEE software*, 27(2), 2010.

[9] M. Fowler. *Patterns of enterprise application architecture.* Addison-Wesley Longman Publishing Co., Inc., 2002.

[10] S. Fraser, E. Hadar, I. Hadar, D. Mancl, G. R. Miller, and B. Opdyke. Architecture in an agile world. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 841–844, New York, NY, USA, 2009. ACM.

[11] D. Garlan. First international workshop on architectures for software systems workshop summary. *ACM SIGSOFT Software Engineering Notes*, 20(3):84–89, 1995.

[12] D. Garlan and M. Shaw. An introduction to software architecture. *Advances in software engineering and knowledge engineering*, 1(3.4), 1993.

[13] T. Keuler, S. Wagner, and B. Winkler. Architecture-aware programming in agile environments. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 229–233. IEEE, 2012.

[14] P. Kruchten. Agility and architecture: an oxymoron? *Architecture*, 2010.

[15] R. Martin. The Clean Architecture, 2012. `https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html`, accessed: 19.01.2018.

[16] J. Meier, D. Hill, A. Homer, T. Jason, P. Bansode, L. Wall, R. Boucher Jr, and A. Bogawat. Microsoft application architecture guide. *Dostupné https://msdn.microsoft.com/enus/library/ff650706.aspx*, 2009.

[17] C. Miyachi. Agile software architecture. *SIGSOFT Softw. Eng. Notes*, 36(2):1–3, Mar. 2011.

[18] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 521–530, New York, NY, USA, 2008. ACM.

[19] Software Engineering Institute. Community Software Architecture Definitions. `https://www.sei.cmu.edu/architecture/start/glossary/community.cfm`, accessed: 19.01.2018.

[20] M. Waterman, J. Noble, and G. Allan. How much up-front?: A grounded theory of agile architecture. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 347–357, Piscataway, NJ, USA, 2015. IEEE Press.

Applications of Combinatorial Testing - A Literature Review

Muhammad
Radman Sheikh
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
radman.sheikh@rwth-aachen.de

Konrad Foegen
RWTH Aachen University
Software Construction
Ahornstr. 55
52074 Aachen, Germany
konrad.foegen@swc.rwth-aachen.de

ABSTRACT
More than 40 tools and different algorithms have been developed until now to achieve combinatorial testing. The research and literature proves through case studies that applying combinatorial testing on different types of applications is not a farfetched thought. Despite of all the research and experimentation the categorization of applications on which combinatorial testing has been applied is missing.

This paper gives a literature overview that to which applications combinatorial testing has been applied and then it categorizes them and shows what were the problems and results. Different sections of this paper take different types of real world applications as example and show how combinatorial testing is applied to them and then categorizes them to different domains.

Keywords
Combinatorial testing, Applications of combinatorial testing,

1. INTRODUCTION
Combinatorial Testing is a failure detection method for several of software systems who have hundreds even thousands of configurations. For-example the recent version of Apache web server requires 1.8 × 1055 [24]unique configurations which in modern computation world is practically impossible. So in other words we can say that combinatorial testing often known as pairwise or All-Pair Testing test a system under a discrete set of input parameters containing all possible discrete combinations of those input parameters[24].CT usually models the system under test to as a set of factors (choice points or parameters).Based on this model , it generates a specific criteria which takes values from a particular domain. This sample is basically a set of all the possible combinations which are required to test the system.

Although this technique of CT is used in many domains and applications and more than 40 tools have been developed now to achieve this approach but there is a practical difficulty in applying this approach to real world applications. Researchers and practitioners are working to fill this gap. On contrary to that literature and case studies give evidence that this approach of CT has been applied to many real world applications and as far as results were concerned, some were satisfactory and some were not.

So, in this paper First we will identify that on which applications combinatorial testing has been applied by reviewing the literature and then will categorize them. The first section of this paper shows the methodology that what methodology is adopted for the categorization. How many applications are selected for the categorization, how much literature is taken under consideration, what were the querying factors?

While doing the research and going through the literature to find out on which applications combinatorial testing has been applied the main focus was on Real world applications, those applications which are being used by different commercial companies or are being used to solves real life problems. The main focus was on to find those applications which were commercially operational in different domains and every possible effort was made to avoid those applications which were either in their experimentation phase or were a demo application. Google scholar was frequently searched to find out the literature to show the appliance of combinatorial testing technique on different applications. Not only Google scholar different reference material provided by big companies like NASA, Intel and NIST was studied to find out our desired applications.

We adopted the methodology of literature review to backup our research. A literature review is an exploration of the published literature on a particular topic. By "literature" we mean books, academic journal articles, book chapters, and other sources. You summarize and analyze your result and put forward a conclusion at the end[23].

The second section of the paper shows a table in which the categorization of the application has been done and discusses the results of applying CT on some of the categories in the table. The third section shows a detailed discussion about some of our findings in the literature and the final section shows the conclusions and future work.

2. BACKGROUND
Before any categorization, facts and applications of combinatorial testing: let us first discuss that why we do a literature review. A literature review is done for the following purposes [33].

- To see what has and has not been investigated.
- To identify data sources that other researchers have used.
- To learn how others have defined and measured key concepts.
- To contribute to the field by moving research forward.
- To provide evidence that may be used to support your own findings.

Why we needed Literature Review?
We needed literature review for the following purposes

1) We wanted to know that on how many applications combinatorial testing has applied and how many of them were documented by the researchers. What were results? What worked and what did not?
2) We wanted to identify the data sources of our and other researchers.
3) We wanted a scale of authenticity for our research.
4) We wanted authentic real life applications, so that we can do categorization.

3. RESULTS
The table1 below shows the categorization of the applications discovered while going through the literature.

Table 1: Applications Categorization

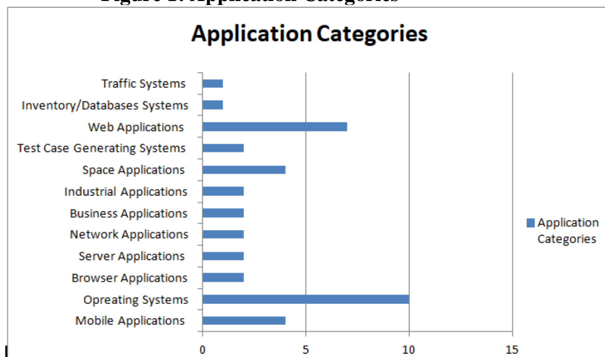
No of Applications	Name of Application	Type of Application /Domain	Type of Testing
1	Traffic Collision Avoidance System [4]	Traffic Systems	Unknown
2	Rax Engine[8]	Space Systems	Automation Testing
3	Rax Interafce[8]	Space Systems	Automation Testing
4	Rax Convergence[8]	Space Systems	Automation Testing
5	Goddard Space Flight Center[16]	Space Systems	Development and Integration Testing
6	PICT[3]	TestCase Generating Systems	Automation Testing
7	ACTs[5]	TestCase Generating Systems	Functionality and Robustness Testing
8	MyIE[18,21]	Browsers Applications	Functional Testing
9	Mozilla Web Browser[15]	Browsers Applications	Manual Testing
10	Apache Server[15]	Servers Systems	Manual Testing
11	AT&T pmx/starmail[7]	Server Systems	Unknown
12	PIV smart card[5]	Busniess Applications	Model Checking (A sort of Automation Testing
13	Data Management and Analysis System (DMAS)[24]	Business Applications	Unit Testing
14	Bellcore inventory database systems[12]	Inventory/Database Systems(ERPS)	Automation (AEGT Tool used)
15	Bellcore's Integrated Services Control Point (ISCP)[12]	Network systems	Automation(AEGT Tool used)
16	MITATE	Mobile Network (Network Systems	Unknown
17	“Simured” network simulator[6]	Network systems	Automation Testing
18	ADA compiler [13]	Compiler Systems	Unknown
19	Book[1]	Web Application	Automation Testing
20	CPM[1]	Web Application	Automation Testing
21	Webgoat - version 5.4[17]	Web Applications	Security Testing
22	Mutillidae II - version 2.6.3.1	Web Applications	Security Testing
23	Damn Vulnerable Web Application (DVWA) - version	Web Applications	Security Testing
24	Gruyere - version 201-07-15	Web Applications	Security Testing
25	– BodgeIt - version 1.3	Web Applications	Security Testing
26	I4Copter[26]	Software Product Line (Automotive Systems	Automation Testing
27	Smartify Android App[20]	Mobile Applications	Manual Testing
28	Hotwire [29]	Mobile Applications	Manual Testing(Functional Testing)
29	BookIt [30]	Mobile Applications	Manual Testing(Functional Testing)
30	Travelocity[31]	Mobile Applications	Manual Testing(Functional Testing)
31	AIX 4.1 [26]	Operating Systems	Automated Robustness Testing
32	Digital Unix 4.02[26]	Operating Systems	Automated Robustness Testing
33	FreeBSD 2.2.5	Operating Systems	Automated Robustness Testing
34	HP-UX B.10.20	Operating Systems	Automated Robustness Testing
35	Irix 6.2	Operating Systems	Automated Robustness Testing
36	Linux 2.0.1	Operating Systems	Automated Robustness Testing
37	LynxOS 2.4.0	Operating Systems	Automated Robustness Testing
38	NetBSD 1.3	Operating Systems	Automated Robustness Testing
39	QNX 4.2	Operating Systems	Automated Robustness Testing
40	SunOS 5.5	Operating Systems	Automated Robustness Testing
41	IBM® POWER7®[22]	Industrial Systems	System Testing
42	System z®[22]	Industrial Systems	System Testing
43	Loan Arranger System (LAS)[24]	Business Systems	Unit Testing

Hence, we wanted to do a proper categorization of applications, so in order to do that you need ample sample space of applications and literature backing it up. A total set of almost 43 applications from different domains were selected as you can see that in Table1 on which this technique of combinatorial testing has been applied and in order to show the results and to backup our claim a total amount of 45 data sources were consulted and a data source is mainly consisted of research papers, books and official presentations of researchers.

In the first part of the search the concentration was more on the automation part that how this technique of generating combinations can be automated and then can be applied to an application and during that search almost 15 applications were discovered. The second part of search was more concentrated on how the combinatorial testing can be extended to different types to fulfill the different requirements of different applications and during that search further 25 applications were discovered.

After doing the categorization an interesting discovery was made that the operating systems and web applications is the dominant category among the sample set of applications that were discovered. The domination of a category among 43 applications is shown in figure1

Figure 1: Application Categories



Now after doing the categorization, let's have a discussion on some of the categories that what was the reason behind creating each category ,why it was created and what was the result of applying CT on that category.

SPACE APPLICATIONS: The reason for creating this category is: majority of the applications in this category are NASA space applications and are utilized for the NASA's space projects. Now as far as result of CT on this category is concerned the study of smith et al. [8] gave us the result of pair wise testing of Remote agent planner(RAX) of software Deep Space 1 mission. The RAX is an expert system that generates plans to carry out spacecraft operations without human intervention. The study showed that testing all pairs of input values detected over 80 percent of the bugs classified as either "correctness" or "convergence" flaws in onboard planning software[16].Four components of different types of this system were tested and faults were detected and number of ratio of faults for each component was also different. Not only RAX another commercial NASA application naming Goddard Space Flight Center was tested by this approach and a total of 329 error reports from development and integration testing were analyzed[8].Different types of error from critical to cosmetics were discovered.

BUSINESS APPLICATIONS: The reason for creating this activity is due to the fact that applications involved in this categories were actually contributing towards the business cycle of the companies which were using them, plus the researched literature[24] to some extend did the same type of categorization or in other words classification. We have Schroeder's study in which code of two applications (the Loan Arranger System (LAS) and the Data Management and Analysis System (DMAS)) was injected with faults, each application was then tested with n-way test set of 2,3 and 4[9].Ten n-way test sets of each type were generated using a greedy algorithm similar to that presented by Cohn, et al. [24].The result of this activity is shown in table2

Table 2: Error Report

Fault Type	LAS	DMAS
2 Way	30	29
3 Way	4	12
4 Way	7	1
> 4 way	7	3
Not Found	34	43

WEB APPLICATION: The reasons for creating this category were that the majority of the applications in this category are web applications and are classified as the same in the literature. The security testing of 5 web applications were done and the technique of combinatorial testing was used in that security testing [17]. A framework is provided in the study for testing and detection of both reflected and stored cross-site scripting (XSS) in web applications. Combinatorial Testing is used to provide input to the penetrating tool used and its goal is to cover standard XSS exploitation attempts by checking whether certain parts of the SUT are vulnerable to potentially malicious scripts.

Two penetration tools BURP and ZAP were used and (JSO,WS1,INT,WS2,EVH,WS3,PAY,WS4,PAS,WS5,JSE) was an attacking vector which was used and the total number of test runs on each tool were 58 and the result was measured in the terms of security level set at the application, total number of successful XSS vulnerabilities detected, total coverage provided and the execution time.

OPERATING SYSTEMS: The reason for this category was that in the researched literature POSIX calls were made to different operating systems**.** Robust testing of a mission critical application using COTS approach is a difficult task as compared to traditional desktop application. In the current study under review, Ballista approach[26] performs fault injection at the API level. Injection is performed by passing combinations of acceptable and exceptional inputs as a parameter list to the system under test via an ordinary function call. The Ballista approach in the study shows robustness testing has been implemented for a set of 233 POSIX calls, including real-time extensions for C. All system calls are defined in the IEEE 1003.1b standard [25].

This Ballista Robustness test suite has been applied to ten different operating system as shown in figure 2 and a total of 233 POSIX calls were tested on each operating system. The synopsis of the experiment is given in figure 2

Figure 2: POSIX Calls

System	POSIX Functions Tested	Fns. with Catastr. Failures	Fns. with Restart Failures	Fns. with Abort Failures	Fns. with No Failures	Number of Tests	Abort Failures	Restart Failures	Normalized Failure Rate
AIX 4.1	186	0	4	77	108 (58%)	64,009	11,559	13	9.99%
Digital Unix 4.0	233	0	2	124	109 (47%)	92,658	18,316	17	15.07%
FreeBSD 2.2.5	175	0	4	98	77 (44%)	57,755	14,794	83	20.28%
HP-UX B.10.20	186	1	2	93	92 (49%)	54,996	10.717	7	13.05%
Irix 6.2	226	1	0	94	131 (58%)	91,470	15,086	0	12.62%
Linux 2.0.18	190	0	3	86	104 (55%)	64,513	11,986	9	12.54%
LynxOS 2.4.0	223	1	0	108	114 (51%)	76,462	14,612	0	11.89%
NetBSD 1.3	182	0	4	99	83 (46%)	60,627	14,904	49	16.39%
QNX 4.24	206	0	4	127	77 (37%)	74,893	22,265	655	22.69%
SunOS 5.5	233	0	2	103	129 (55%)	92,658	15,376	28	14.55%

WEB BROWSERS : In a study conducted by D. Richard Kuhn and Michael J. Reilly they explored the bug reporting database of an open source web browser naming Mozila web browser. A total 194 bugs were reviewed in the database. Each bug has been classified as fixed, verified and critical and there is a bug trigger alongside a bug which helps you replicate the same bug present in the database.

Hence a description of each bug is given with instructions on how to replicate the bug when available, the researcher reviewed a total of 194 bug reports in the browser database and replicated the interactions and applied an n-way approach of the combinatorial testing. The findings of the experiment are shown in the figure 3

Figure 3: Bug Report

Conditions (values of *n*)	Browser (194 bugs)	
	(percent)	(cumulative percent)
1	28.6	28.6
2	47.5	76.1
3	18.9	95.0
4	2.2	97.2
5	2.2	99.4
6	0.6	100.0

Apart from Mozilla another open source browser naming myIE was tested using this approach[18].MyIE is a wrapper for the Internet Explorer engine. It offers several special features, including support for up to six IEs each in its own tab, user definable search engines, multiple search engine support, favorites support, visual bookmarks, grouped bookmarks, and online version check support. Two test suites were taken , the first consisted of 243 test cases and were executed through the help of Vermont HighTest Plus1 test tool and the second test suite consisted of 50 test case and fuzz testing was done. Fuzz testing [27, 28] is a testing method used to detect security flaws in software. In fuzz testing, random messages are sent to a program's message queue.

The main goal of the study is to manipulate all of the options selected for each configuration space selected each time, it was infeasible to use the entire set of options for the MyIE browser and to investigate the changes to fault detection effectiveness and code coverage across configurations.[21]

4. DISCUSSION

In this section there will be a detailed discussion about some of the literature that what were our main findings in the literature, will discuss about the type of testing that was applied to the different applications and will discuss the results in detail and finally will discuss some categories that what was the effect of combinatorial testing on some of the categories.

The study of Sergiy Vilkomir and Brandi Amstutz[20] shows that in the modern demanding age of mobile quality: combinatorial testing plays a vital role towards the quality of mobile applications. The literature under study also shows an experiment in which three mobile applications Hotwire [29], Travelocity [30], and BookIt [31] were chosen with two sets created with the 5 random devices as discussed in the literature[20]. Testing was done remotely on real mobile devices (i.e., not on simulators) using the Perfecto Mobile service [32]. The detailed result of the testing can be found at researched literature[20] but to summarize about the result is that it gives us 12 cases for comparison of effectiveness: each EC set is compared with two random sets for each of three applications. Literature[20] shows that in seven cases (58.5%) the study's approach is more effective than the random selection of devices; in four cases (33.5%) they provided equal effectiveness; and only in one case (8%) the random set was more effective.

Testing mobile application nowadays, is becoming difficult because of the device specifics errors , with hundreds and even thousands of devices its practically impossible to do it , the best option is to select an optimal subset of options and test the systems on it . Here combinatorial testing comes in handy. It allows you select an optimal subset from the practical and theoretical points of view. In order to do that you have to follow an approach consisting of three points mentioned in the literature[20] and in our opinion it's an optimal approach because it delivers optimal results. The interesting thing to mention is while investigating the result: literature [20] found out that this approach of selecting a subset through combinatorial approach is more effective than the random selection of devices.

Now mobile applications are somewhat temporary systems: the next question is that can combinatorial testing can be applied to a real life system. A joint case study conducted by National Institute of Standards and Technology and The University of Texas at Arlington to show that can CT(Combinatorial Testing) be applied to real life systems. In order to prove that they applied combinatorial testing to a CIT tool naming ACTS. In short they tested ACTS using ACTS and the results were very interesting .They generated a total number of 1105 tests, and the execution of these tests achieved about 88% statement coverage, and detected 15 bugs in a rather mature system.

Before going into details lets us explain a little bit about ACTS. ACTS is a test case generating tools for t-way combinatorial test set. Currently the set is up to only 6. The tool is implemented in Java and provides both command line and graphical user interfaces. You can find out more about ACTS in the given literature[5].ACTS supports two test generation modes, namely, scratch and extend. The former allows a test set to be built from scratch, whereas the latter allows a test set to be built by extending an existing test set. You can go through the literature [5] to check what parameters were set? What constraints were introduced? What were the relationships between constraints? But the important thing to mention is that what their t-way strength was. They started with 2-way testing but later extended to 3-way for greater code coverage and efficiency.

While going through the literature it has been concluded that if you are combinatorial testing a real life systems these steps should be followed: 1) Construct an abstract model 2) Define parameters for it 3) Define relations4) Define constraints on parameters 5) Using the model develop the test cases and test the system .You can go through the literature [5] to find out which abstract model was constructed, what were its constraints, relations and test cases? Which components of the system were test and what were the results but the important to mention is while combinatorially testing a real world application input space modeling is a very important task and it should be done carefully, the purpose of the study was to determine that having different combinations of parameters, constraints and relations sometimes yield to such factors which does not have any type of relation with your SUT. Plus the result of this study clearly shows that if input space modeling is done properly higher code coverage and efficiency can be achieved. Plus through this approach you can functionally test the SUT by giving the valid parameter and can check the robustness of the system by giving in valid parameter

Not only on mobile and real life systems you can also apply combinatorial testing on commercial applications as well. In the literature under study the researcher presented a comprehensive analysis of the use of Combinatorial Test Design (CTD) to design system level test cases for two real world IDM's commercial applications naming IBM POWER7 and IBM System z. This study of the researcher involves the unexplored aspect of combinatorial testing of how to apply CTD on system level of two real life complex industrial applications. The literature under study explains on system level that how you can do test space definition, evaluation and changing the test plan according to the resources and requirements available. The research also proves through practical results that an unexpected improvement in the quality of the features of system under test has been seen which clearly shows the flexibility of CTD technique. While applying CTD in practice the researcher also found out that applying CTD early on design level based on requirement documents will actually improve the efficiency and issues will be caught as early as possible at development life cycle. Before making further conclusions lets have a brief introduction of our systems under test

CHARM: IBM® Power® Server Concurrent Maintenance: CHARM is basically IBM's server maintenance software which do dynamic updating and troubleshooting without disturbing the service provided by the servers to different users. You can read further more about CHARM in the literature[22]that what were features under study? How many models where tested? How many Parameters were involved? But the most important thing to mention here is that this maintenance feature is a sub component of IBM® POWER7® and there is a continuous comparison of itself to its previous version POWER 6.

EDM: System z Enhanced Driver Maintenance:
Enterprise customers demand and expect 24×7 availability from System z servers. Meeting that demand requires concurrency of maintenance including software, firmware and hardware. A major part of that equation is concurrent update of firmware fixes and delivery of new functions. The main purpose of this EDM is to concurrently update the firmware stack called the driver from one version to another version.

You can go through the literature [22] to find out what feature of the systems were tested? What parameters were involved? How many model versions were included? How many nodes were considered? But the important thing to mention here is that if you are dividing the test space of a complex enterprise system then you have to divide it from two perspectives, functional and system level. On functional level individual feature is tested and on system level a group of features are tested to check the stress of the system. While identifying parameters the researchers ask some questions to reduce the input space. Each is explained in detail in the literature[5].

In order to build a test space same procedure is followed as was in the case of real life systems but there is slight change : first indentify the parameters then assigned values to them after that identify the bad or negatives flows and consider them as well for more coverage and finally define some constraints on the input values. After the construction of the test plan, a refinement process is done and the constructed combinatorial model is evaluated. After the evaluation of the model testing is done. The same procedure is followed in literature and the evaluated test results can be found at [5].

In order to determine quality, well known test case effectiveness matrix was used which they refer to as the defects per trial ratio. You can study more about this matrix and result in the literature[5] but important thing to mention here is that two types of improvements were seen while power 7 was compared with its previous version power 6 on which CTD approach was not applied . Those improvements were Improvement in quality and Early detection of bugs

Overall the collusion can be made that if you want to apply CTD to complex industrial applications while doing system level testing, keep these factors in mind.

- Closing coverage gaps.
- Affordability.
- Importance of bad path.
- Broader view of the test space.
- Importance of root cause analysis.

Now as far mobile, commercial and real life applications are concerned Combinatorial testing can be applied to them but what about a dynamic field like networking and the answer is yes. Computer security division head of National institute of standard and technology NIST Rick Kuhn proposed a way of testing network applications and proposed a combinatorial method for discrete event simulation of a grid network. Before purposing any techniques or algorithm, he asked a simple question. Question: can combinatorial methods help us find attacks on networks?

In order to do that he purposed an experiment that find out deadlock configurations in a network using grid like network simulator with random simulation input and covering array of 2-way, 3-way and 4-way combinations. Automation Combinatorial testing methodology was recommended with Goals to reduce testing cost and improve cost-benefit ratio. The algorithm that was adopted was the covering array to model the input space. You can study about the reason of his choice and comparison of this algorithm with other ones in the literature [6].

The system under test SUT is “Simured” network simulator. It has a Kernel of approximately 5,000 lines of C++ code (not including GUI) and it main purpose was to simulate a grid like network. The main objective of the study is detect configurations that can produce deadlock, prevent connectivity loss when changing network , attacks that could lock up network ,Compare effectiveness of random vs. combinatorial inputs.

While going through the literature we found out that in order to properly identify the deadlocks the researchers asked these questions

- Are any of them dangerous?
- If so, how many?
- Which ones?

On the basis of input model discussed in literature[6] 31,457,280 configurations were detected .Now the next step was to compare the combinatorial approach with the random after performing the testing it was concluded that more deadlocks were detected using the combinatorial approach. This approach detected 14 configurations that can cause deadlock: 14/ 31,457,280 = 4.4 x 10-7[] and the most alarming thing was that combinatorial approach found one very specific configuration that most of the random test could not found.

5. CONCLUSION AND FUTURE WORK

In this paper we presented you with some of the real world applications on which combinatorial testing has been applied directly or indirectly. The process of discovery and identification was through literature review. We discussed that why this approach is important and why we adopted it. After that we describe our sample space that about 43 applications from different domains were identified and after the identification proper categorization was done using a proper schema. After doing the categorization we discussed some of the results of application on which combinatorial testing has been applied category wise.

Although literature and many case studies proves that applying combinatorial testing to real world applications is a practical thought and availability of many application is the testified prove of this idea but classification not only on the application but also on the testing types level is missing. You will find many applications on which combinatorial testing has been applied even you will find some categorization on application level as well but you will not find categorization on testing type level(functional, system) which can be an interesting research area in near future in the field of combinatorial testing.

6. REFERNCES

[1]. Xun Yuan, Myra B. Cohen, Member, IEEE, and Atif M. Memon, Member, IEEE. GUI Interaction Testing: Incorporating Event Context, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 37, NO. 4, JULY/AUGUST 2011

[2]. Sebastian Oster, Florian Markert and Philipp Ritter.Automated Incremental Pairwise Testing of Software Product Lines, Real-Time Systems Group,Computer Systems Group Technische Universitat Darmstadt, Germany

[3]. Jacek Czerwonka. Pairwise Testing in the Real World:Practical Extensions to Test-Case Scenarios, Microsoft Corporation February 2008.

[4]. D. Richard Kuhn and Vadim Okun National Institute of Standards and Technology Gaithersburg, MD 20899. Pseudo-Exhaustive Testing for Software, Software Engineering Workshop, 2006. SEW '06. 30th Annual IEEE/NASA.

[5]. Mehra N.Borazjany, Linbin Yu, Yu Lei,Raghu Kacker, Rick Kuhn. Combinatorial Testing of ACTS: A Case Study ,2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, April 2011.

[6]. Rick Kuhn Computer Security Division National Institute of Standards and Technology. Combinatorial Methods for Discrete Event Simulation of a Grid Computer Network, Computer Security Division National Institute of Standards and Technology, ModSim World, 14 Oct 09

[7]. Robert Brownlie,James Prowse and Mahdav S. Phadke.Robust Testing of AT&T PMX/StarMAIL Using OATS,Bell Labs Technical Journal,6 May 1992.

[8]. Ben Smith, Martin S Feather, Nicola Muscetolla, Challenges and Methods in Testing Remote Agent Planner. AIPS,2000 PROCEEDINGS.

[9]. James Bach,Patrick J. Schroeder. Pairwise Testing: A Best Practice That Isn't ,2006 satisfice Inc

[10]. D.M. Cohen,S.R. Dalal,J. Parelius. The combinatorial design approach to automatic test generation: 83 - 88,IEEE.Sep 1992

[11]. D.M. Cohen,S.R. Dalal,Morristown, NJ, USA,A. Kajla. The Automatic Efficient Test Generator (AETG) system,Software Reliability Engineering, 1994. Proceedings.,5th International Symposium. IEEE 6-9 Nov. 1994

[12]. S.R. Dalal Bellcore,A. Jain,N. Karunanithi.Model-based testing of a highly programmable system,Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on, IEEE 4-7 Nov. 1998

[13]. Robert Mandl,Analogic Corporation, Wakefield, MA. Orthogonal Latin squares: an application of experiment design to compiler testing,Magazine. Communications of the ACM CACM Homepage archive Volume 28 Issue 10, Oct. 1985 Pages 1054-1058 ACM New York, NY, USA.

[14]. Wenhua Wang1,Sreedevi Sampath2,Yu Lei1,Raghu Kacker. An Interaction-Based Test Sequence Generation Approach for Testing Web Applications, 2008 11th IEEE High Assurance Systems Engineering Symposium

[15]. D.R. Kuhn,M.J. Reilly Nat. Inst. of Stand. & Technol., Gaithersburg, MD, USA. An investigation of the applicability of design of experiments to software testing, Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE,5-6 Dec. 2002.

[16]. D. Richard Kuhn,Dolores R. Wallace, Member,Albert M. Gallo Jr. Software Fault Interactions and Implications for Software Testing, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING:VOL. 30, NO. 6, JUNE 2004.

[17]. Bernhard Garn,Ioannis Kapsalis, Dimitris E. Simos, Severin Winkler. On the applicability of combinatorial testing to web application security testing: a case study :Pages 16-21 San Jose, CA, USA — July 21 - 21, 2014

[18]. D. Richard Kuhn ,Raghu N. Kacker, Yu Lei National. SP 800-142. Practical Combinatorial Testing, Publication: Technical Report SP 800-142. Practical Combinatorial Testing/National Institute of Standards & Technology Gaithersburg, MD, United States ©2010

[19]. Sergiy Vilkomir,Brandi Amstutz. Using Combinatorial Approaches for Testing Mobile Applications,2014 IEEE International Conference on Software Testing, Verification, and Validation Workshops,31 March-4 April 2014

[20]. Utkarsh Goel, James Espeland, Upulee Kanewala, and Mike P. Wittie.Quality Assurance of a Mobile Network Measurement. Testbed Through Systematic Software Testing, Department of Computer Science, Montana State University, Bozeman, MT USA 59717

[21]. Myra B. Cohen,Joshua Snyder,Gregg Rothermel. Testing across configurations: implications for combinatorial testing, SIGSOFT Software Engineering Notes Volume 31: Pages 1-9 , Issue 6, November 2006

[22]. Paul Wojciak,Rachel Tzoref-Brill. System Level Combinatorial Testing in Practice -- The Concurrent Maintenance Case Study,2014 IEEE International Conference on Software Testing, Verification, and Validation, IEEE 31 March-4 April 2014.

[23]. Cemal Yilmaz, Sandro Fouché,Myra B. Cohen,Adam Porter, Gulsen Demiroz and Ugur Koc Moving Forward with Combinatorial Interaction Testing. Computer(Volume: 47, Issue: 2,):37 - 45,Feb 2014

[24]. 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, vol. 23, no. 7, pp. 437-444, 1997.

[25]. IEEE Standard for Information Technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API)—Amendment 1: Realtime Extension [C Language] (IEEE Std 1003.1b-1993), IEEE Computer Society, 1994.

[26]. Nathan P. Kropp, Philip J. Koopman, Daniel P. Siewiorek. Automated Robustness Testing of Off-the-Shelf Software Components, Digest of Papers. Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, IEEE 23-25 June 1998

[27]. J. E. Forrester and B. P. Miller. An empirical study of the robustness of Windows NT applications using random testing. In USENIX Windows System Symposium, pages 1–10, July 2000.

[28]. B. Miller, G. Cooksey, and F. Moore. An empirical study of the robustness of MacOS applications using random testing. In First International Workshop on Random Testing, pages 46–54, July 2006.

[29]. Hotwire - Android Apps on Google Play, https://play.google.com/store/apps/details?id=com.hotwire.hotels

[30]. Travelocity.com - Android Apps on Google Play, https://play.google.com/store/apps/developer?id=Travelocity.com&hl=en

[31]. BookIt.com - Android Apps on Googlelay, https://play.google.com/store/apps/details?id=com.bookit

[32]. Perfecto Mobile, http://www.perfectomobile.com

[33]. Literature review, Deakin University Australia http://www.deakin.edu.au/library/learn/literature-reviewRetreived November 12, 2017.

A Meta-Model for Maturity Model classification

Nils Wild
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
nils.wild@rwth-aachen.de

Horst Lichter
RWTH Aachen University
Software Construction
Ahornstr. 55
52074 Aachen, Germany
horst.lichter@swc.rwth-aachen.de

ABSTRACT

Maturity Models are a common approach to measure and improve a company's process capabilities. Although Maturity Models share a common structure, numerous Maturity and Assessment Models have been developed, in fact so many that it requires a lot of effort for scholars and practitioners to choose an adequate one for their business and situation. This paper proposes a Meta-Model that represents this common structure of such models and their attributes. Based on that Meta-Model these models are classified, such that choosing an appropriate model becomes easier.

For a better understanding of the common structure an overview of the history of Maturity Models is given and how they evolved. By analyzing different models criteria for grouping will be identified. Based on those findings a Meta-Model is created and finally applied to some models, to give an example on how this Meta-Model can be used to categorize Maturity Models.

Categories and Subject Descriptors

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

Keywords

ACM proceedings, Maturity Models, Assessment Model

1. INTRODUCTION

Over time multiple maturity models (MM) have been developed and evaluated. This is because maturity models and assessments allow an organization to identify weaknesses and to assess and compare its own practices against best practices. Some of those models are very domain specific others are made to be used as a general structure for a specific field but have to be highly customized.[22][31] This leads to a rather complex and time-consuming implementation of those models and Bamberger — who participated in v 1.0 of CMMI — realized that it was a big challenge for companies to analyze which elements and properties of the maturity model are important for them.[7] Anyway those Models share a common structure. This is also because of their historical development.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SWC Seminar 2017/18 RWTH Aachen University, Germany.

2. HISTORY

Maturity models represent the maturity of "something" in a hierarchical structure. "Something" because there is no limit of application fields for those models. The hierachical representation of goals that have to be archived to reach a certain maturity level has many benefits as hierarchy is an important organizational property in many natural and man-made systems. No matter if those are ecosystems, companies, societies or the Internet.[20][27][30][25] But why are hierarchical models beneficial?

2.1 Origins of hierarchical models

Humans always used hierarchical systems to organize and understand living and artificial systems, as they help to pervade complexity. The natural first step to simplify networks is to group entities into different modules. For example, we divide a company into smaller departments like marketing or production. Entities within one module are highly connected while sparsely connected to entities in other modules. This already helps to pervade complexity of systems.[14][26] Hierarchical systems add another layer on top of that. Instead of just saying entity A is related to entity B, hierarchical systems give that relation a meaning: entity A is greater than entity B. So those connections between entities are getting more specific and thus help to pervade complexity.[6]. If you're working in a small company with only a few employees and there would be no hierarchy only some departments, you might know who to ask for if you're missing some knowledge as this is a property of a specific entity (worker) or module (department) but you don't know who has to tell you what to do. This is where hierarchy simplifies complexity as you instantly know who is the one that can tell you what to do. Anyway hierarchies can also make problems more complex if they are too fine-grained for the problem. For example if you implement a ten layer hierarchy in a company with ten employees, the time that is needed to escalate a problem up to the top can be more of a hindrance. This is why hierarchical models have to fit their application.[6] Maturity Models are layered models, a specific type of hierarchical models that can be used to describe progress. In this case the progress is represented by the level that is achieved.

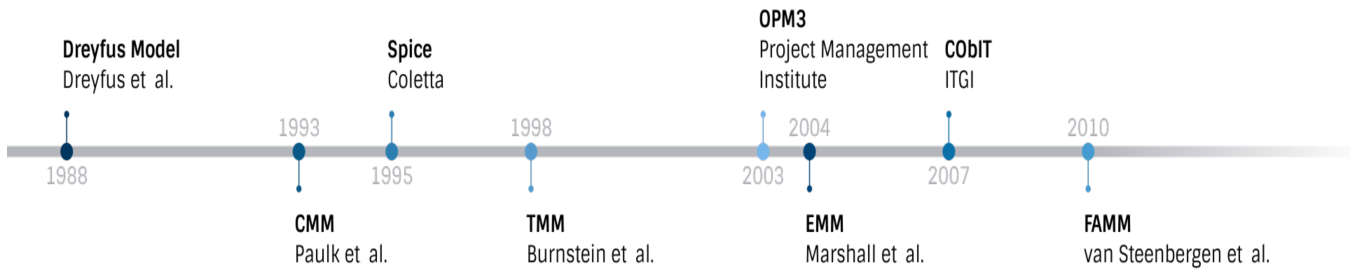


Figure 1: Timelie of Maturity Models

To chose a fitting Maturity Model for your application can be a rather complex task as there have been developed many over the past few decades.[12][24]

2.2 Evolution of Maturity Models

A lot of Maturity Models have been developed through the last years (Figure 1). A very basic and fairly famous maturity model is Maslows pyramid of needs (Figure 2.[19] It is rather simple and gives an idea of what a maturity model is. It is used in the field of social psychology to define and track the maturity of persons social desires. The only way to reach a higher maturity level is to fulfill all needs in the lower one.[19] This constraint is shared among all maturity models and it defines the lowest level of hierarchy in those models.

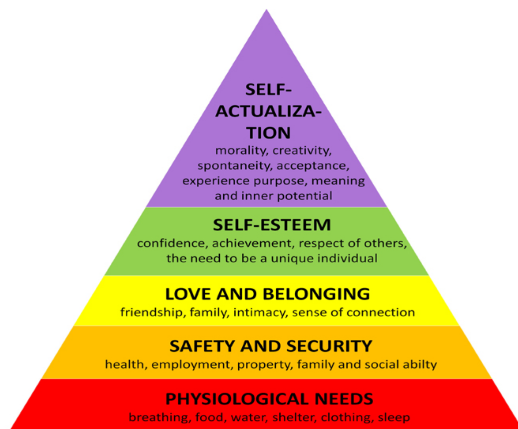


Figure 2: Marslow's pyramid of needs [19]

Business maturity models share that basic structure but add other layers of hierarchy and interdependence between Capabilities. They also focus on certain business areas or even Processes to make assessments against those models more effective. Maturity models can be divided into staged or fixed-level maturity models and continuous maturity models.[22][31] A maturity model always consists out of a maturity reference model and an assessment model that is assessed against the reference model.[23][17]

2.2.1 *Staged- and fixed-level maturity models*

Staged- and fixed-level maturity models define maturity in discrete levels. Specific goals have to be reached to archive a higher level.

One of the staged maturity models is the Capability maturity model (CMM) that has been released in August 1991 and was later refined to the Capability maturity model Integration (CMMI).[7] It defines five level of maturity. As mentioned earlier specific goals have to be reached to achieve a higher level of maturity and enable certain capabilities. Those goals have to be achieved in 22 Key Process Areas (KPA) that are organized by common features which contain key practices that are used to achieve the specific goal. In fact a KPA is equivalent to a desire in Marslows Pyramid that has to be optimized or fulfilled respectively. Anyway it's not strictly defined so it has to be adapted for your specific needs. This is why there has been different versions for different business areas.[2][3][4] It's usually accompanied by the Standard CMMI Appraisal Method for Process Improvement (SCAMPI) as an assessment framework.[5]

Another staged maturity model is the ISO/ IEC 15504 created in 1993 also known as Software Process Improvement and Capability Determination (SPICE). In contrast to CMMI it has six levels of maturity but only nine Process Groups the equivalent of KPAs in CMMI. In its early years it focused exclusively on software development processes. Later it was expanded to include all related processes in software business.[28][1][13]

2.2.2 *Continuous maturity models*

Continuous maturity models don't have discrete levels of maturity. Instead of defining maturity by specific levels, maturity is measured by Key Performance Indicators (KPI). Those are used to determine if a capability exists and to what degree.

One of those models is the Organizational Project Management maturity model (OPM3) is a standard developed by the Project Management Institute and release in December 2003. Because capabilities build on other capabilities there is still some sort of hierarchy although not defined through specific levels.[28][16]

2.2.3 *Focus area maturity models*

A fairly new type of maturity models are focus area maturity models (FAMM). Instead of fixed maturity levels that don't express the dependencies between capabilities inside a single level, they help to define a road map on how to enable

Table 1: Model Attributes based on van Looy et. al. [18]

Category	Criteria	
Structural	(1) number of assessment items	(2) number of business processes
	(3) rating scale	(4) presence of capabilities
	(5) architecture type (i.e. staged with maturity levels or continuous with capability levels)	(6) number of lifecycle levels (i.e. maturity levels or capability levels)
	(7) level calculation	(8) level representation
	(9) labelling of levels	(10) external view of levels
Confidence	(11) certification	(12) benchmarking
	(13) number of assessors	(14) functional role of respondents
	(15) business versus IT respondents	(16) data collection technique
	(17) number of assessed organisations	(18) validation methodology
	(19) architecture details (i.e. level of guidance)	(20) creation methodology
	(21) lead assessor	(22) type of business processes
costs	(23) assessment duration	(24) direct costs to access and use a MM

capabilities by considering those dependencies. By adding that additional layer of hierarchy complexity is further pervaded. This makes it easier to use especially for smaller organizations. Focus area maturity models are based on the concept of a number of focus areas instead of KPAs, that have to be developed to achieve maturity in a functional domain. In each of those areas capabilities have to be enabled in order to mature. The overall maturity is then depicted by the least mature focus area.[29]

3. CRITERIA FOR GROUPING

In the previous chapter different maturity models were presented. They obviously differ in some details but also share a common structure and terminology that is mostly interchangeable (e.g. KPA and Process Groups).[21] To classify maturity models, attributes of different MMs that share and isolate one another were identified.

Van Looy et al. identified 24 criteria that should be considered when choosing a fitting maturity model.[18] These criteria can be grouped into three categories. The first category contains criteria defining the structure and detail of the maturity model. E.g. the number of assessment items, the number of levels or the type of maturity that can be either staged or continuous. The second category consists of criteria like the number of assessed organizations or the type of business processes that are assessed by that model. Those criteria are related to the level of confidence one could have regarding the effectiveness and success probability of that model. The third and last category is related to the costs. This can either be direct costs to access and use the model or indirect costs like the assessment duration. Table 1 shows these categories identified by van Looy et al.[18]

Most of the attributes defined by the criteria in the first category can be derived from the models structure. To make that possible a single meta model, that combines the models of different maturity models, is needed. Such model is presented in the next section. This would also lead to better comparison of maturity models with different structures.

4. MATURITY REFERENCE MODEL STRUCTURE

In order to make maturity reference models comparable, the attributes and structure of different maturity reference models has to be integrated in a single Meta Model. To create such model the components and relations of those reference models are analyzed. Further more differences between those are lifted to a more abstract level to derive a single meta model that results in a single structure of components and relations that is applicable to each reference model.

4.1 CMMI

CMMI defines five levels of maturity. The first and initial level describes the state at the beginning of any project. Only base practices are applied and processes are ad hoc and chaotic. There is no stable environment in this stage. To get to a higher level certain goals in specific key process areas have to be achieved by using advanced practices.[2][3][4] Figure 3 shows a model for CMMI based on the more complex model developed by Ingalsbe et al.[15]

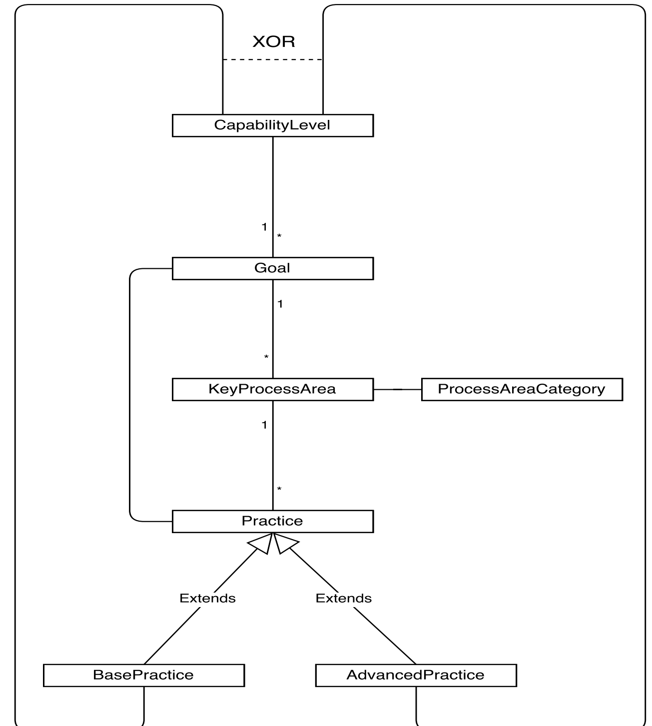


Figure 3: Model for CMMI

4.2 SPICE

SPICE defines six capability levels. Those are applied to processes through certain process attributes that can be measured. Each of these attributes consist of one or more practices that are elaborated into practice indicators. Those are used to assess each attribute on a four point rating scale.[28][1][13] Figure 4 shows a model for SPICE.

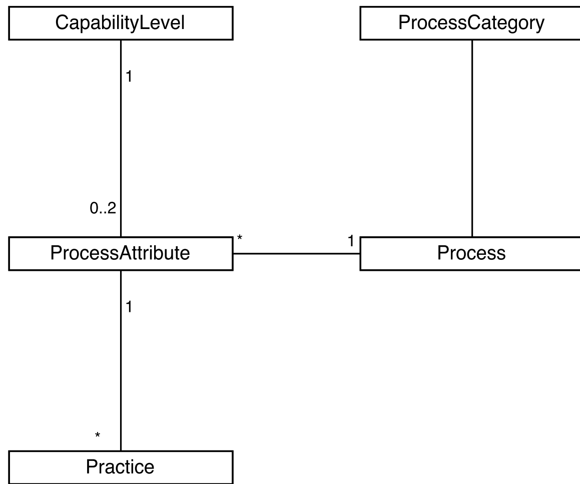


Figure 4: Model for SPICE

4.3 OPM3

OMP3 defines maturity as a continuous process instead of stages. The maturity is measured by key performance indicators as mentioned earlier. Specific practices are implemented to improve these KPIs to enable capabilities. A Capability can depend on other capabilities and refers to a specific object — in one of three domains — that should be managed. Due to that interdependence between capabilities, implicit staged levels are created.[28][16] Figure 5 shows a model for OMP3.

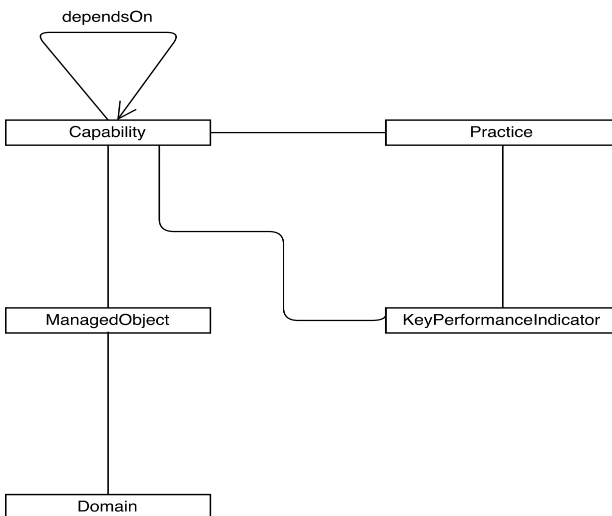


Figure 5: Model for OPM3

4.4 FAMM

Focus area maturity models consist out of a number focus areas in which capabilities can be enabled. These capabilities do have interdependencies that determine how many maturity levels exist. In order to enable these capabilities new processes and practices have to be implemented.[29] Figure 6 visualize that structure.

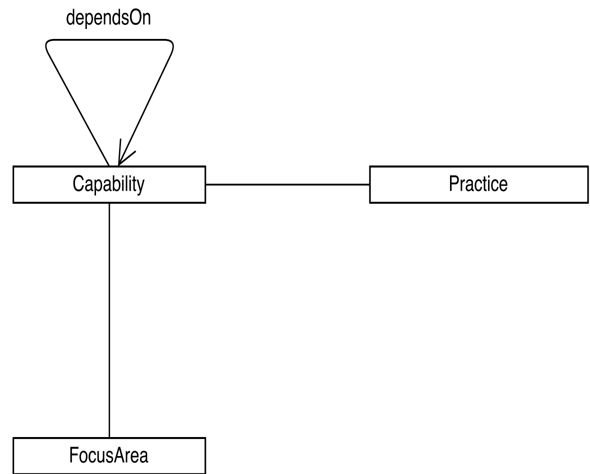


Figure 6: Model for FAMMs

5. GENERALIZED META MODEL

At first, they might seem to differ in many ways but in fact each component of one model can be mapped to a component in another model.[11][8]. First of all, the meta models structure is defined. Subsequently, a methodology is presented to derive the original attributes — like levels — from the meta models structure.

5.1 Structure

The structure of the meta model is shown in figure 7. It's similar to the model of OPM3.[28][16] It contains the common features of the different types of maturity models. Domains are used to group several managed objects together. For each of those capabilities can be enabled. A capability can depend on other capabilities. To enable a capability practices are used. Measures are used to check if a capability is enabled.

We apply this concept to the four mentioned maturity models. "ProcessAreaCategorie" (CMMI), "Domain" (OPM3) and "ProcessCategorie" (SPICE) can be mapped as "Domains". Domains aren't need for FAMMs in general but could be used to group certain focus areas together. "KeyProcessArea" (CMMI), "Process" (SPICE), "MangagedObject" (OPM3) and "FocusArea" (FAMM) can be handled mapped to "ManagedObject". In SPICE process attributes hold information about measures that can be improved using specific practices to enable capabilities. This is similar to "Goal" in CMMI. The equivalent in OPM3 is an "KeyPerformanceIndicator". There is no specific object for measures for FAMMs but there has to be some kind of method to decide whether a capability is enabled or not. All these measures can be mapped to "Measure" objects. The "Capability" in combination with its interdependencies to other capabili-

ties in OPM3 and FAMM can be directly mapped mapped. For staged and fixed-level maturity models like SPICE and CMMI, we specify that each capability of a higher level is dependent on all capabilities of a lower level. This mapping allows to map the different types of maturity models — no matter if its a staged, fixed-level, continuous or focus area model — to the meta model shown in figure 7.

If there are no circular dependencies between capabilities, the number of levels can be easily derived by analyzing those dependencies — this will always be the case for staged or fixed-level maturity models. The capability to derive attributes from the object structure is one of the benefits of object oriented models.[9][10]

In the next section we will take a look on how the original properties can be derived for the different type of models as described in section4.

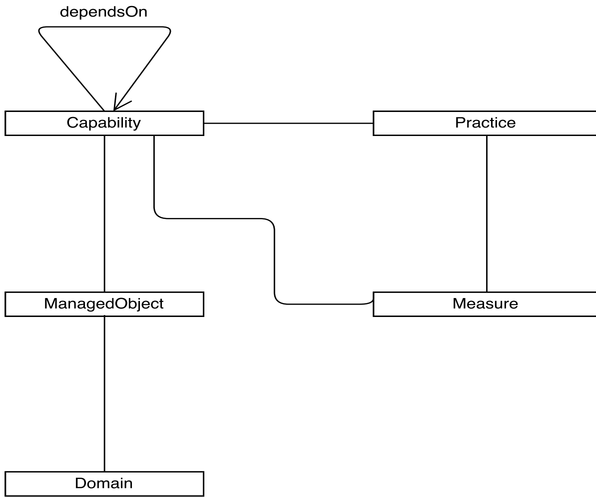


Figure 7: Meta Model

5.2 Level Mapping for staged Maturity Models

In order to show that the model is sufficient, the CMMI4.1 SPICE4.2 and FAMM4.4 models are transformed into that meta model. We don't consider OPM34.3 as this is a 1:1 mapping.

To transform the CMMI model to the meta model all goals of a KPA are mapped to capabilities that should be enabled. Their interdependencies are derived from their KPAs. For example: each goal that has been set for Organizational Innovation and Deployment (KPA of CMMI level 5) is dependent on all goals of Organizational Process Performance and Quantitative Project Management (KPAs of CMMI level 4). The original levels can then be computed.

THEOREM 1. *Given a set of dependencies expressed as tuples $D = \{(a, b)|a, b \in G\}$ - a depends of b - where G is the set of goals, each set of goals in an level can be computed by*

$$L_i = \{a|a \in G \wedge \forall(a, b) \in D \rightarrow b \in L_{i-1}\}$$

and a starting Level

$$L_0 = \{a|a, b \in G \rightarrow (a, b) \notin D\}$$

The computation can be stopped when

$$\bigcup_i L_i = G$$

The same method can be used to map SPICE. The other components can be mapped in the same manner as described in Section 5.

Focus area maturity models as presented by van Steenbergen et al can be modeled as well as described in Section 5. The dependencies between capabilities that are derived from the capabilitytable[29]. The information provided by the measures is then used to determine the current maturity of each focus area.

The computation of levels as given in Theorem 1 can be used to derive the levels but we still have to determine the maturity. This has to be model specific as it depends on the type of the underling maturity model.

In the case of staged models, maturity can be computed by finding the $max_i(\{L_i|\forall a \in L_i \rightarrow capabilityaisenabled\}$. For continuous maturity models, the measures are used to measure the overall maturity by either computing the median or weighting domains and managed objects.

6. CONCLUSIONS

In this paper commonly used maturity models have been compared, in order to create a meta model for those models. To create that model the basic components of maturity models have been evaluated. A maturity model is a description of the requirements (goals) that an organization should meet in order to achieve a desired maturity level. During the evaluation, it showed that each maturity reference model employed a slightly different structure and terminology. Furthermore, they use different type of measures to determine the projects' maturity. Based on those findings a meta model was presented that allows mapping those models to a higher level of abstraction without losing information about attributes or structure of those models. It even gives a better understanding of the hierarchy of Capabilities, as it can be computed easily. Additional useful attributes for grouping of those models have been presented. Those are used to find a fitting maturity model for a specific use case. This is important as either overfitting as well as underfitting can lead to higher costs and less acceptance in the organization. Some model attributes might be useful as measures for a models' complexity like the number of managed objects or levels. Due to the higher level of abstraction these measures can be compared across different types of maturity models.

7. FUTURE WORK

Another topic that should be considered is a meta model for maturity model assessment methods, as a maturity model is always accompanied by an assessment model. A detailed description of the assessment method is needed for the repeatability of an assessment, which contributes to the reliability of the resulting measures. This is relevant when maturity scores, especially between different maturity models, are compared with each other as well as the criteria of the second category (Table 1). Such model would allow better comparison of different models, as differences in maturity scores would be less attributable to factors other than the differences between assessed models and organizations.

8. REFERENCES

[1] Iso/iec 33001:2015 - information technology – process assessment – concepts and terminology.

[2] Cmmi for acquisition, version 1.3, 01.11.2010.

[3] Cmmi for development, version 1.3, 01.11.2010.

[4] Cmmi for services, version 1.3, 01.11.2010.

[5] Standard cmmi appraisal method for process improvement (scampi), version 1.1: Method definition document, 01.12.2001.

[6] C. Anderson and C. Brown. The functions and dysfunctions of hierarchy. *Research in Organizational Behavior*, 30:55–89, 2010.

[7] J. Bamberger. Essence of the capability maturity model. *Computer*, 30(6):112–114, 1997.

[8] J. Becker, R. Knackstedt, and J. Pöppelbuß. Developing maturity models for it management. *Business & Information Systems Engineering*, 1(3):213–222.

[9] B. T. Borsoi and J. L. R. Becerra. A method to define an object oriented software process architecture. In F. K. Hussain, editor, *19th Australian Conference on Software Engineering, 2008*, pages 650–655, Los Alamitos, Calif. [u.a.] and Los Alamitos, Calif. [u.a.], 2008. IEEE Computer Soc.

[10] B. T. Borsoi and J. L. R. Becerra. The use of object orientation to define process models. In W. Dosch, editor, *Proceedings*, pages 85–92, Los Alamitos, Calif., 2008. IEEE Computer Society.

[11] P. Brooks, O. El-Gayar, and S. Sarnikar. A framework for developing a domain specific business intelligence maturity model: Application to healthcare. *International Journal of Information Management*, 35(3):337–345, 2015.

[12] C. L. Carvajal and A. M. Moreno. The maturity of usability maturity models. In A. Mas, A. Mesquida, R. V. O'Connor, T. Rout, and A. Dorling, editors, *Software Process Improvement and Capability Determination: 17th International Conference, SPICE 2017, Palma de Mallorca, Spain, October 4–5, 2017, Proceedings*, pages 85–99. Springer International Publishing, Cham, 2017.

[13] W. Guédria, Y. Naudet, and D. Chen. Interoperability maturity models – survey and comparison –. In R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2008 Workshops: OTM Confederated International Workshops and Posters, ADI, AWeSoMe, COMBEK, EI2N, IWSSA, MONET, OnToContent + QSI, ORM, PerSys, RDDS, SEMELS, and SWWS 2008, Monterrey, Mexico, November 9-14, 2008. Proceedings*, pages 273–282. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[14] B. Henriksen and C. C. Røstad. Paths for modularization. In B. Grabot, editor, *Advances in production management systems*, volume 440 of *IFIP Advances in Information and Communication Technology*, pages 272–279. Springer, Heidelberg, 2014.

[15] J. Ingalsbe, D. Shoemaker, and V. Jovanovic. A metamodel for the capability maturity model for software. *AMCIS 2001 Proceedings*, 2001.

[16] M. Khoshgoftar and O. Osman. Comparison of maturity models. In I. Staff, editor, *2009 2nd IEEE International Conference on Computer Science and Information Technology*, pages 297–301, [Place of publication not identified], 2009. I E E E.

[17] M. Lepasaar and T. Makinen. Integrating software process assessment models using a process meta model. In *IEMC-2002*, pages 224–229, [Piscataway, NJ], 2002. IEEE.

[18] A. V. Looy, M. D. Backer, G. Poels, and M. Snoeck. Choosing the right business process maturity model. *Information and Management*, 50(7):466–488, 2013.

[19] A. H. Maslow. A theory of human motivation. *Psychological Review*, 50(4):370–396, 1943.

[20] W. Miller. The hierarchical structure of ecosystems: Connections to evolution. *Evolution: Education and Outreach*, 1(1):16–24, 2008.

[21] J. Patas, J. Pöppelbuß, and M. Goeken. Cherry picking with meta-models: A systematic approach for the organization-specific configuration of maturity models. In J. Vom Brocke, editor, *Design science at the intersection of physical and virtual design*, volume 7939 of *LNCS Sublibrary: S L 3 - Information Systems and Application, incl. Internet/Web and HCI*, pages 353–368. Springer, Berlin [etc.], 2013.

[22] J. Poeppelbuss, B. Niehaves, A. Simons, and J. Becker. Maturity models in information systems research: Literature search and analysis. *Communications of the Association for Information Systems*, 29(1), 2011.

[23] J. Poeppelbuss and M. Roeglinger. What makes a useful maturity model? a framework of general design principles for maturity models and its demonstration in business process management, 2011.

[24] D. Proença and J. Borbinha. Maturity models for information systems - a state of the art. *Procedia Computer Science*, 100:1042–1049, 2016.

[25] E. Ravasz and A.-L. Barabasi. Hierarchical organization in complex networks, 2002.

[26] G. G. ROGERS and L. BOTTACI. *Journal of Intelligent Manufacturing*, 8(2):147–156, 1997.

[27] R. Rowe, G. Creamer, S. Hershkop, and S. J. Stolfo. Automated social hierarchy detection through email network analysis, 2007.

[28] A. Van Looy, G. Poels, and M. Snoeck. Evaluating business process maturity models. *Journal of the Association for Information Systems*, 18(6), 2017.

[29] M. van Steenbergen, R. Bos, S. Brinkkemper, I. van de Weerd, and W. Bekkers. The design of focus area maturity models. In R. Winter, J. L. Zhao, and S. Aier, editors, *Global Perspectives on Design Science Research: 5th International Conference, DESRIST 2010, St. Gallen, Switzerland, June 4-5, 2010. Proceedings*, pages 317–332. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[30] A. Vazquez, R. Pastor-Satorras, and A. Vespignani. Large-scale topological and dynamical properties of internet, 2001.

[31] R. Wendler. The maturity of maturity model research: A systematic mapping study. *Information and Software Technology*, 54(12):1317–1339, 2012.

ENTERPRISE ARCHITECTURE RESEARCH

GOING BEYOND ARTIFICIAL EVALUATION

NIELS VON STEIN
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
niels.von.stein@rwth-aachen.de

SIMON HACKS
RWTH Aachen University Software Construction
Ahornstr. 55
52074 Aachen, Germany
simon.hacks@swc.rwth-aachen.de

INTRODUCTION
ENTERPRISE ARCHITECTURE (EA)

BUSINESS

PROCESS

INTEGRATION

SOFTWARE

TECHNOLOGY

Enterprise Architecture as a deliverable

- Holistic representation of an enterprise

Enterprise Architecture as a process

- Continuous alignment between business model and information systems

INTRODUCTION
ENTERPRISE ARCHITECTURE (EA)

What is the focus of EA research?

Social-Economic

Tooling

Analysis & Optimization

Security

INTRODUCTION
MOTIVATION

What is the problem with evaluation in EA research?

- Good evaluation requires real world data!
- Very confidential → Hard to get real model data-sets!

What are our options then as a researcher?

- Acquire real world EA model data-sets
- Make up your exemplary EA model data-set

Lower the barriers of sharing real world EA models

- What are the requirements on evaluative EA model data-sets?

METHODOLOGY
RESEARCH DESIGN

Systematic Literature Review (SLR)

- Observing requirements on evaluative EA model data-sets

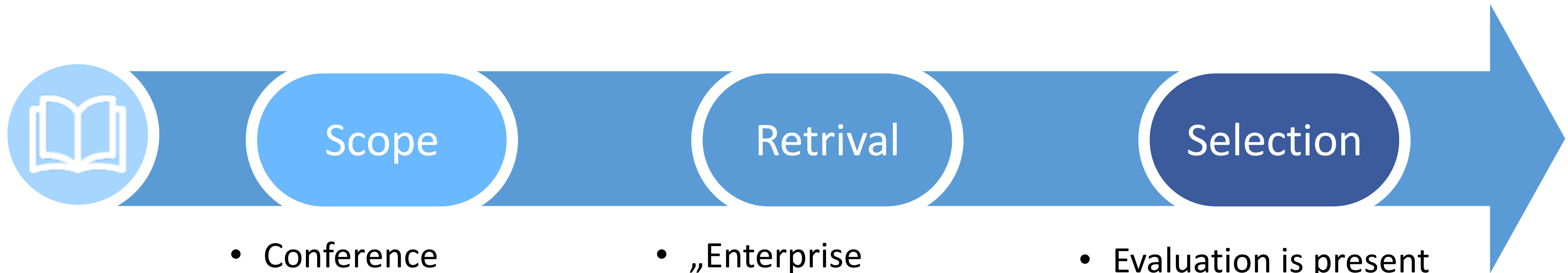
Development of taxonomy

- Classification schema for requirements on model content
- Combining observations & accepted findings of EA literature

Case Study

- Application of taxonomy
- Refinement of taxonomy

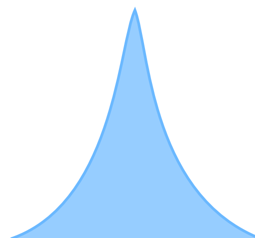
METHODOLOGY
RESEARCH DESIGN - SLR



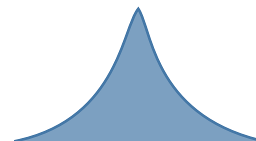
- Conference publications
- Since 2009

- „Enterprise Architecture“ …
- … and all variants

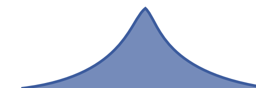
- Evaluation is present
- EA model case study
- Exemplary EA data-set



Publications from 8 year



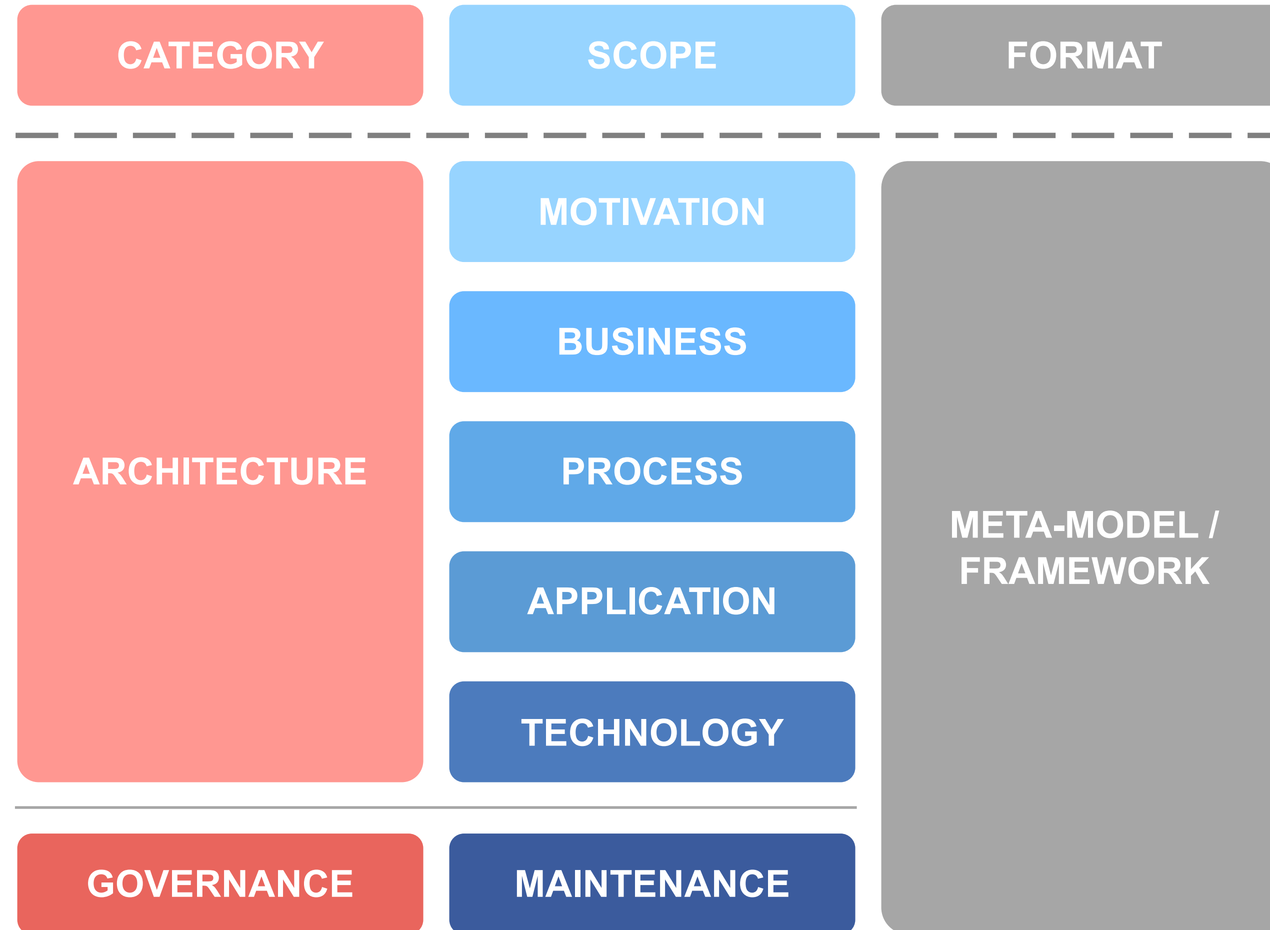
>100 Publications



11 Publications

MODEL CLASSIFICATION

SCHEMA DEVELOPMENT



MODEL CLASSIFICATION

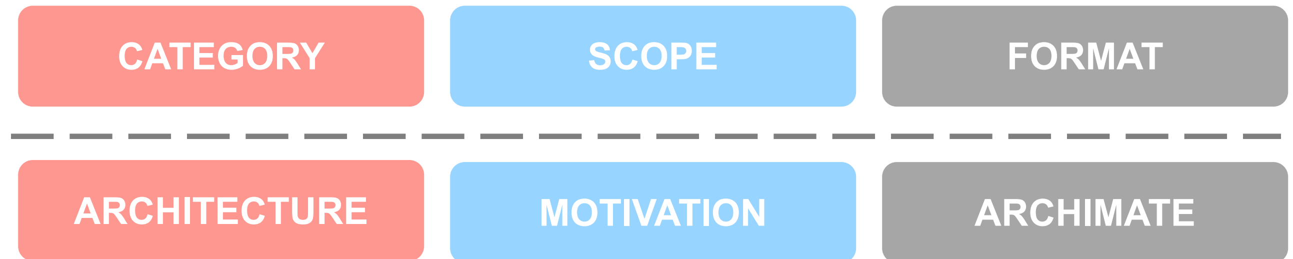
CASE STUDY

Paper A

- Risk mitigation based on measures from EA models
- Investigates visions and goals of EA model to avoid risk

Evaluation

- Exemplary data-set
- ArchiMate Framework



MODEL CLASSIFICATION

CASE STUDY

Paper B

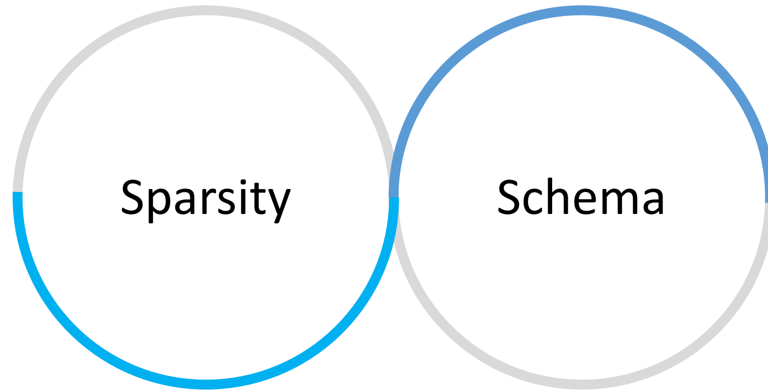
- Automated re-design of operational layer based on data mining
- Motivation, Business functions and process model

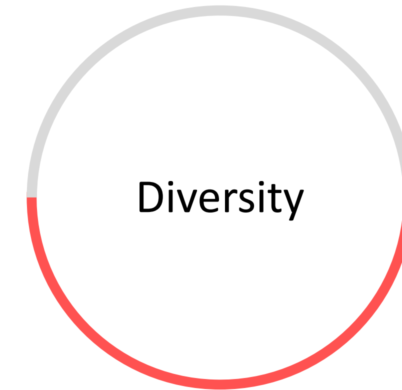
Evaluation

- Real world data-set
- ArchiMate Framework
- Business Process Model Notation (BPMN)

CATEGORY	SCOPE	FORMAT
ARCHITECTURE	MOTIVATION	ARCHIMATE
	BUSINESS	
	PROCESS	BPMN

CONCLUSION
FINDINGS & LIMITATIONS





Sparsity of model requirements

- Federated model data-sets

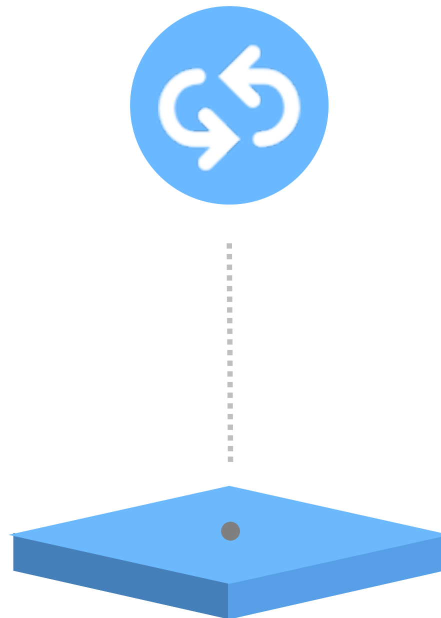
Taxonomy as a Schema

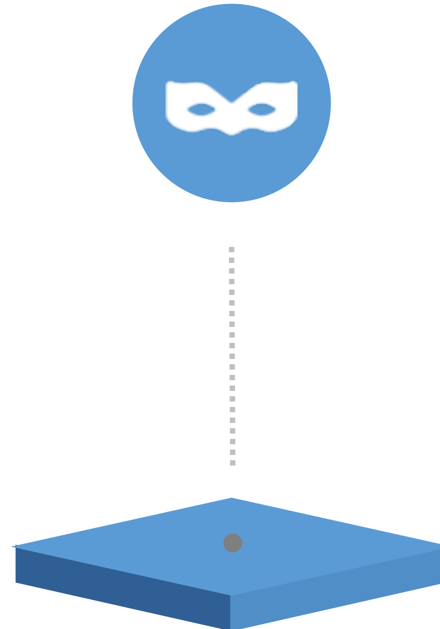
- Basis for EA model indexing

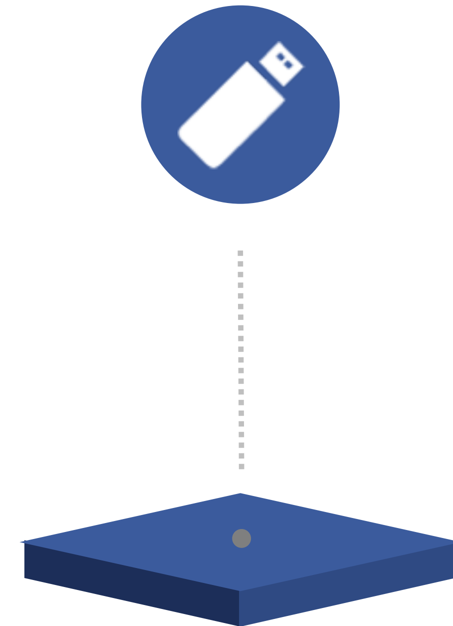
Missing meta-model diversity

- Often same meta-model

CONCLUSION
FUTURE WORK







INTRODUCTION
ENTERPRISE ARCHITECTURE (EA)

MODEL CLASSIFICATION
SCHEMA DEVELOPMENT

INTRODUCTION
ENTERPRISE ARCHITECTURE (EA)

CONCLUSION
FINDINGS & LIMITATIONS

METHODOLOGY
RESEARCH DESIGN

CONCLUSION
FUTURE WORK

Questions?

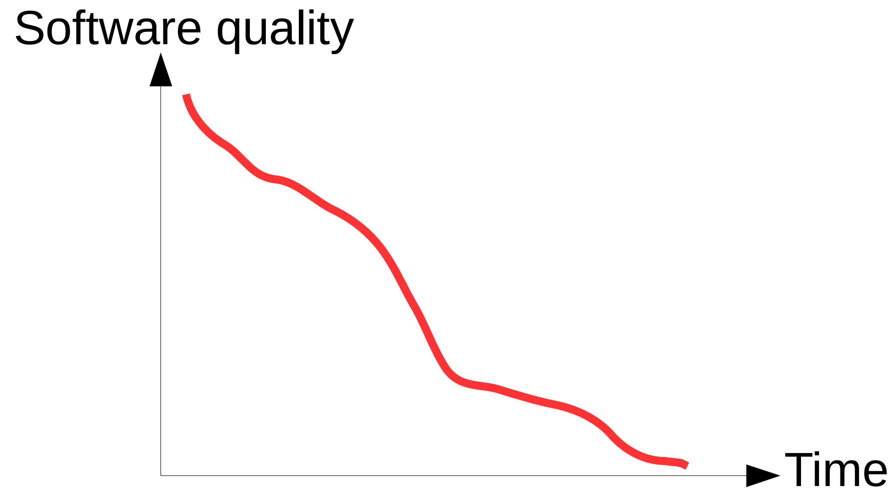
& Thanks for your attention!

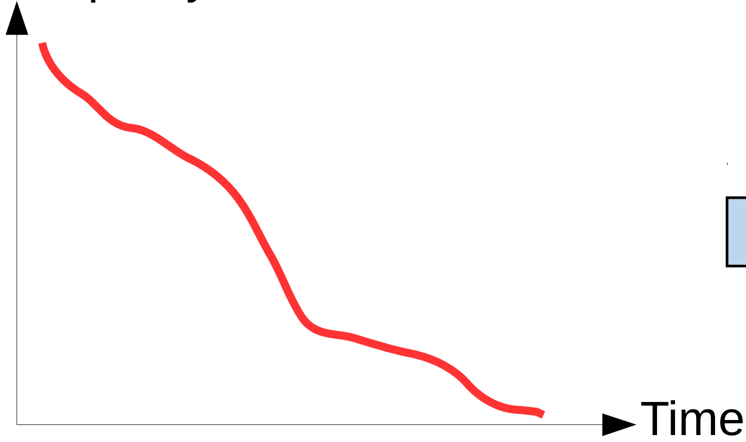
7th Feb., 2018

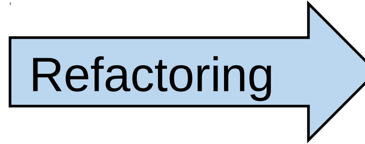
Jonas Hollm

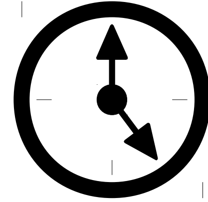
Automated Refactoring Techniques

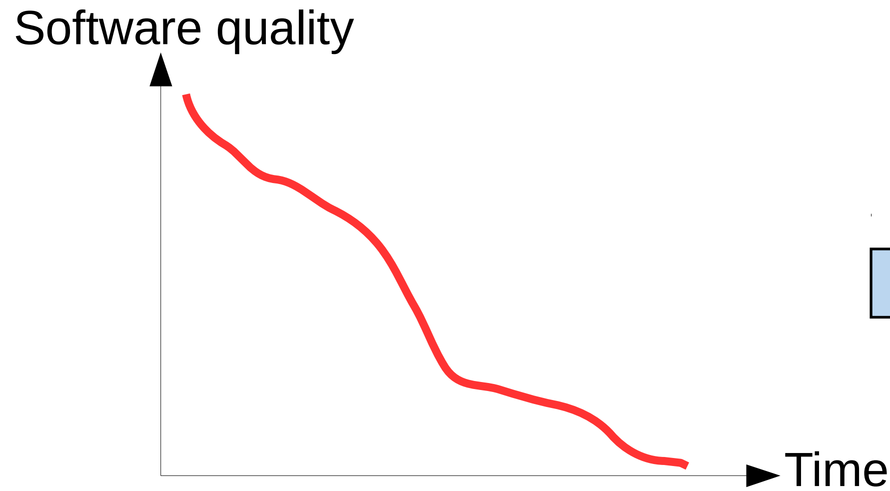
SWC Software Construction | RWTH AACHEN UNIVERSITY


Software quality
Time

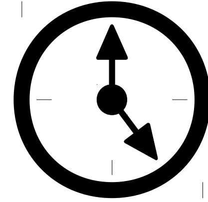

Software quality
Time


Refactoring





Refactoring



Solution: automate the refactoring process

Automated Refactoring - Goals

Software Construction
RWTH AACHEN UNIVERSITY

Automated Refactoring - Goals

- Remove code smells
  - Shotgun surgery
  - Lazy class
  - Long method

Automated Refactoring - Goals

- Remove code smells
  - Shotgun surgery
  - Lazy class
  - Long method
- Improve software metrics
  - Minimize unused methods
  - Maximize abstract classes

Automated Refactoring - Goals

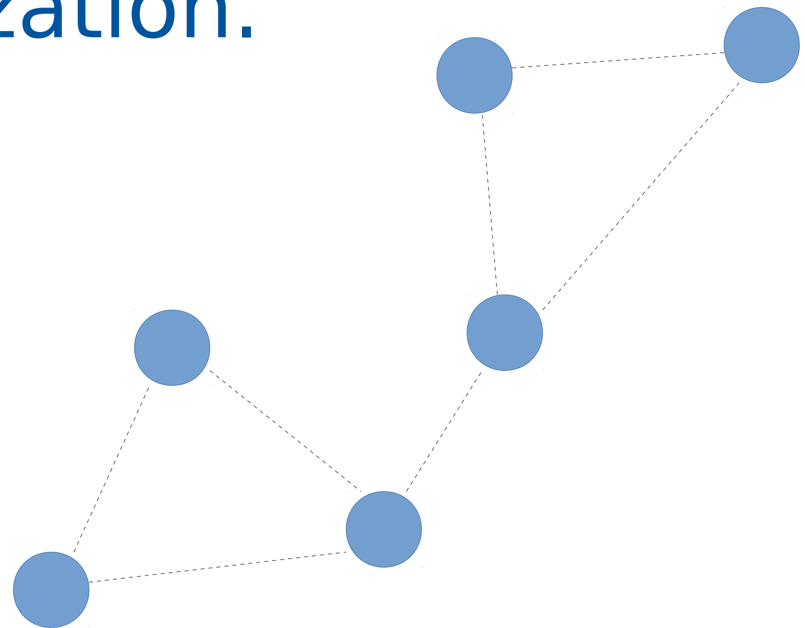
- Remove code smells
  - Shotgun surgery
  - Lazy class
  - Long method
- Improve software metrics
  - Minimize unused methods
  - Maximize abstract classes
- Introduce design patterns
  - Adapter
  - Prototype
  - Proxy

SWC Software Construction
RWTH AACHEN UNIVERSITY

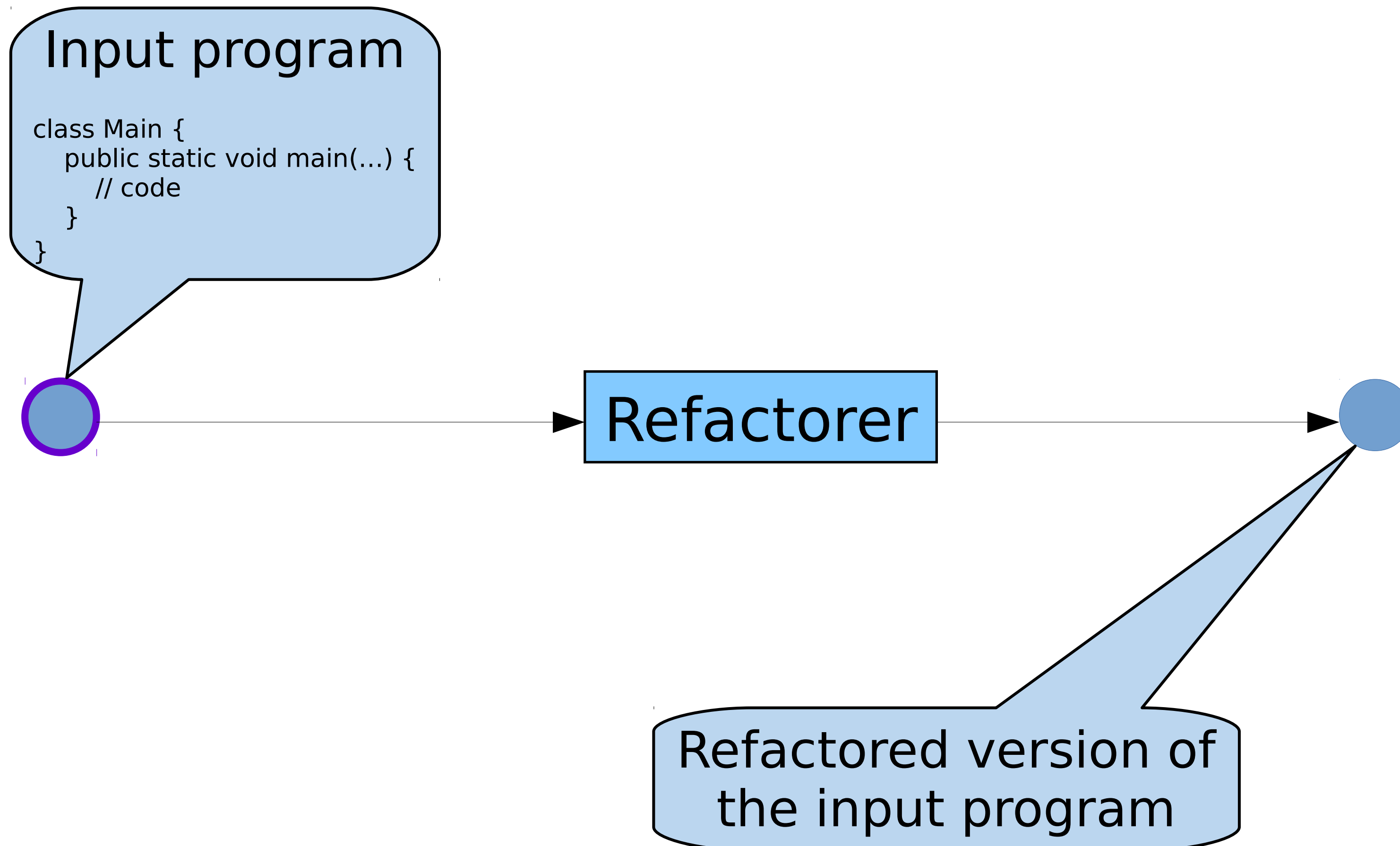
Automated Refactoring - Tools

- A-CMA
  - Technique: *Combinatorial Optimization*
- TrueRefactor
  - Technique: *Genetic Programming*
- Dearthóir
  - Technique: *Combinatorial Optimization*
- REMODEL
  - Technique: *Genetic Programming*

Combinatorial Optimization.

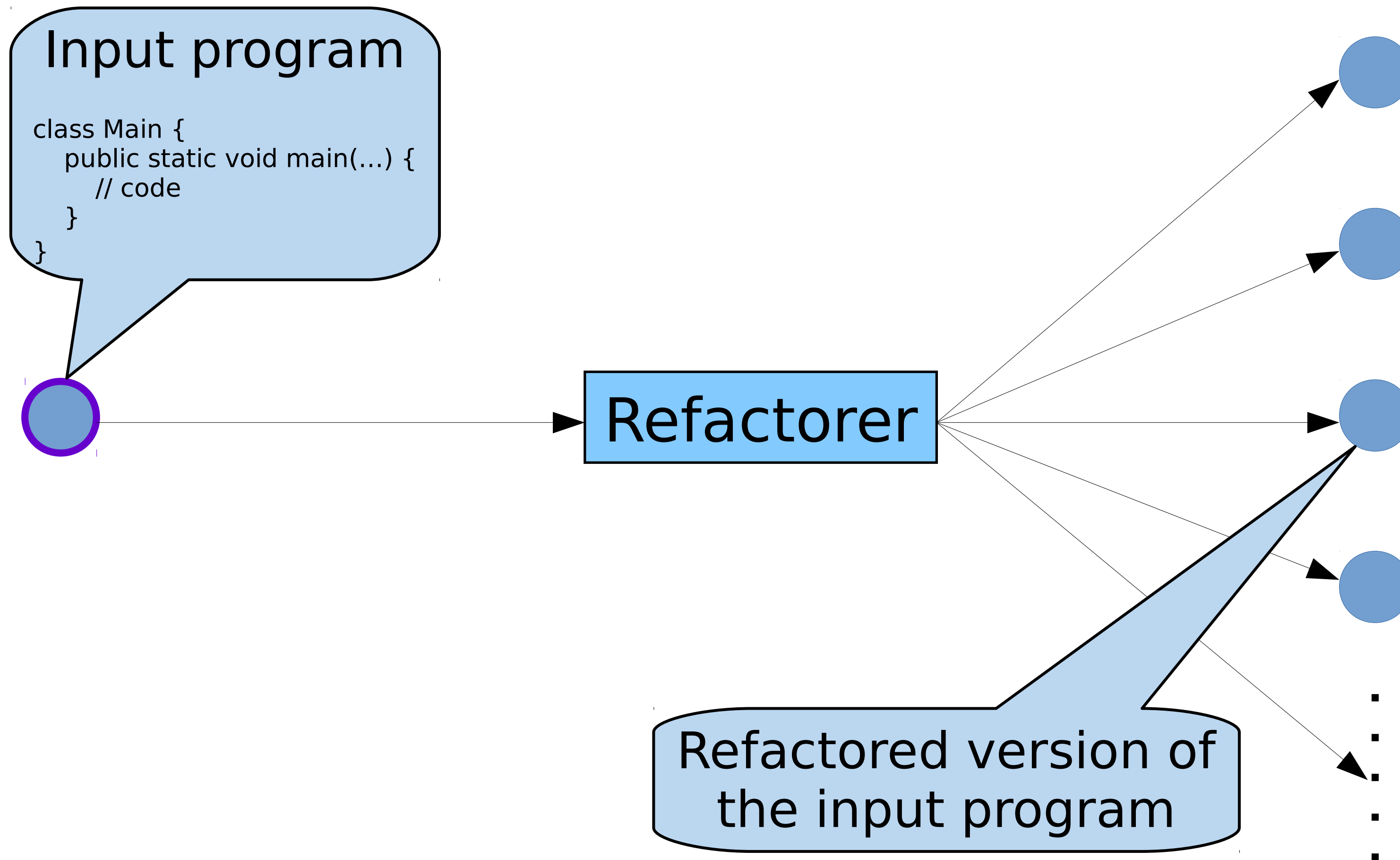


Combinatorial Optimization



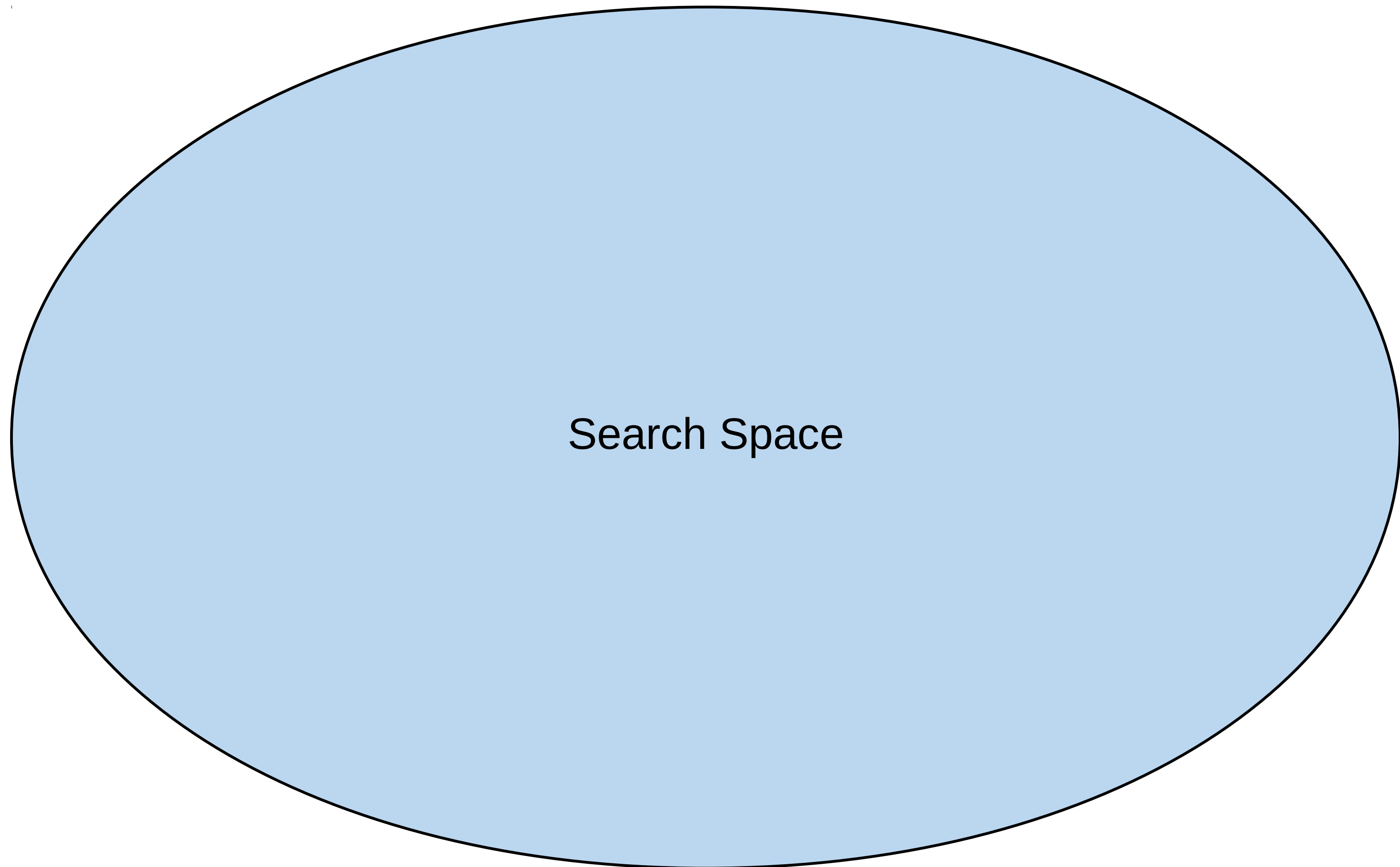
SWC Software Construction
RWTH AACHEN UNIVERSITY

Combinatorial Optimization



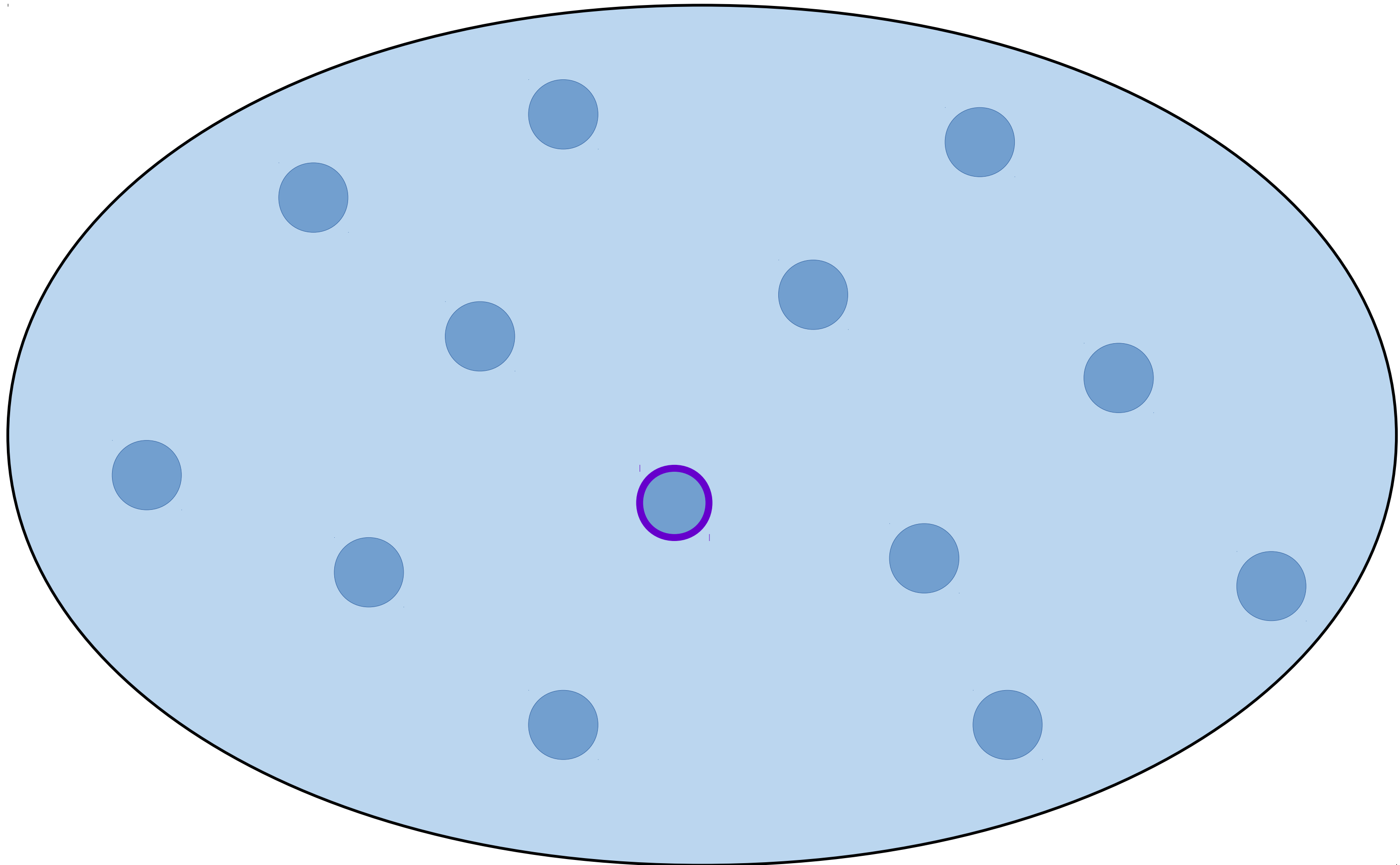
Software Construction
RWTH AACHEN UNIVERSITY

Combinatorial Optimization - Search Space

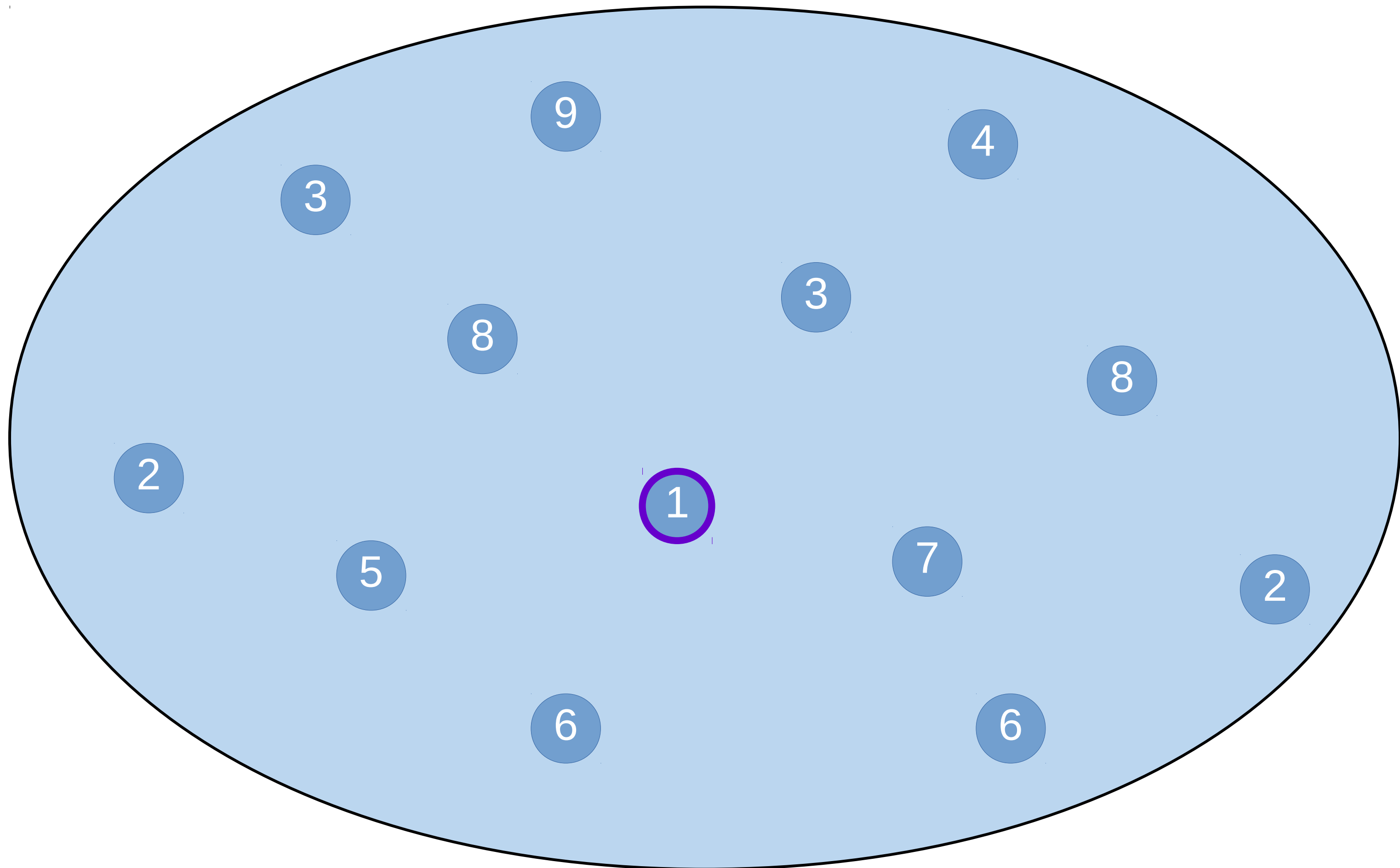


SWC Software Construction
RWTH AACHEN UNIVERSITY

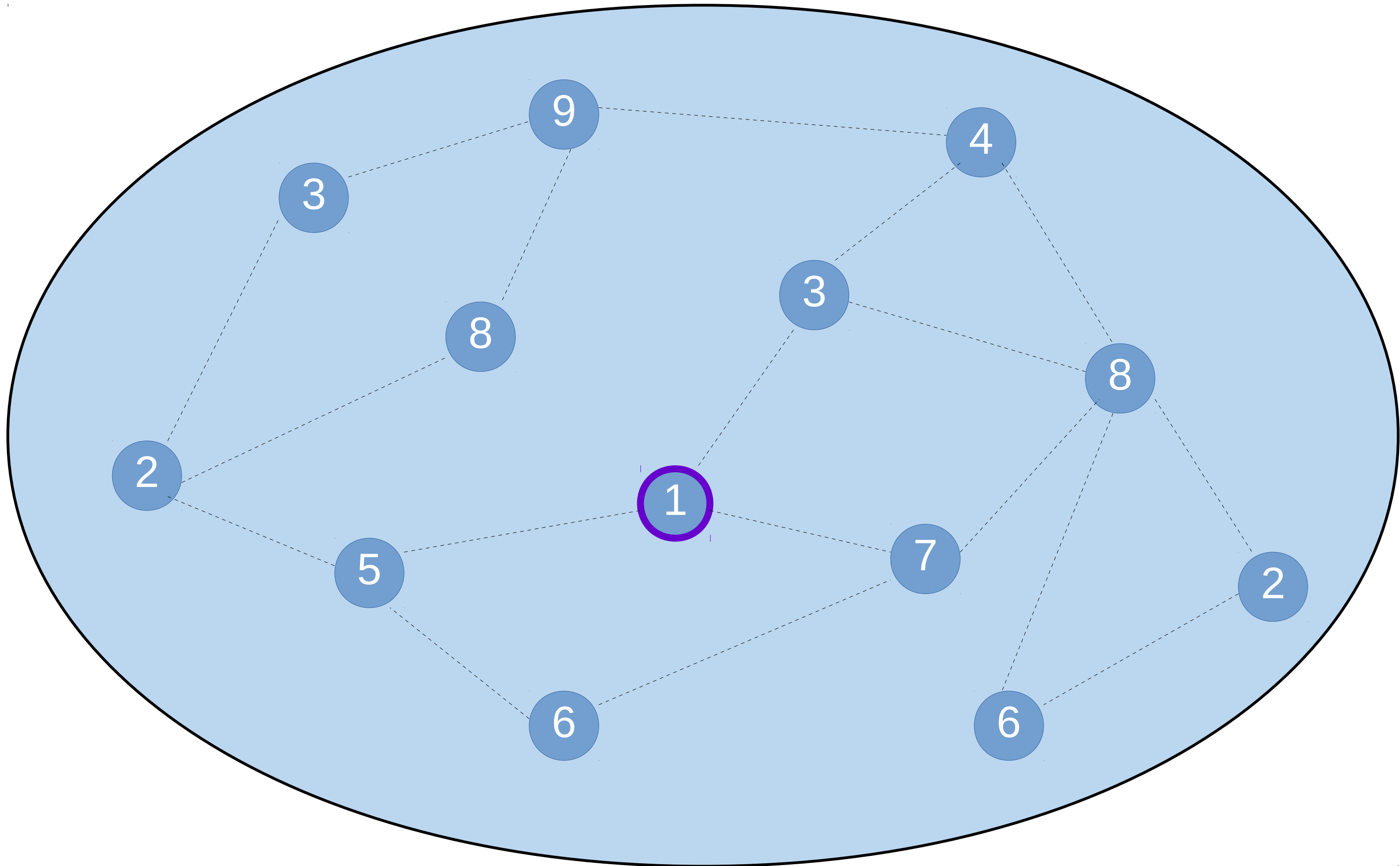
Combinatorial Optimization - Search Space



Combinatorial Optimization - Quality Values



Combinatorial Optimization - Neighborhood



Combinatorial Optimization - Refactoring Operations

- Remove unused field

```
int add(int a, int b) {
    int tmp;
    return (a + b);
}
```

```
int add(int a, int b) {
    return (a + b);
}
```


Software Construction RWTH AACHEN UNIVERSITY

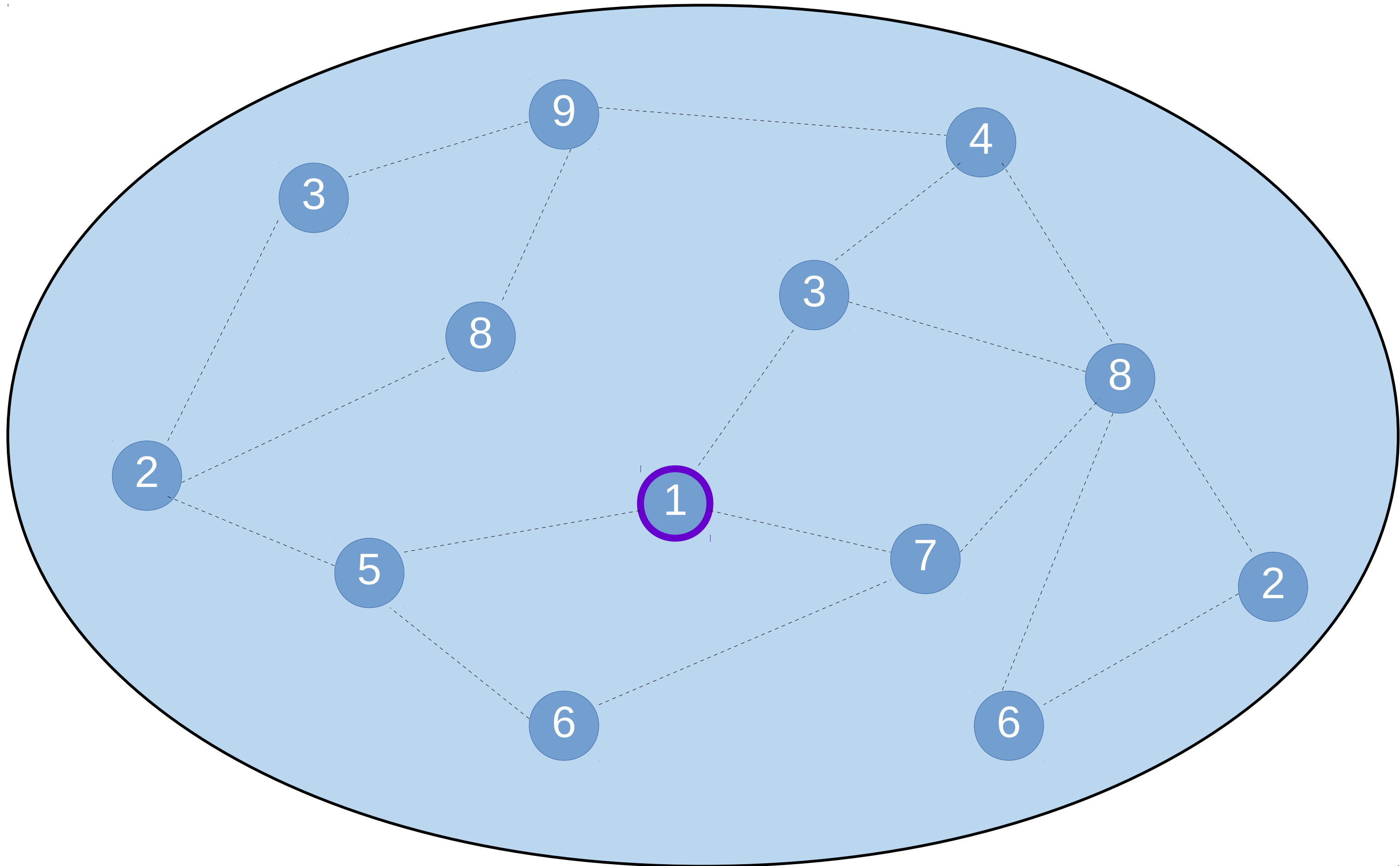
Combinatorial Optimization - Refactoring Operations

- Split long function

```
int longFunction(int param) {
    // many lines of code
}
```

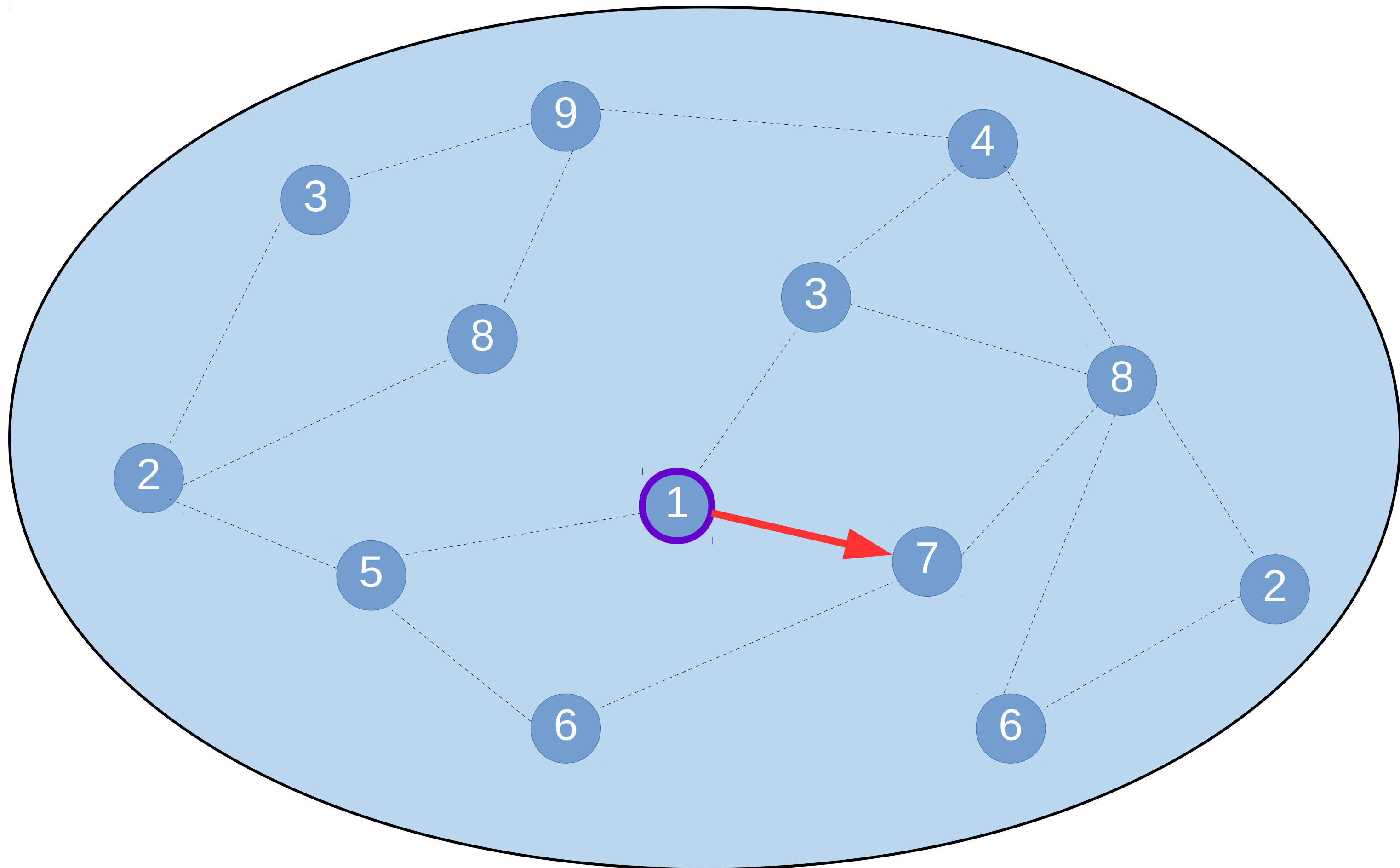
```
int shortFunction(int param) {
    // code
    helperFunction(...);
    // code
}
int helperFunction(...) {
    // code
}
```

Combinatorial Optimization - Heuristic



Software Construction
RWTH AACHEN UNIVERSITY

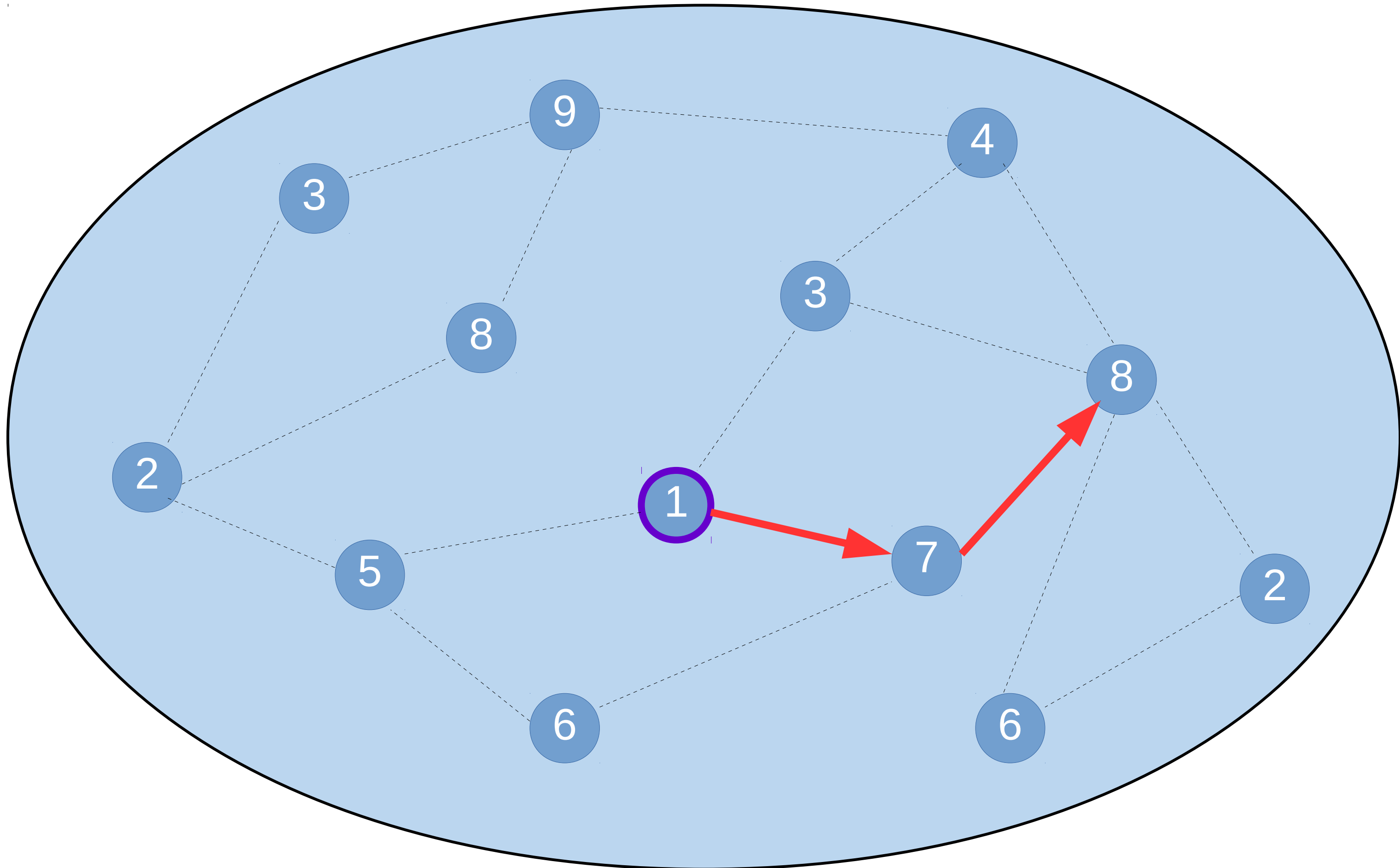
Combinatorial Optimization - Heuristic



Software Construction

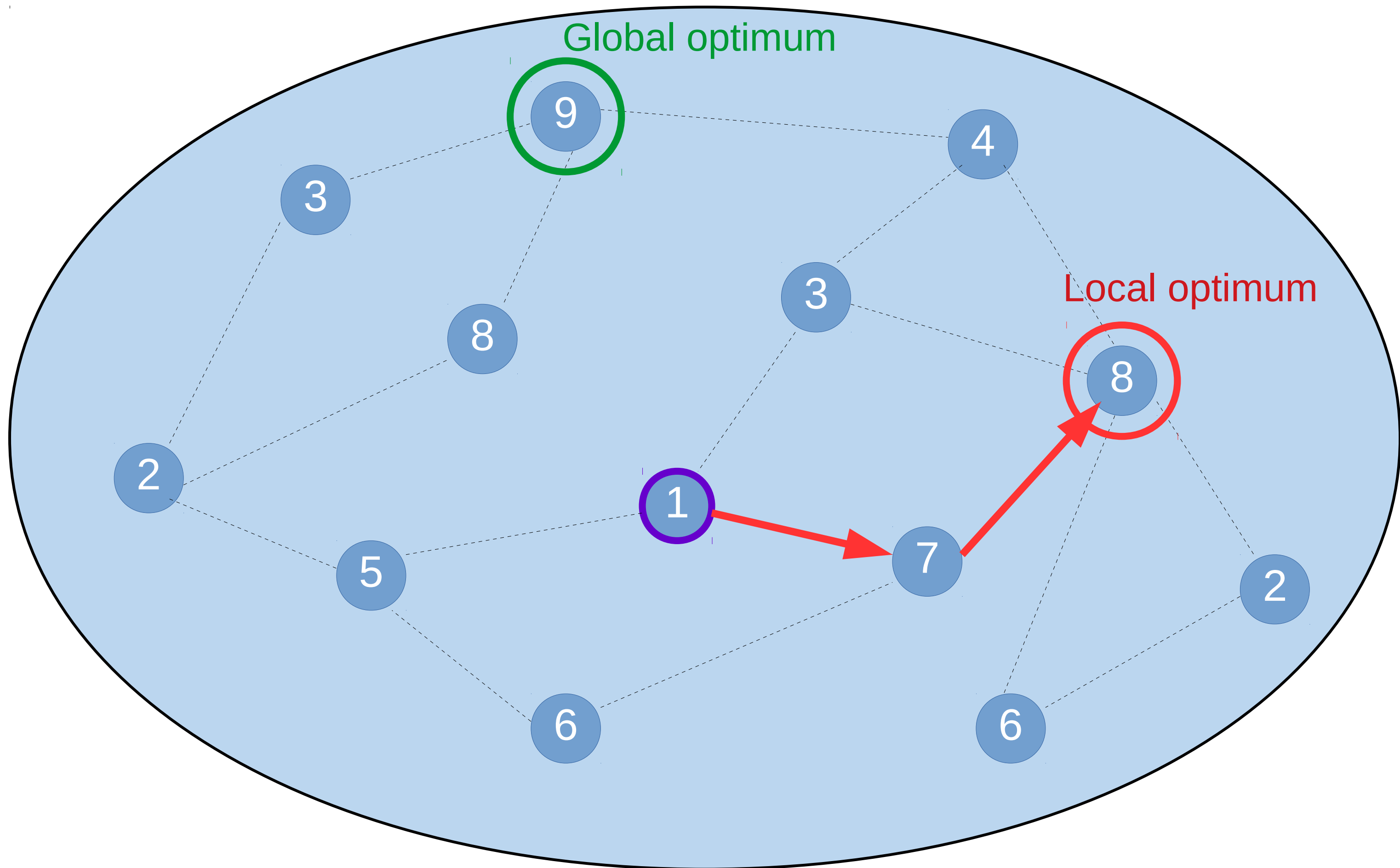
RWTH AACHEN UNIVERSITY

Combinatorial Optimization - Heuristic



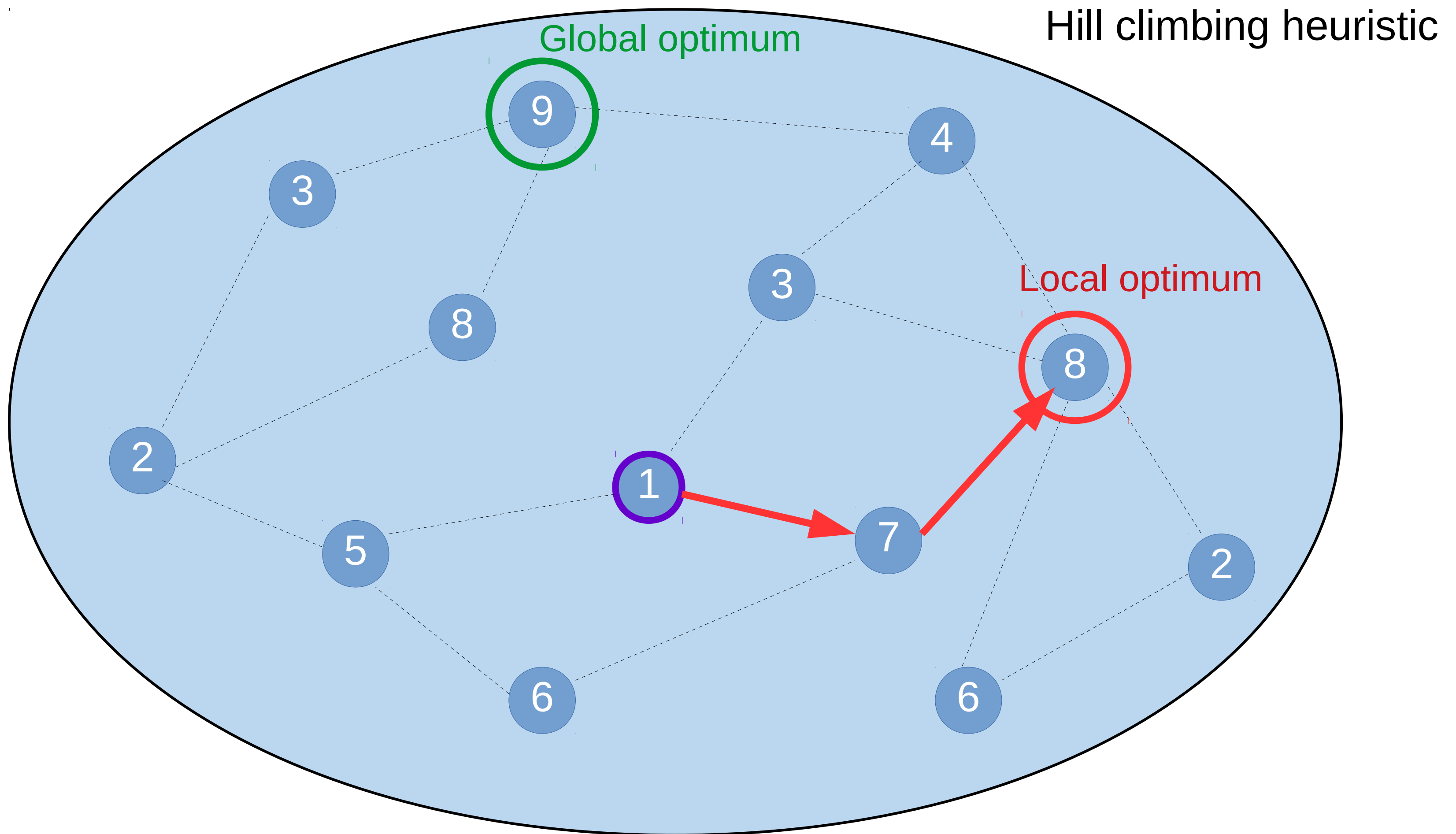
SWC Software Construction
RWTH AACHEN UNIVERSITY

Combinatorial Optimization - Heuristic



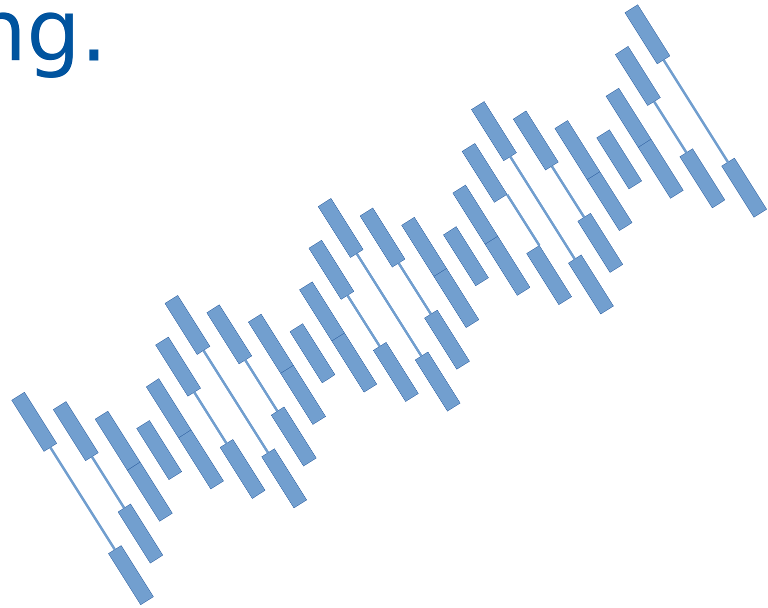
SWC Software Construction
RWTH AACHEN UNIVERSITY

Combinatorial Optimization - Heuristic

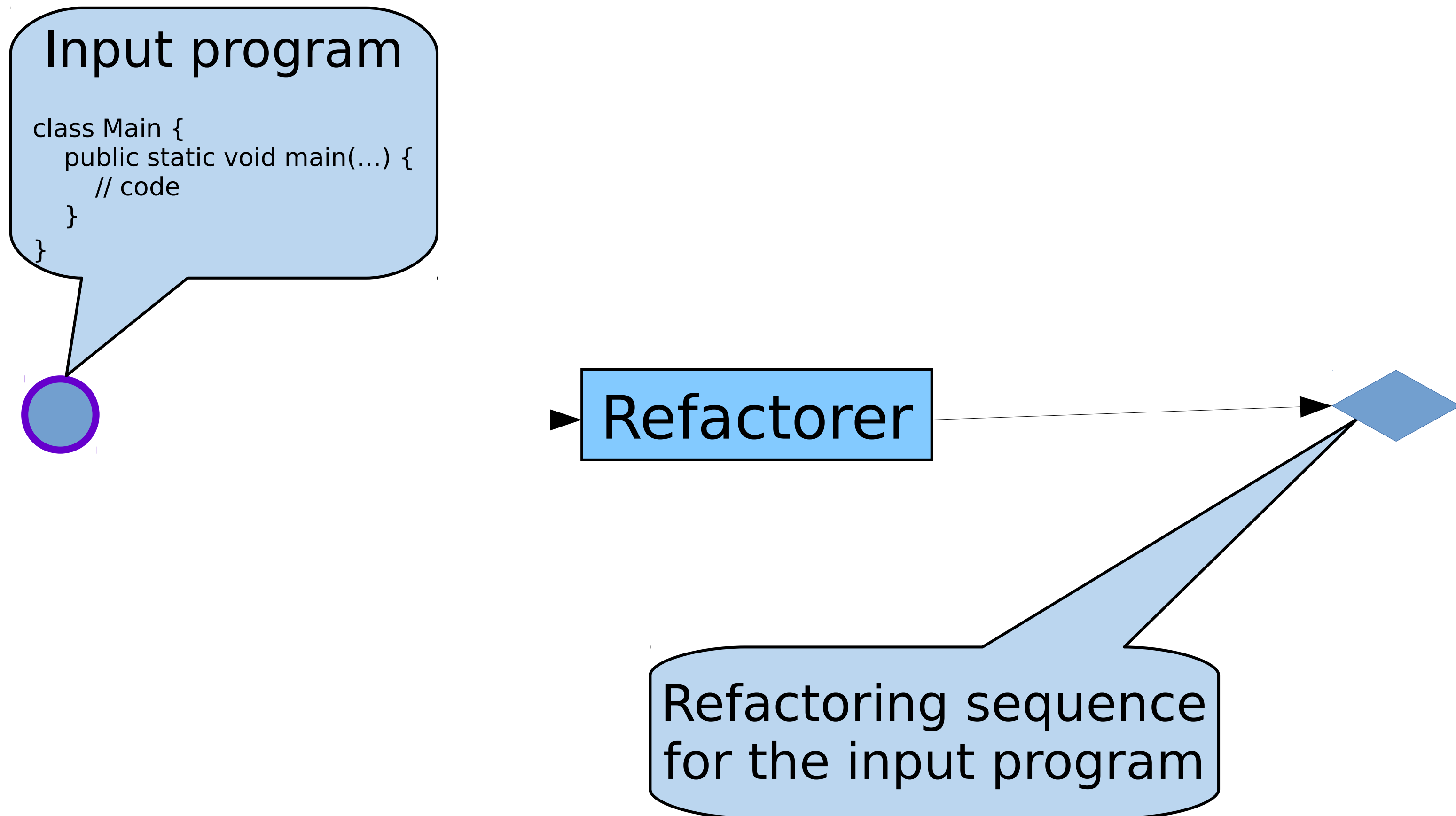


SWC Software Construction | RWTH AACHEN UNIVERSITY

Genetic Programming.

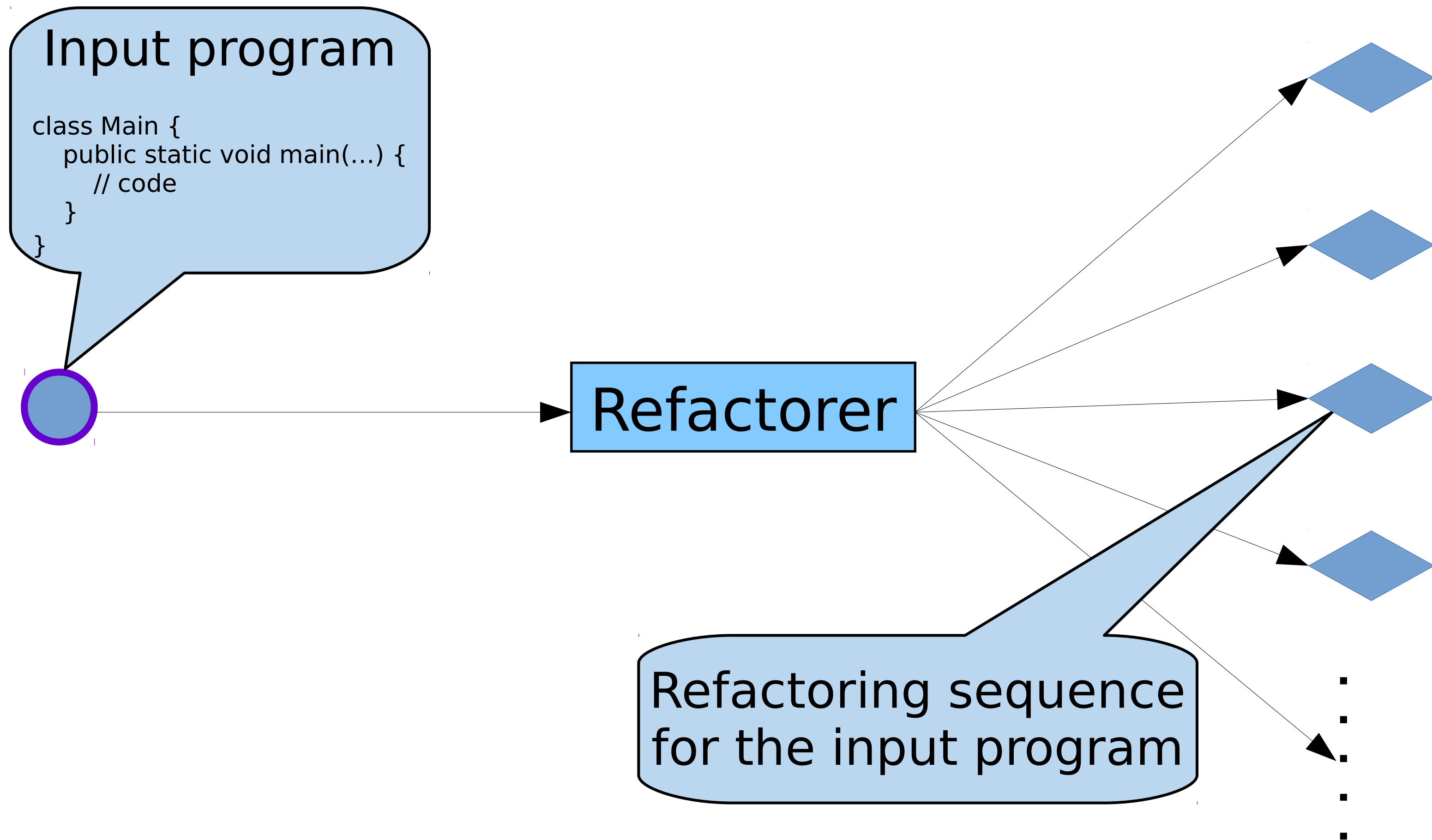


Genetic Programming

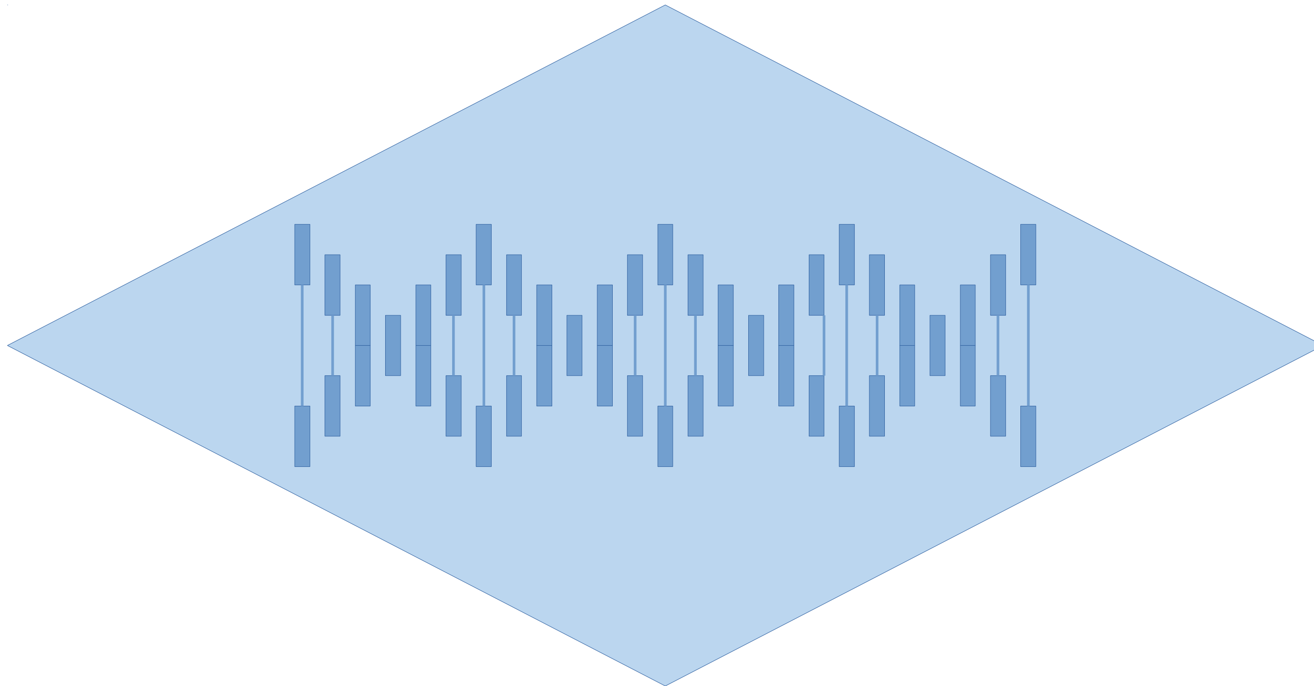


Software Construction
RWTH AACHEN UNIVERSITY

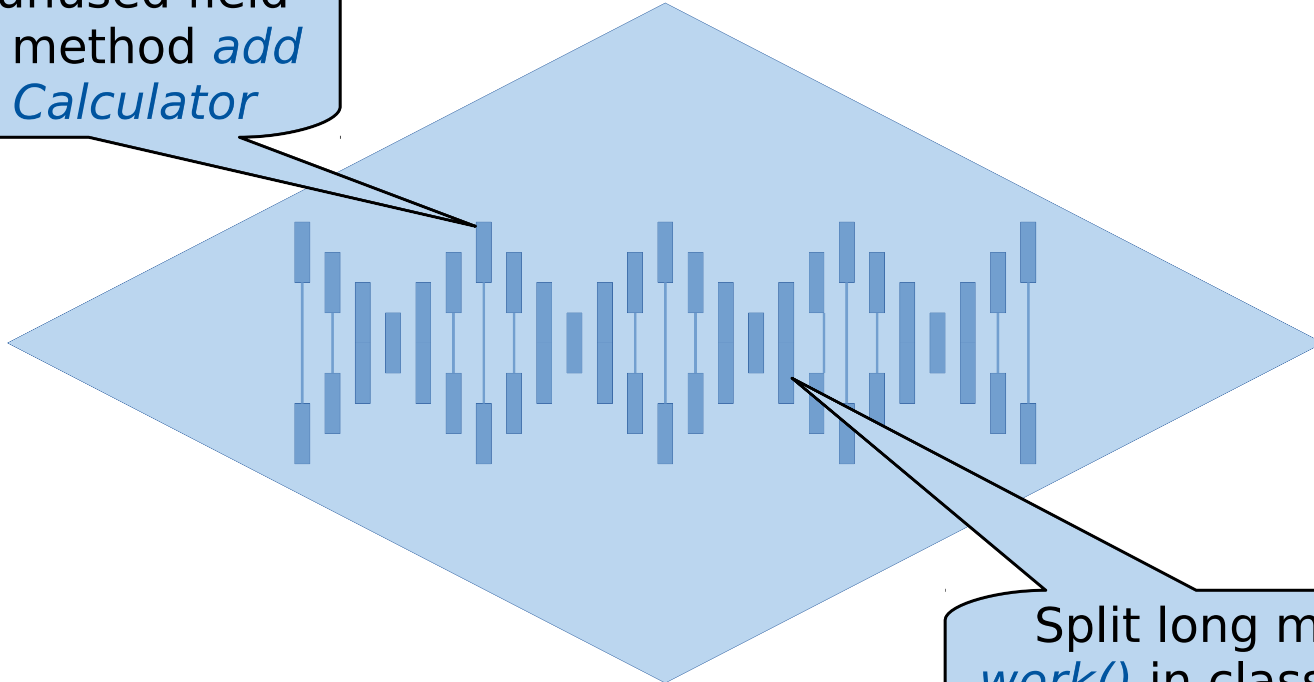
Genetic Programming



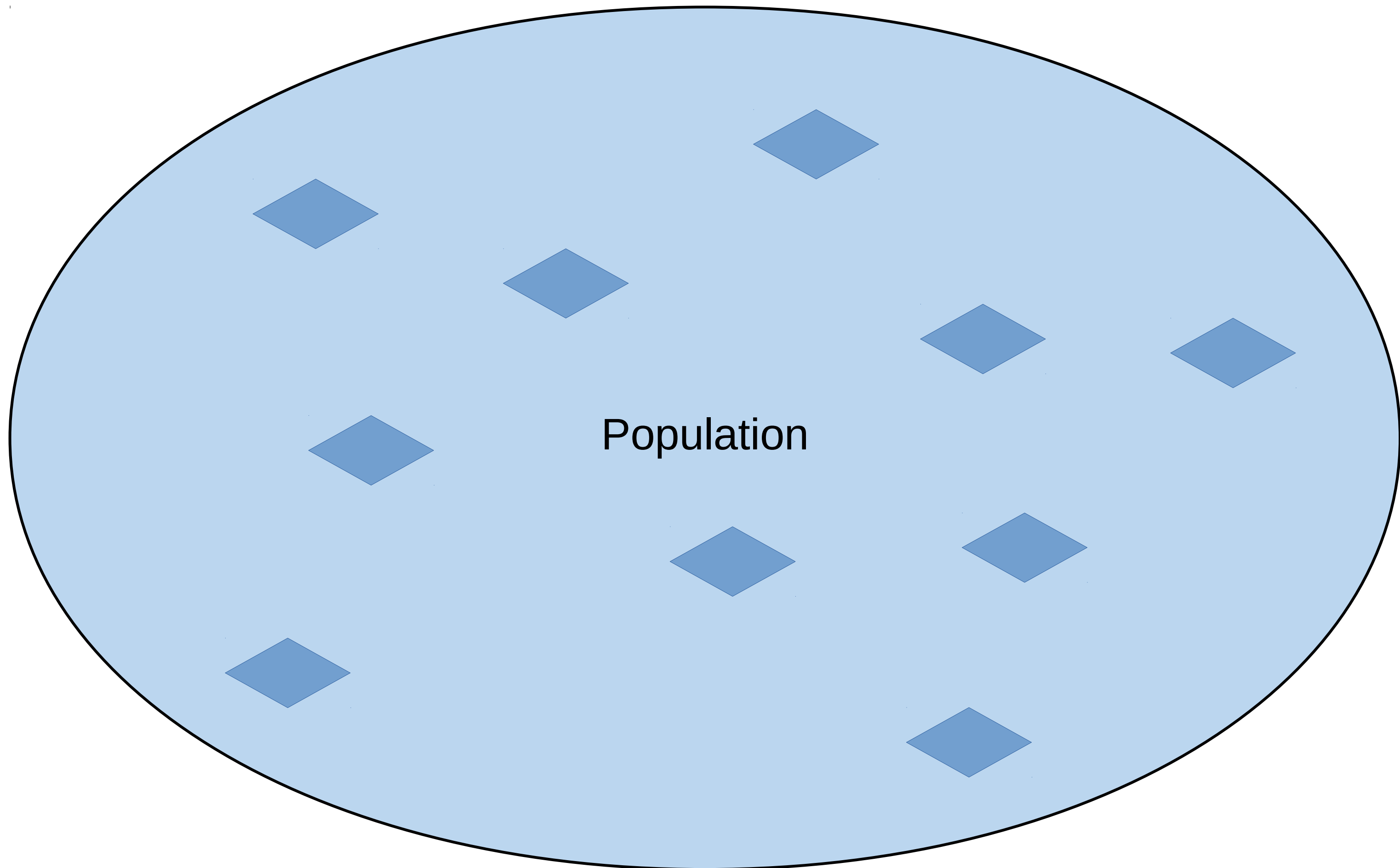
Genetic Programming - Refactoring Sequence



Genetic Programming - Refactoring Sequence



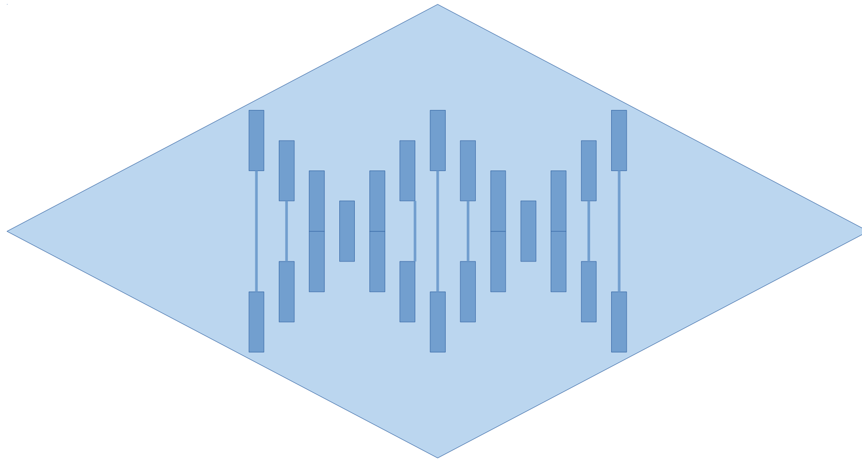
Genetic Programming - Population

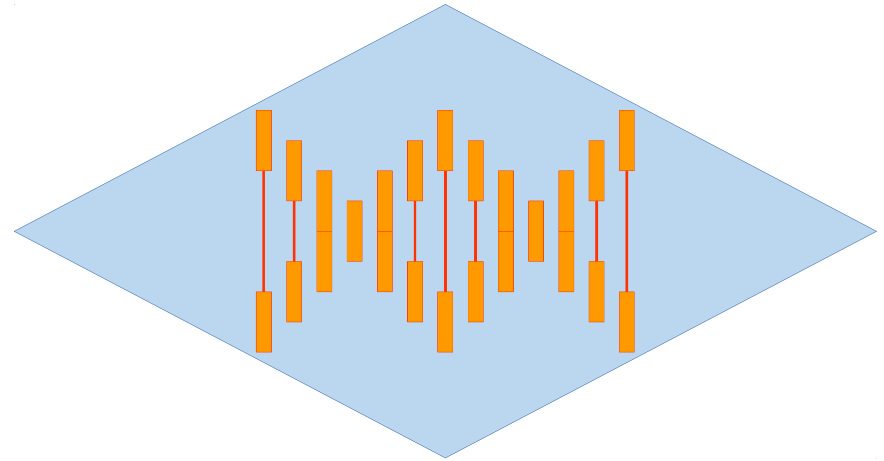


Software Construction

RWTH AACHEN UNIVERSITY

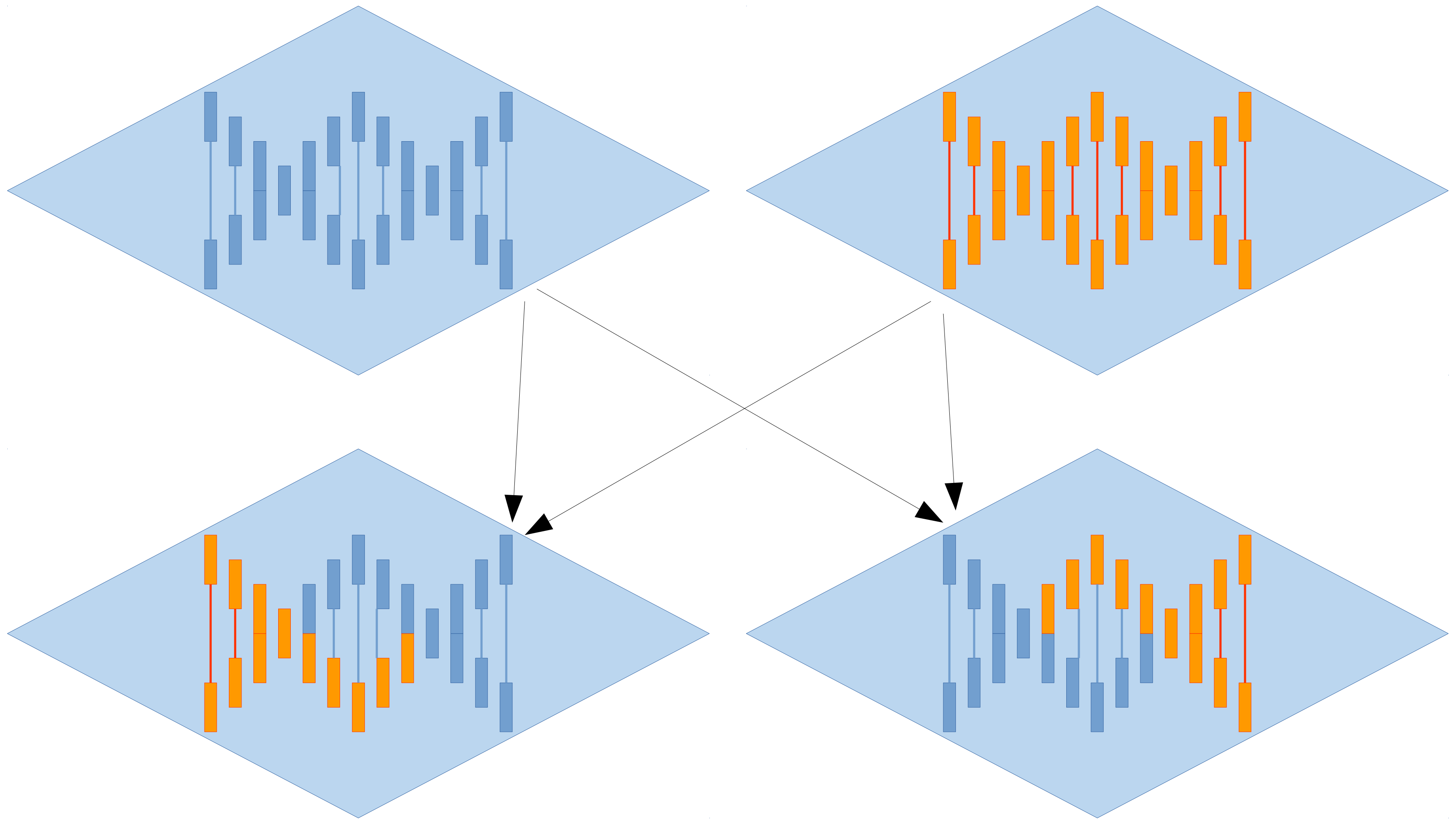
Genetic Programming - Recombination





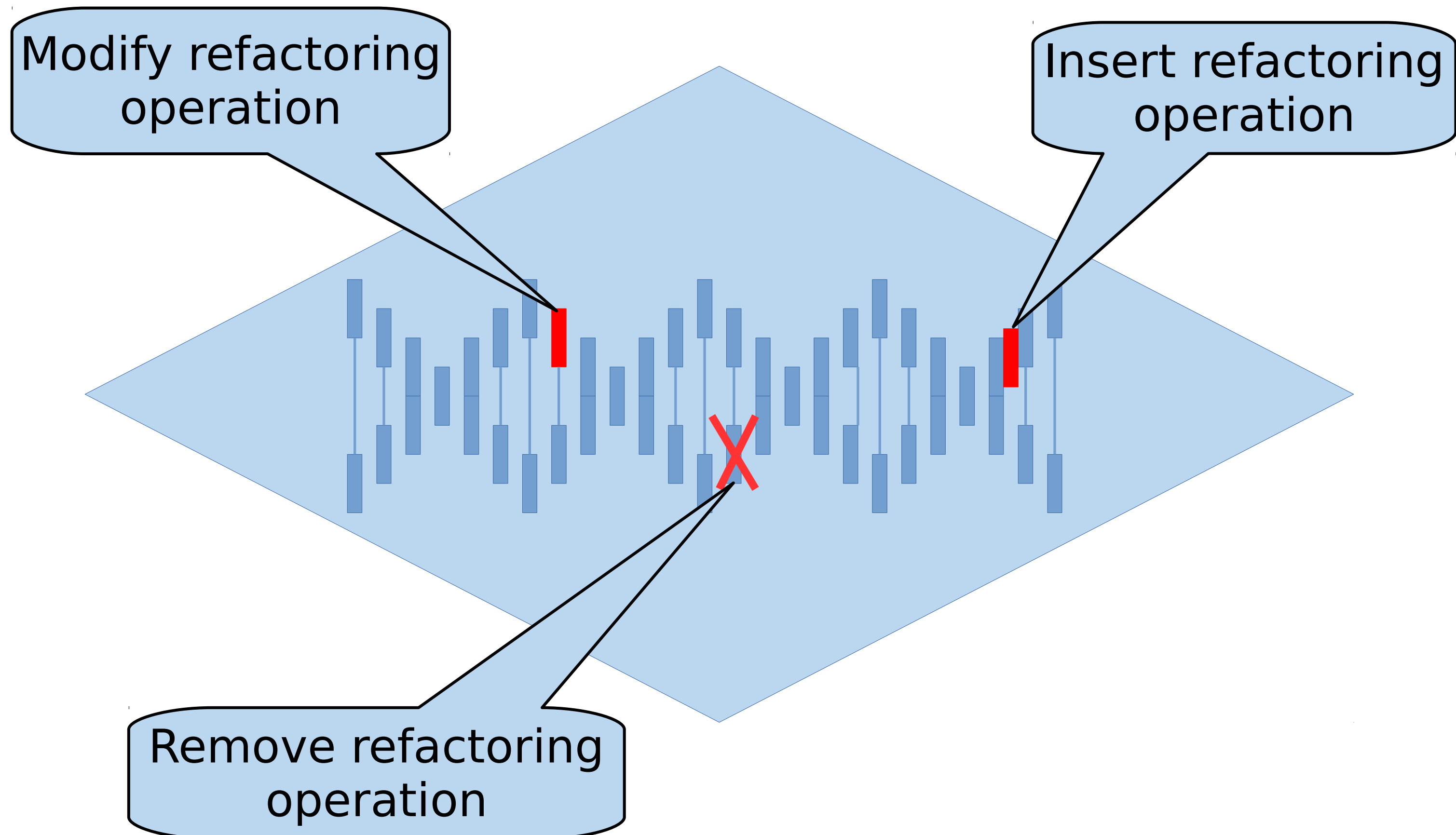
SWC Software Construction | RWTH AACHEN UNIVERSITY

Genetic Programming - Recombination

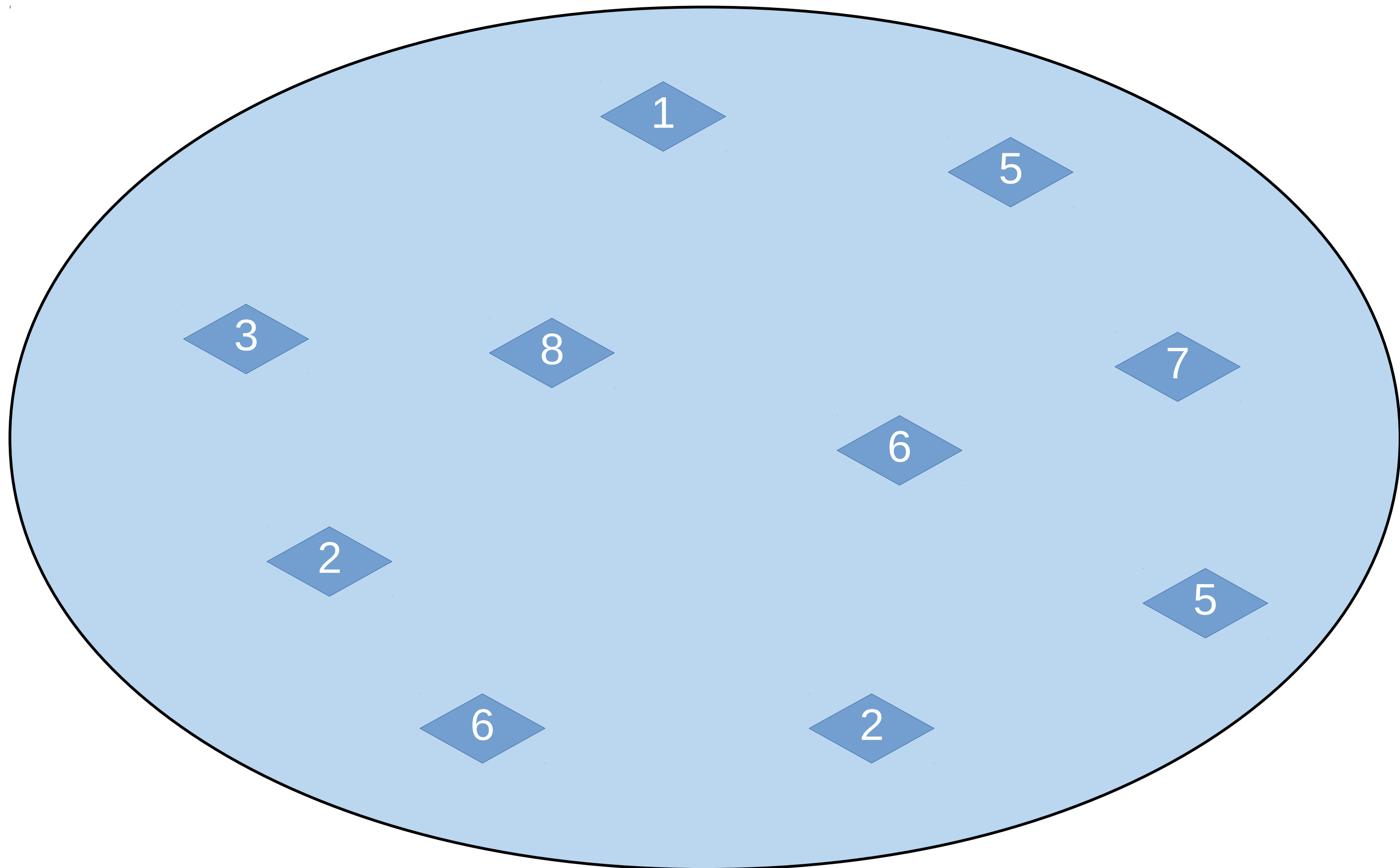


SWC Software Construction | RWTH AACHEN UNIVERSITY

Genetic Programming - Mutation



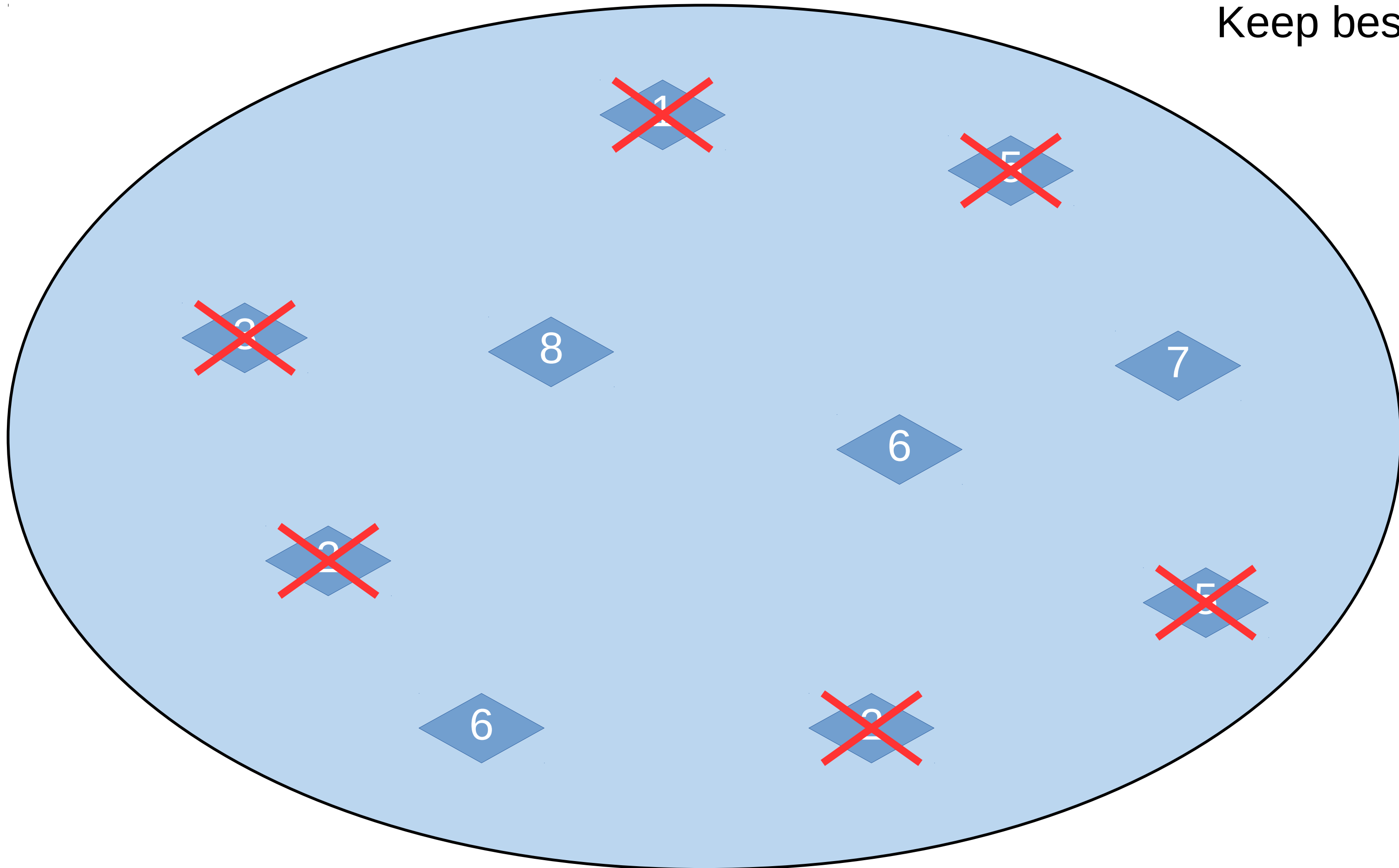
Genetic Programming - Selection



SWC Software Construction | RWTH AACHEN UNIVERSITY

Genetic Programming - Selection

Keep best 40%



SWC Software Construction | RWTH AACHEN UNIVERSITY

Genetic Programming - Generations

- Recombine and mutate remaining individuals
  → next generation

- Solution with sufficient quality has been found
  → stop

SWC Software Construction | RWTH AACHEN UNIVERSITY

Conclusion.

Automated Refactoring - Key Findings

SWC Software Construction | RWTH AACHEN UNIVERSITY

Automated Refactoring - Key Findings

- Both techniques: systematic trial and error

Software Construction
RWTH AACHEN UNIVERSITY

Automated Refactoring - Key Findings

- Both techniques: systematic trial and error
- No guarantee to find good/any solution

Automated Refactoring - Key Findings

- Both techniques: systematic trial and error
- No guarantee to find good/any solution
- Combinatorial optimization
  → iterative small-scale changes

Automated Refactoring - Key Findings

- Both techniques: systematic trial and error
- No guarantee to find good/any solution
- Combinatorial optimization
  → iterative small-scale changes
- Genetic programming
  → large-scale refactorings are possible

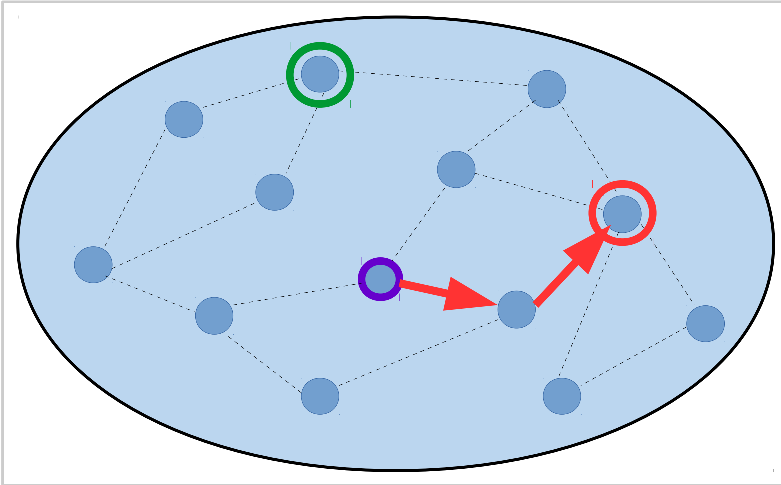
Automated Refactoring - Key Findings

- Both techniques: systematic trial and error
- No guarantee to find good/any solution
- Combinatorial optimization
  → iterative small-scale changes
- Genetic programming
  → large-scale refactorings are possible
- Most tools only work with Java programs

Summary

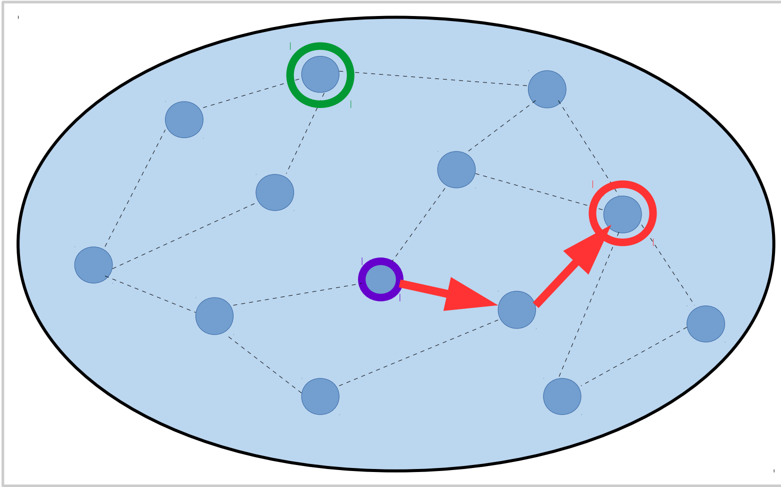
SWC Software Construction | RWTH AACHEN UNIVERSITY

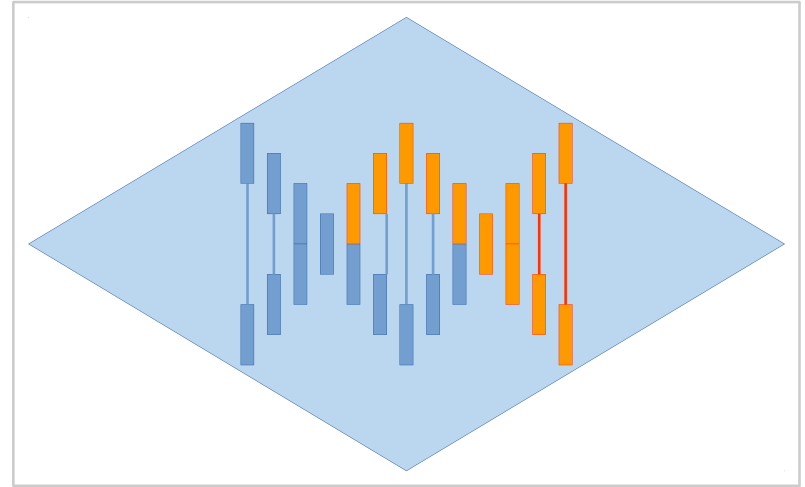
Summary



Software Construction
RWTH AACHEN UNIVERSITY

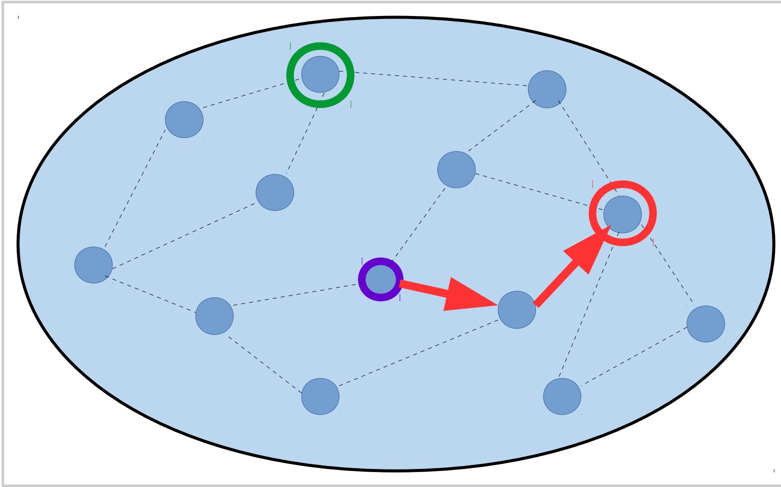
Summary

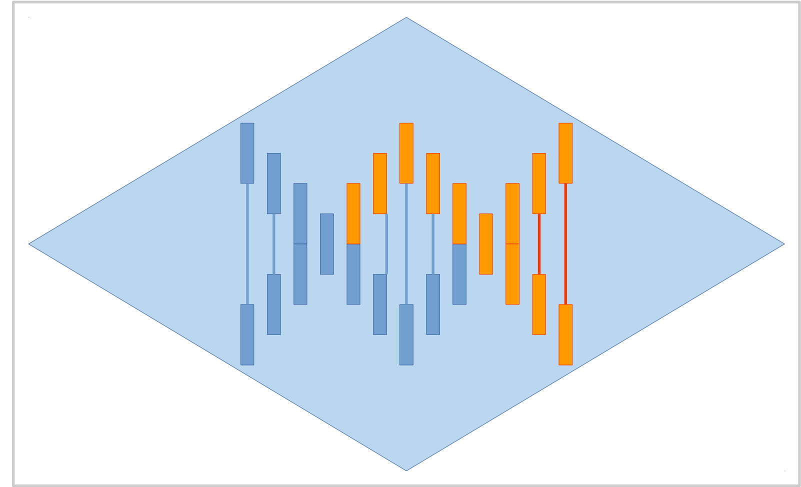




Software Construction
RWTH AACHEN UNIVERSITY

Summary





Would you use an automated refactoring tool
for your programming projects?

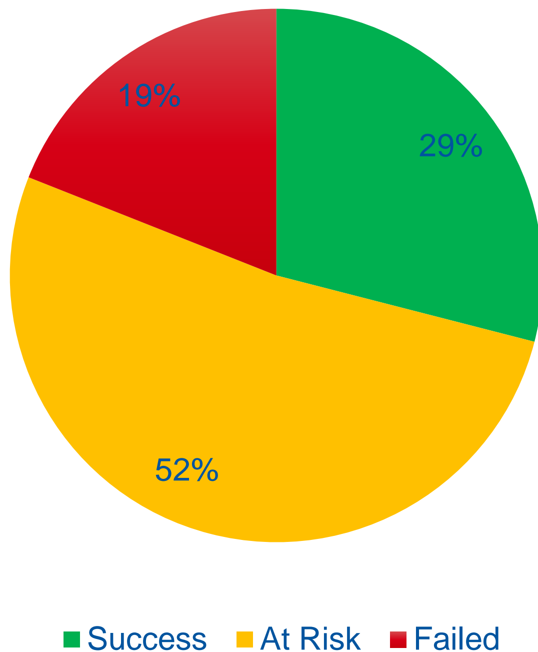
SWC Software Construction | RWTH AACHEN UNIVERSITY

Matthias Hansen

matthias.hansen@rwth-aachen.de

Metrics in Agile Projects — Does that matter?

Lots of Software projects fail

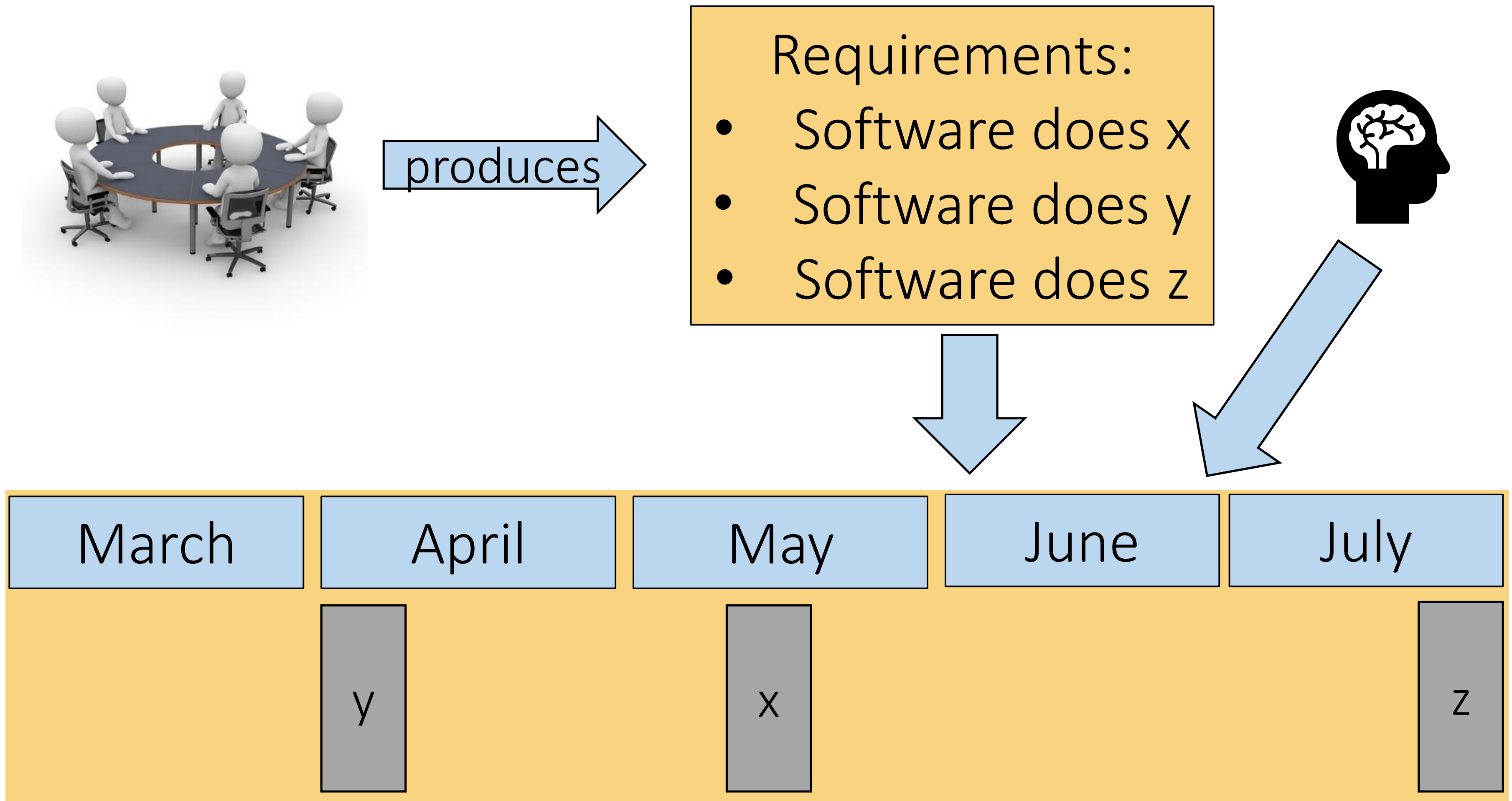


Reasons cited for Failure:

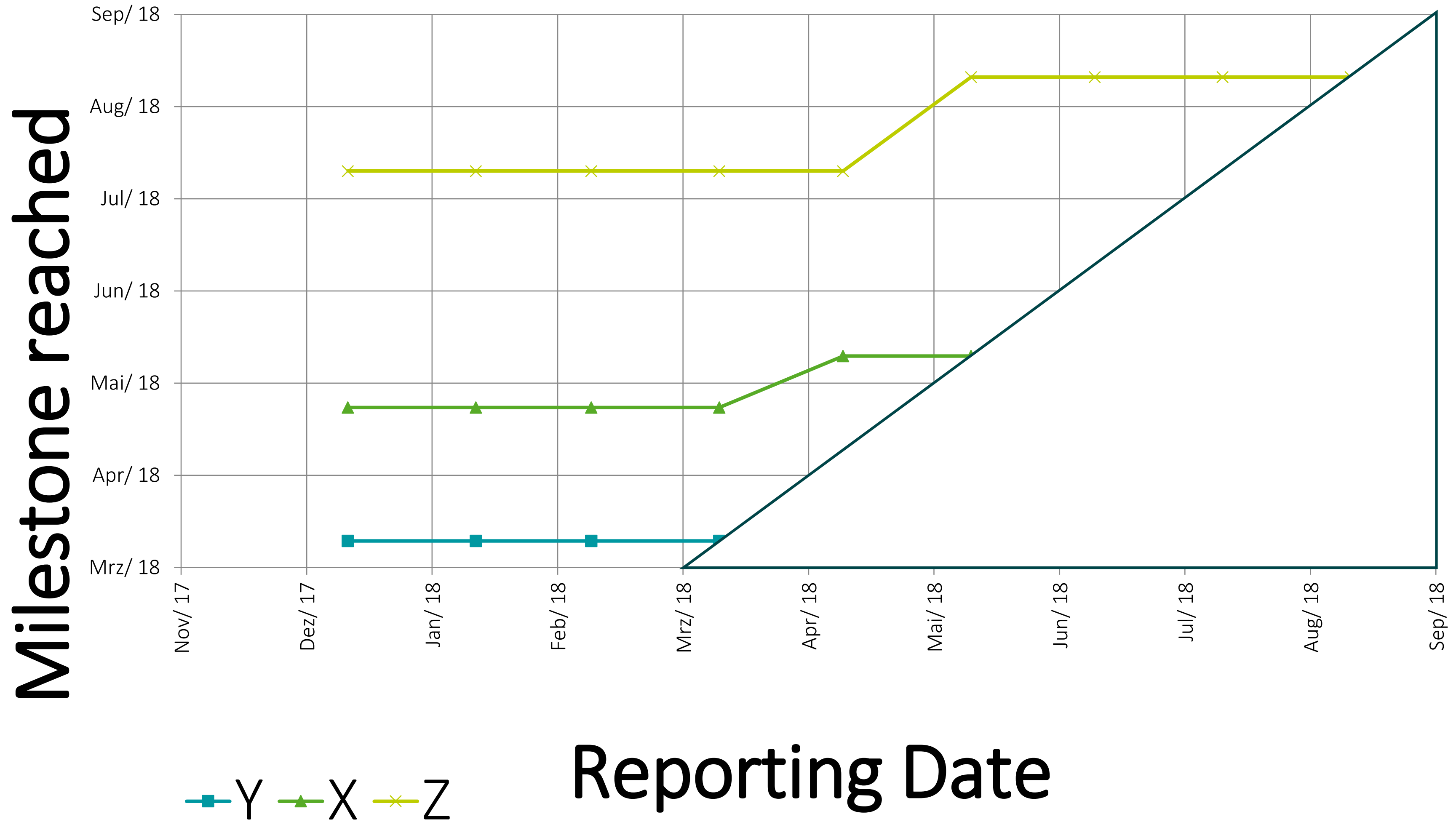
- Unclear requirements
- Users not incorporated
- Requirements change

Data from Standish Group CHAOS Report 2015

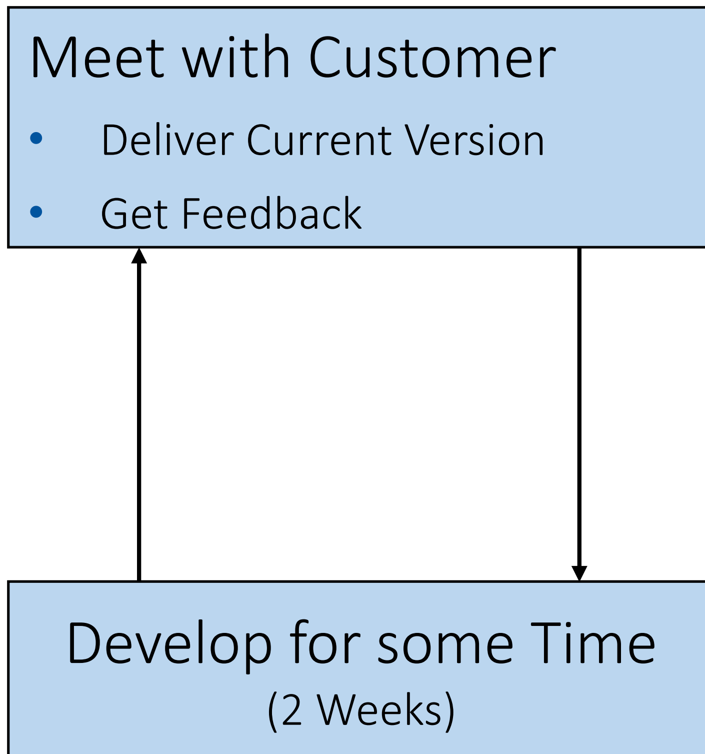
Traditional Project Management Approach



Monitoring via Milestone Analysis

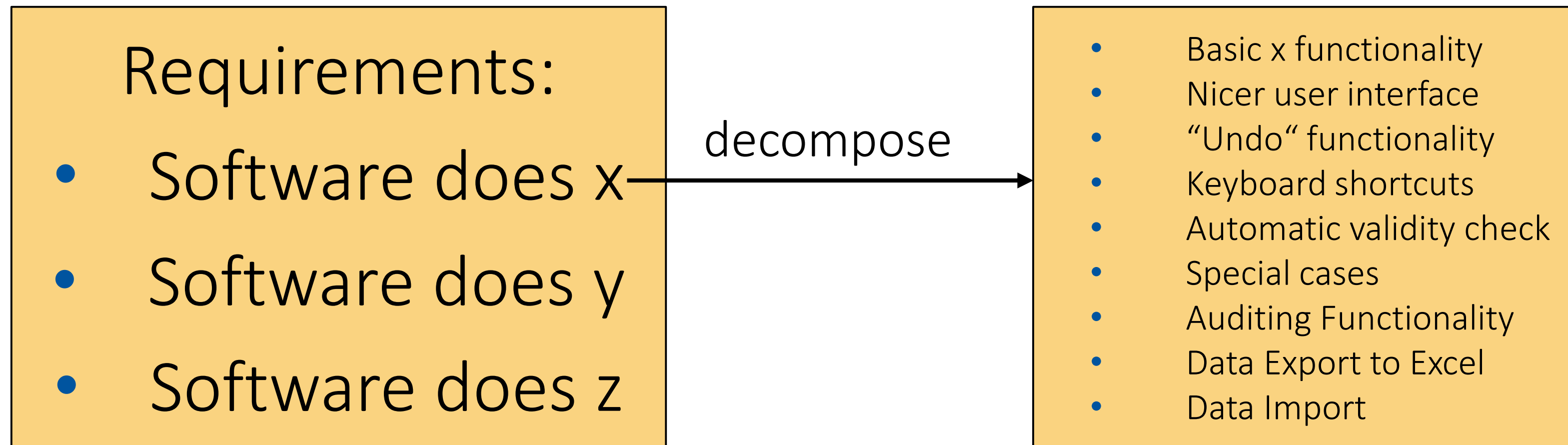


Agile Approach



Framework	Problem that is Solved
Xtreme Programming	Technical Preconditions
Scrum	Meetings, Roles & Terminology
Kanban	Process Evolution, Metrics
SAFe	Scaling to Company

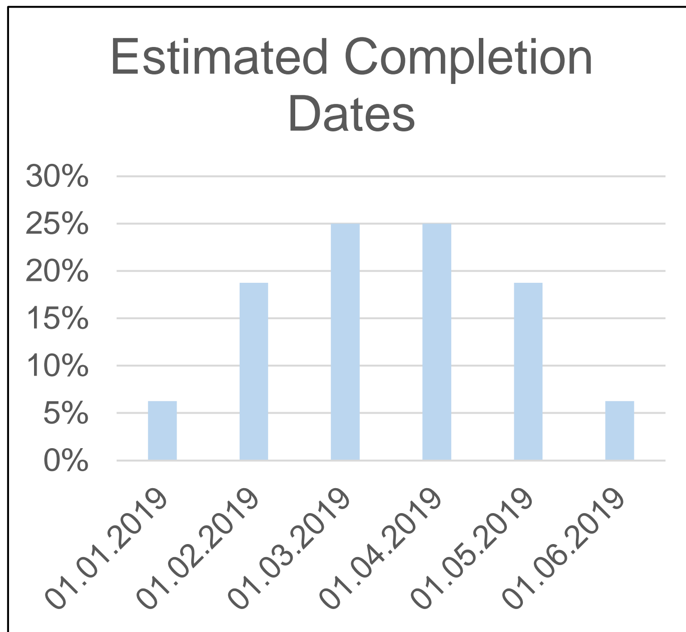
How to measure progress in Agile Projects

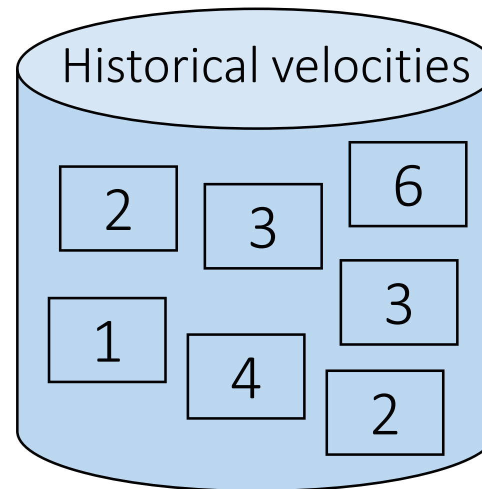


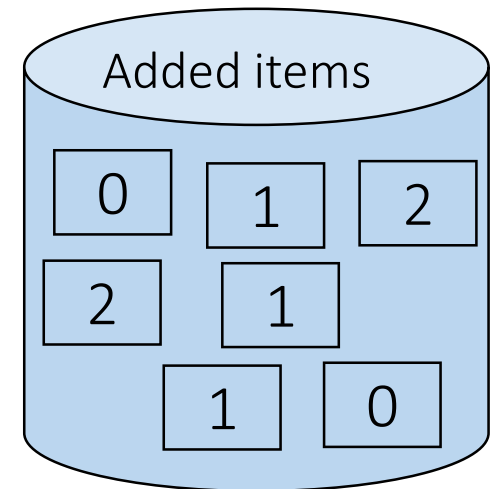
Velocity: # of items done between 2 meetings

Important:
Also account for changes in requirements

The Problem with using Averages and how to Counteract







Solution: Use e.g. 90th Percentile of Estimates

SWC Software Construction | RWTH AACHEN UNIVERSITY

Lessons Learned

- “Adapting to Change over Following a Plan”
- Plan “How much?” not “What?” or “How?”
- Plan with Empirical Data
- Explicitly Model Uncertainty

Matthias Hansen

matthias.hansen@rwth-aachen.de

Metrics in Agile Projects — Does that matter?

Literature

- D.J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business.* Blue Hole Press, 2010.
- K. Beck. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, 2000.
- K. Beck et al. *Manifesto for Agile Software Development*, 2001. http://agilemanifesto.org
- K. Schwaber and J. Sutherland. *The Scrum Guide*. <http://www.scrum.org/resources/scrum-guide>

Features of Combinatorial Testing Tools

A Literature Review

Joshua Bonn

```
if (OS == “Windows“)
  if (Browser == “Chrome“) {
    //Bad code
  } else {
    //Good code
  }
} else {
  //Good code
}
```

OS	Browser	Ping	Speed
Windows	Chrome	10 ms	1 KB/s
Linux	Edge	100 ms	10 KB/s
MacOS	Firefox	1000 ms	100 KB/s
Android	Safari		1000 KB/s
iOS			

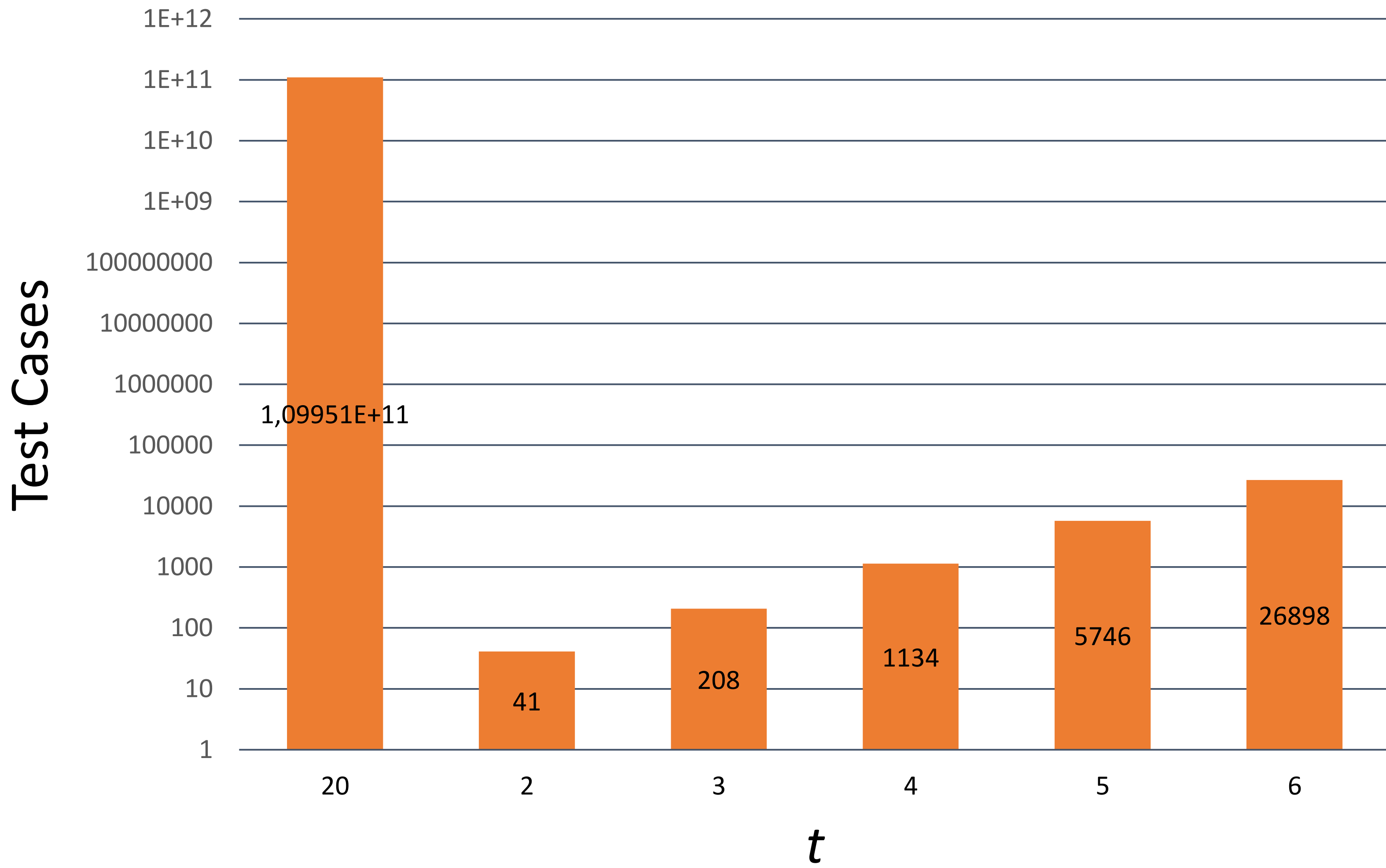
240 test cases!

95 2-way combinations

Linux Firefox 10 ms 1000 KB/s

6 2-way combinations

20 test cases



Test Cases
1E+12
1E+11
1E+10
1E+09
100000000
10000000
1000000
100000
10000
1000
100
10
1
1,09951E+11
41
208
1134
5746
26898
20
2
3
4
5
6
t

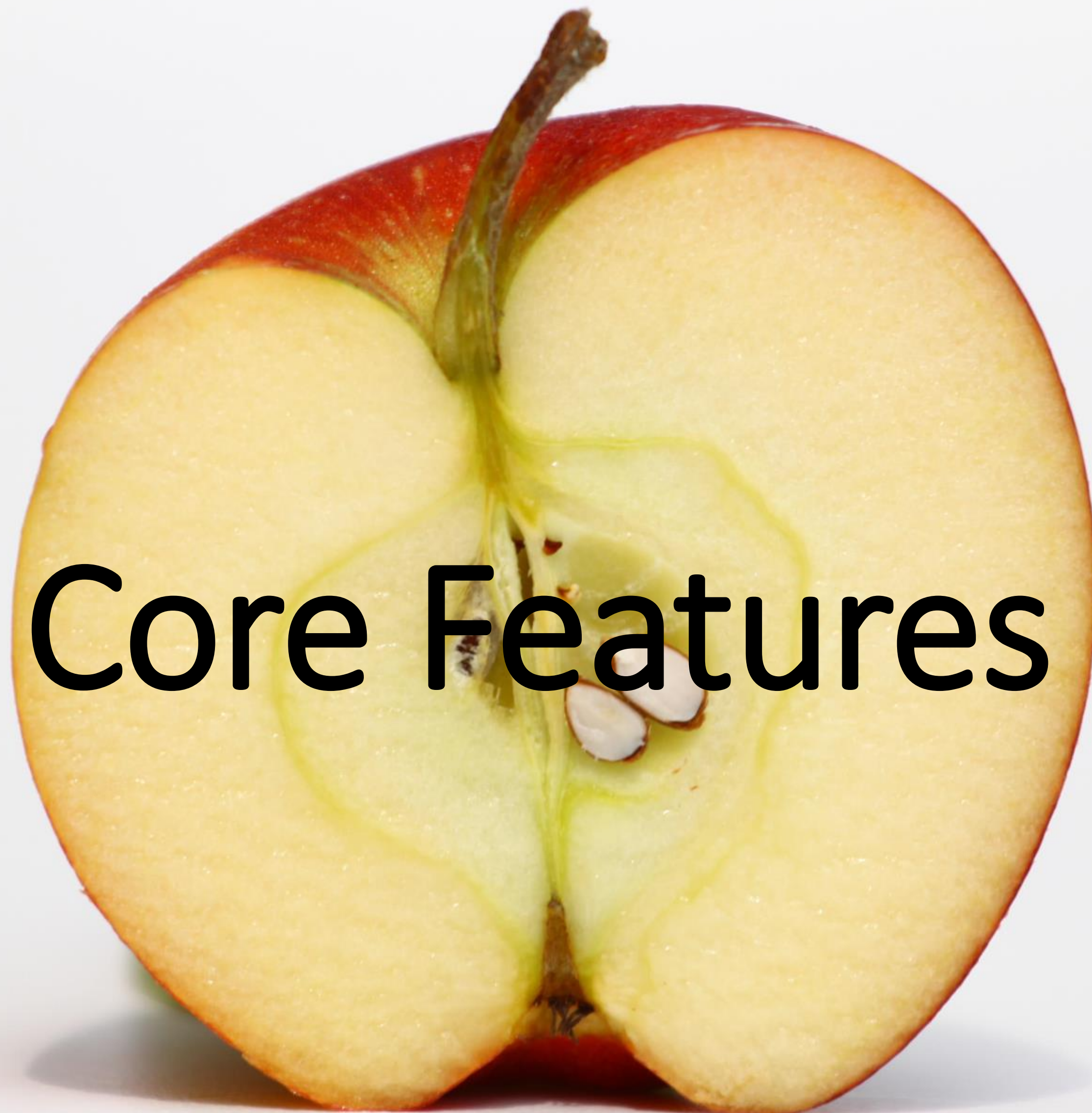
Tools

CATS	AETG	PairTest	TConfig
AllPairs (Perl)	Pro-Test	Jenny	TestCover
TVG	TESTONA	AllPairs (Python)	PICT
rdExpert	OATSGen	ATD	ACTS
IPO-s	VPTAG	FOCUS	Hexawise
PictMaster	NTestCaseBuilder	Tcases	Pairwiser
Nunit	ecFeed	JCUnit	CITLAB
TOSCA	Pairwise (Ruby)	Pairwise (RMN)	JCombinatorial

Research Questions

- What features are available?
- How are these features distributed?

Core Features



https://www.hongkiat.com/blog/high-quality-fruits-stock-photos/

Parameters | ACTS

Parameters | Constraints | Relations

System Name Game

System Parameter

Parameter Name

Parameter Type Enum

Parameter Values

Selected Parameter **Enum**

Simple Value

Range Value 0 - 1

Add ->

<- Remove

Base Choice

Invalid Value

Add to Table

Saved Parameters

Parameter Name	Parameter Type	Parameter Value
OS	Enum	[Windows,Linux,MacOS,Android,iOS]
Browser	Enum	[Chrome,Edge,Firefox,Safari]
Ping	Number	[10,100,1000]
Speed	Number	[1,10,100,1000]

Remove Modify

Add System Cancel

```
OS: Windows, Linux, MacOS, Android, iOS
Browser: Chrome, Edge, Firefox, Safari
Ping: 10, 100, 1000
Speed: 1, 10, 100, 1000
```

Constraints | ACTS

Parameters | Constraints | Relations

Symbols

( ) = != > < <= >= && || => ! * / - % +

Parameters

Parameter Name	Parameter Type	Parameter Value
OS	Enum	[Windows,Linux,MacOS,Andr
Browser	Enum	[Chrome,Edge,Firefox,Safari]
Ping	Number	[10,100,1000]
Speed	Number	[1,10,100,1000]

Constraint Editor

Browser = "Edge" => (OS != "Linux" && OS != "MacOS")

Clear | Add Constraint

Added Constraints

Browser = "Safari" => (OS != "Windows" && OS != "Android" && OS !=...

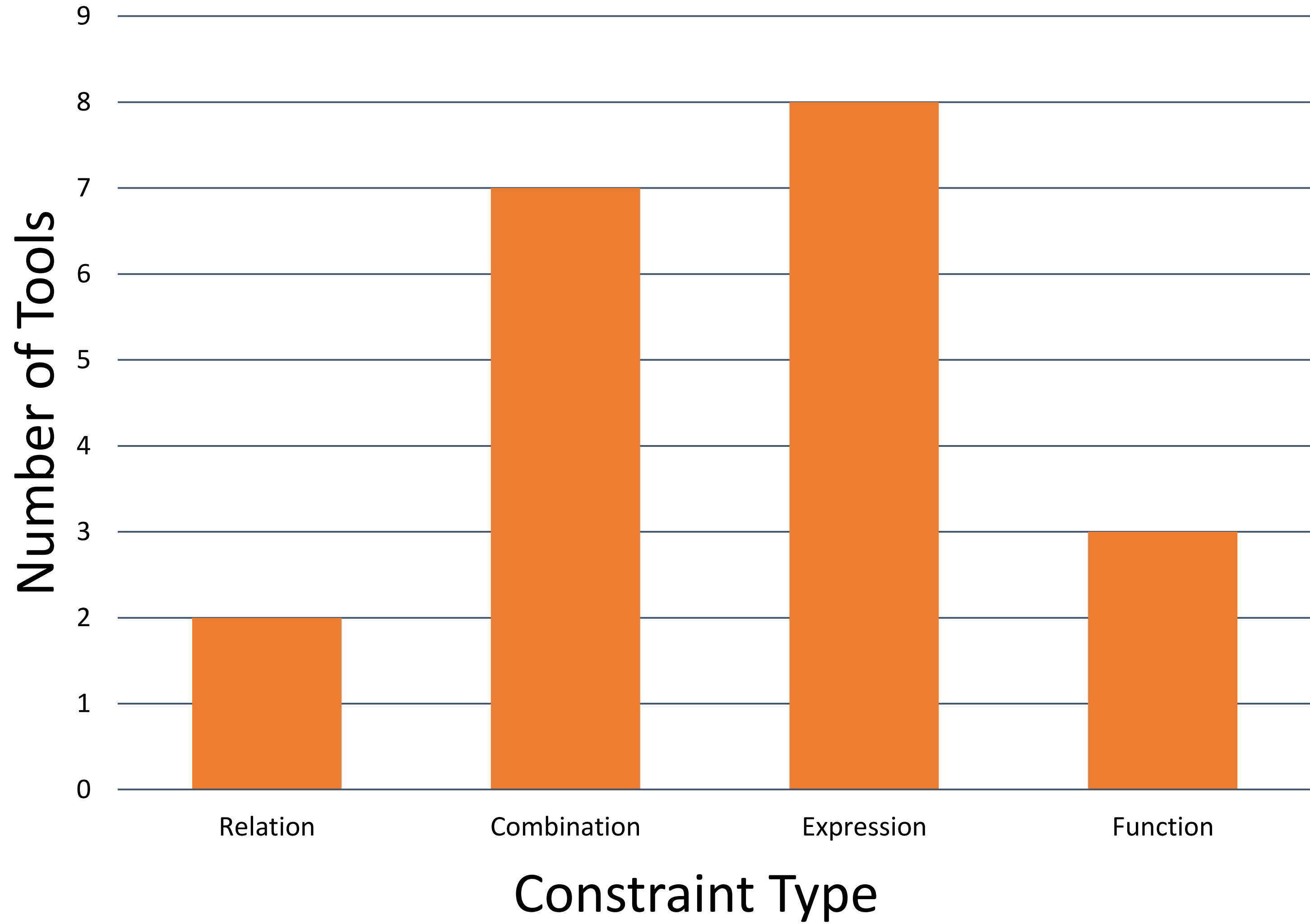
Browser = "Edge" => (OS != "Linux" && OS != "MacOS")

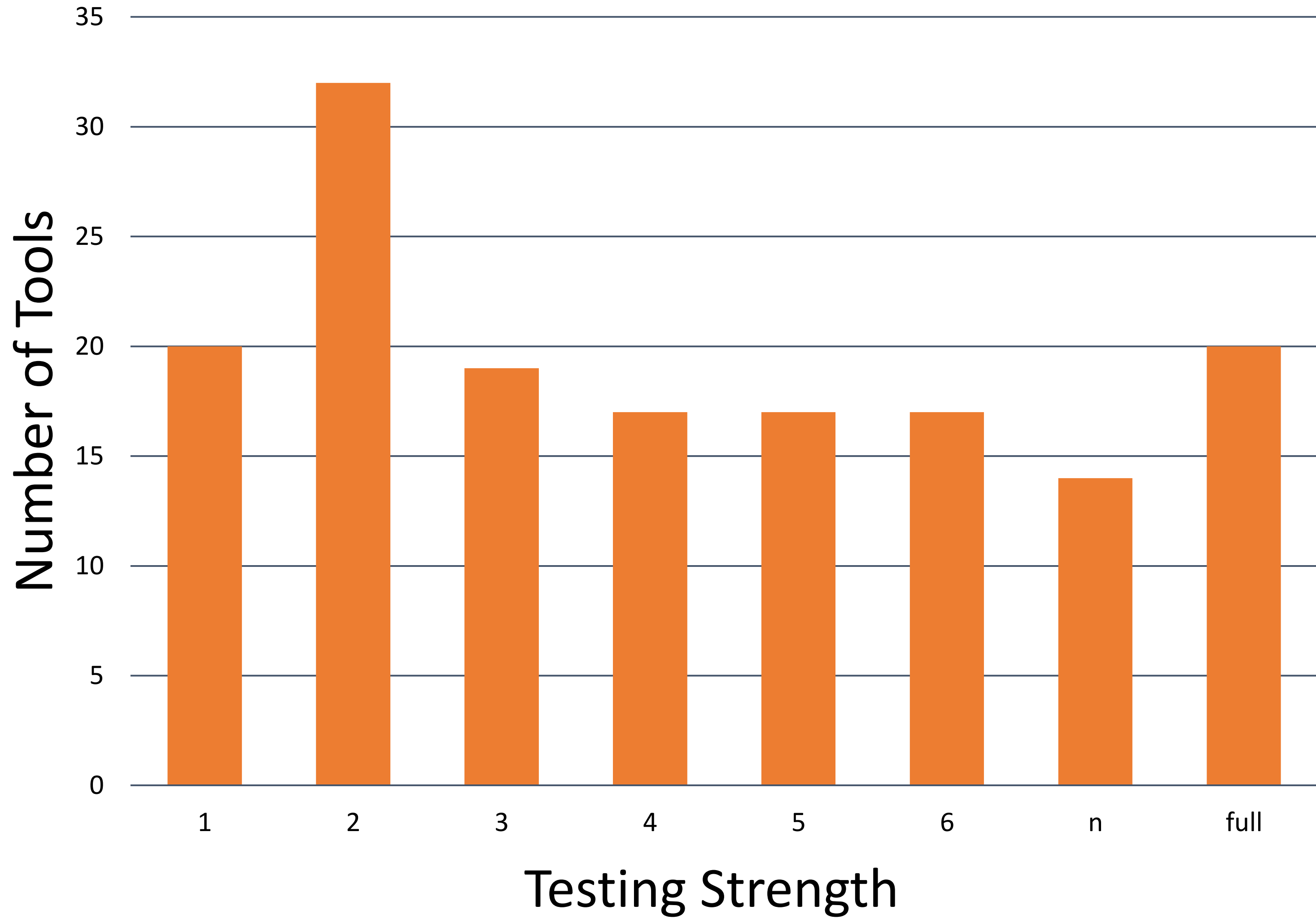
Remove | Load From File

Add System | Cancel

Constraints | PICT

```
IF [Browser] = “Edge“ THEN
    [OS] <> “Linux“ AND
    [OS] <> “MacOS“
```


9
8
7
6
5
4
3
2
1
0
Number of Tools
Relation
Combination
Expression
Function
Constraint Type


35
30
25
20
15
10
5
0
Number of Tools
1
2
3
4
5
6
n
full
Testing Strength

Advanced Features

https://www.hongkiat.com/blog/high-quality-fruits-stock-photos/

Relations | ACTS

Parameters | Constraints | Relations

Parameters

OS
Browser
Ping
Speed

Strength

Add ->>

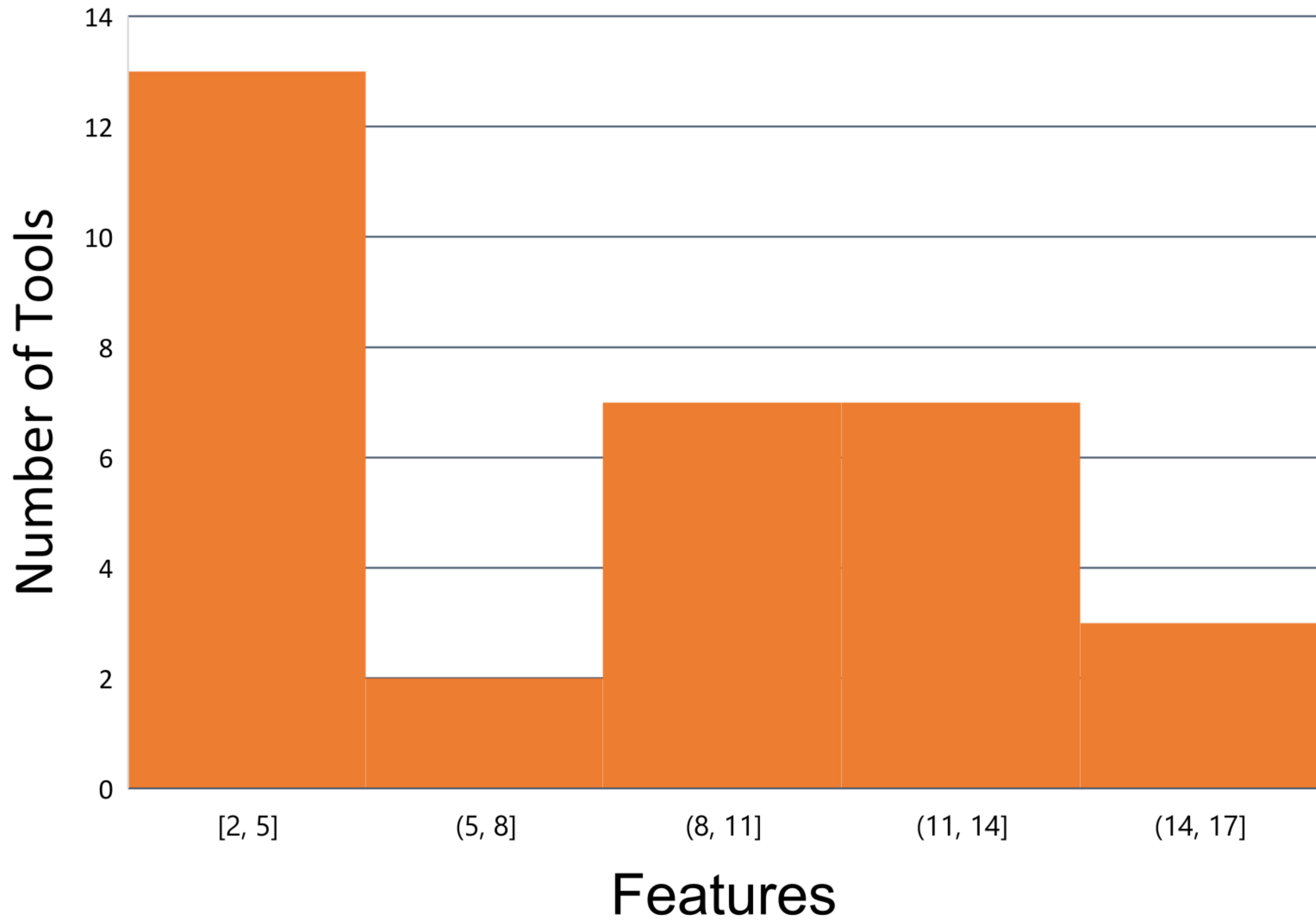
Remove

Strength	Parameter Names
3	Browser,Ping,Speed

Modify System

Cancel

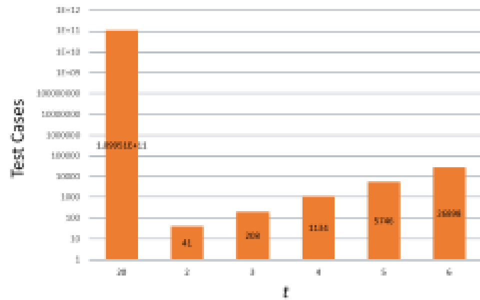
```
{Browser, Ping, Speed} @ 3
```


14
12
10
8
6
4
2
0
Number of Tools
[2, 5]
(5, 8]
(8, 11]
(11, 14]
(14, 17]
Features

Summary

- Fewer test cases through Combinatorial Testing
- Core features/advanced features
- Feature distribution

https://www.makingdifferent.com/wp-content/uploads/2014/11/thank-you-cartoon.png



5

Research Questions

- What features are available?
- How are these features distributed?

7

Parameters | PICT

```
OS: Windows, Linux, MacOS, Android, iOS
Browser: Chrome, Edge, Firefox, Safari
Ping: 10, 100, 1000
Speed: 1, 10, 100, 1000
```

10

Constraints | PICT

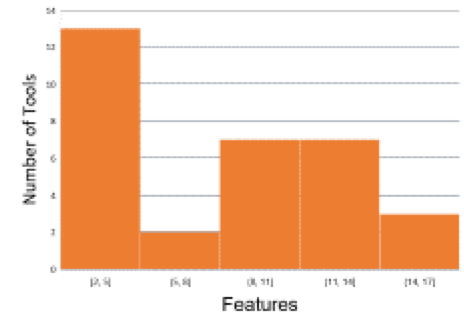
```
IF [Browser] = “Edge“ THEN
    [OS] <> “Linux“ AND
    [OS] <> “MacOS“
```

12

Sub-Models | PICT

```
{Browser, Ping, Speed} @ 3
```

17



18

Nils Wild, nils.wild@rwth-aachen.de

A meta model for maturity models

SWC Software Construction | RWTH AACHEN UNIVERSITY

1. The idea of maturity models
2. Properties of maturity models
3. Types of maturity models
4. The meta model
5. Conclusion

SWC Software Construction | RWTH AACHEN UNIVERSITY

Everything becomes mature over time

- We can influence the process of getting mature (unfortunately not physically)
- In case of products and projects we can and we have to control the process of getting mature
- Maturity models help to track and control that process

SWC Software Construction | RWTH AACHEN UNIVERSITY

Nursery school

- Become old enough for school

School

- Foreign language
- Math
- Physics
- History
- …

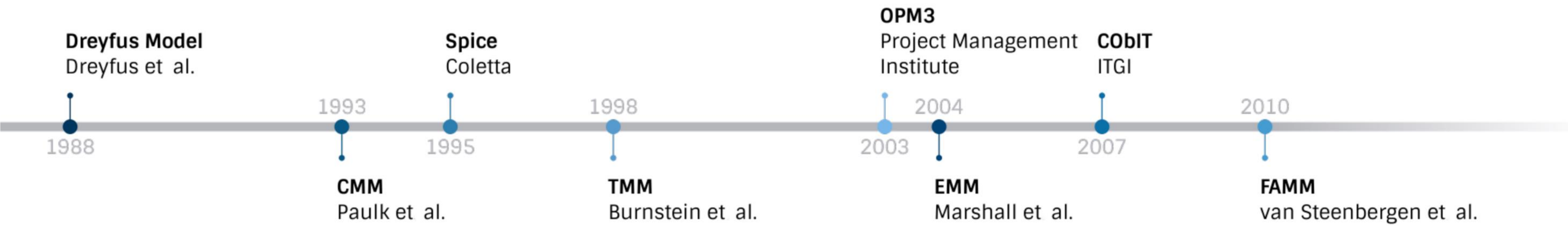
University

- Scientific work
- Specific expertise
- …

1. The idea of maturity models
2. Properties of maturity models
3. Types of maturity models
4. The meta-model
5. Conclusion

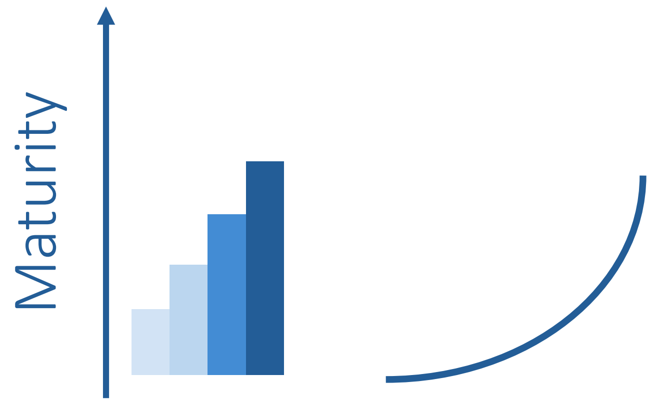
Software Construction

RWTH AACHEN UNIVERSITY

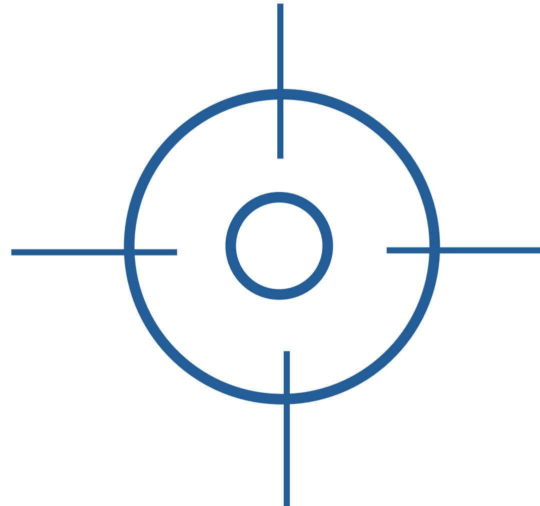


Based on van Looy et al.

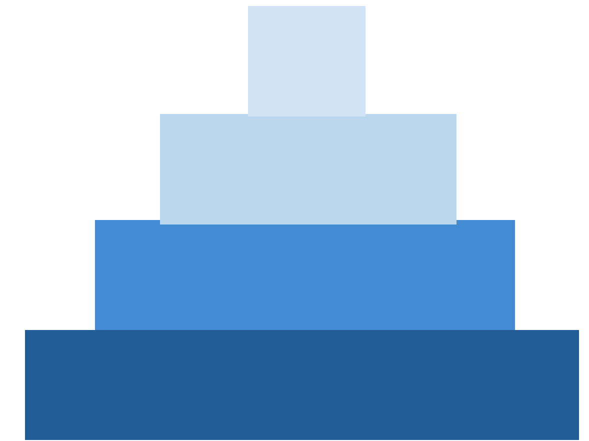
Properties of maturity models



type of maturity



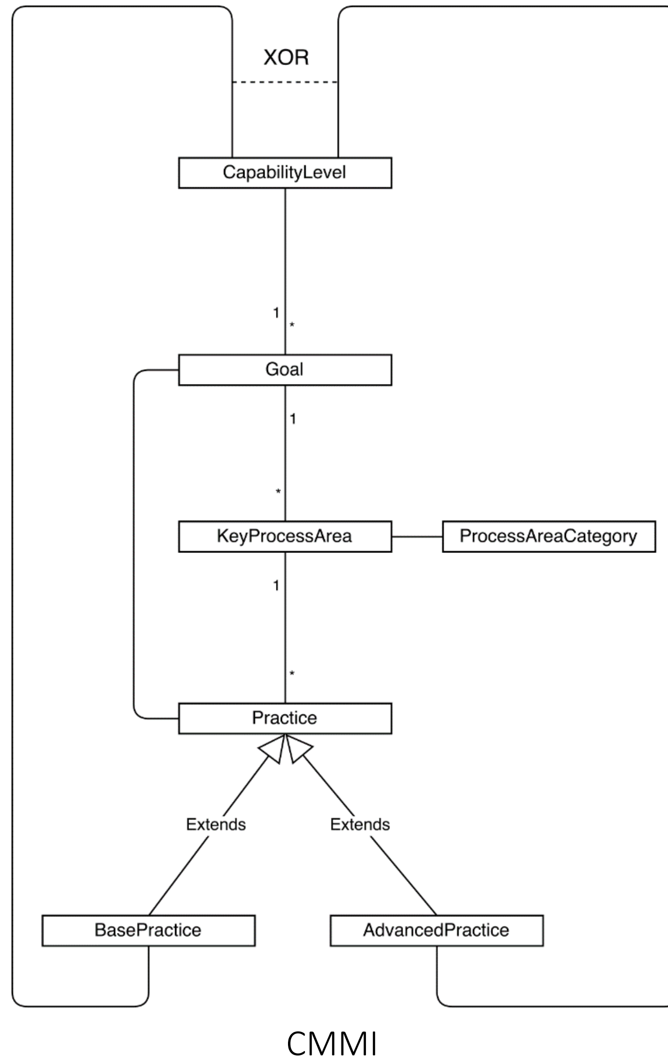
domains



#of levels

1. The idea of maturity models
2. Properties of maturity models
3. Types of maturity models
4. The meta-model
5. Conclusion

Staged / fixed-level

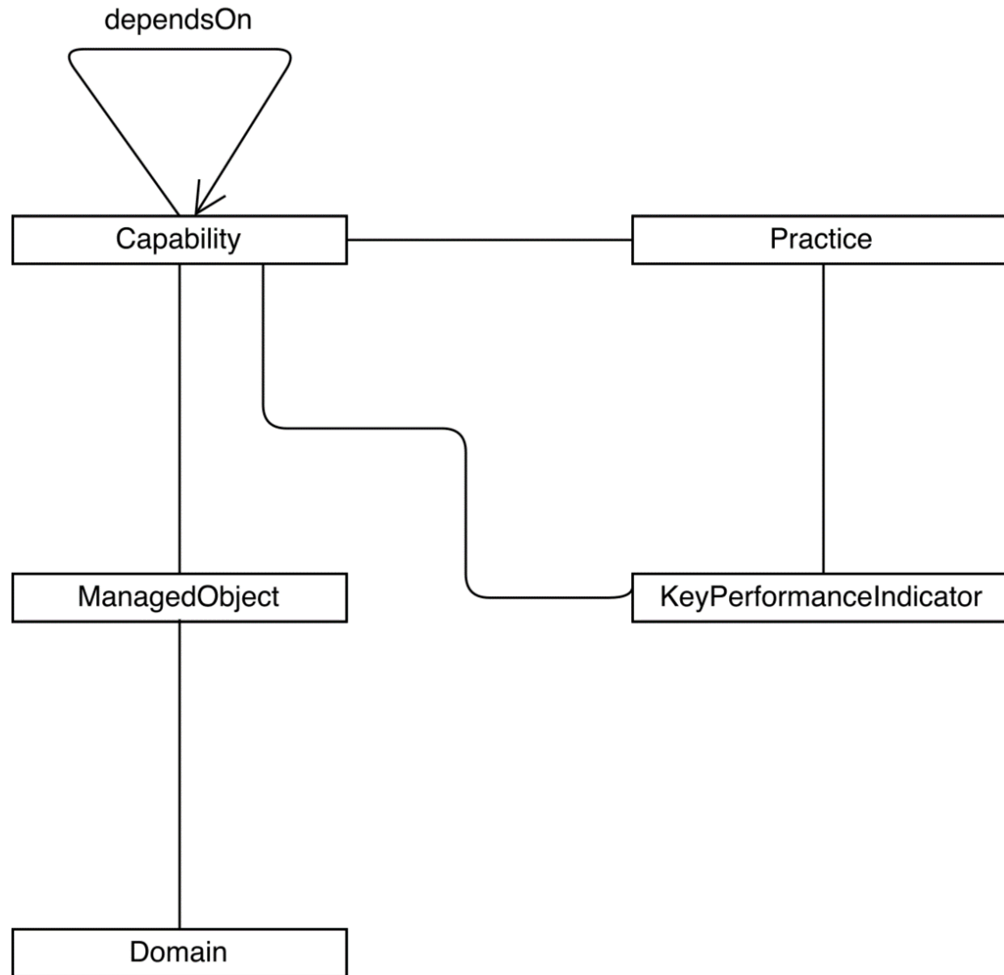


CMMI

- Subjects are key process areas (KPA)
- Goal is to pass the exam (discrete)
- Achieve a certain level of knowledge (capability)

SWC Software Construction | RWTH AACHEN UNIVERSITY

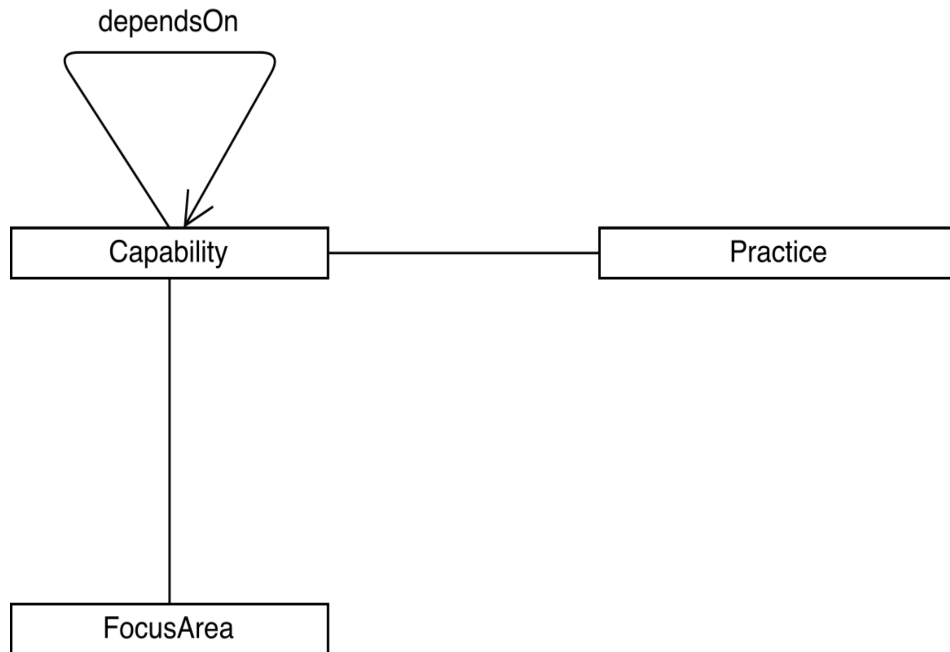
Continous



OPM3

- Subjects are managed objects
- Key performance indicator (KPI) is the number of vocabulary (continuous)
- Achieve a certain level of knowledge (capability)

FAMM



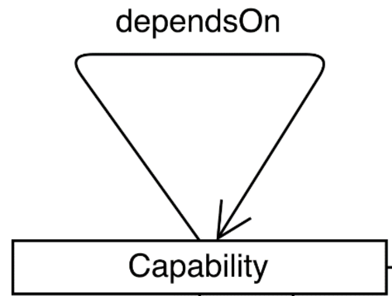
Maturity Scale / Focus Area	0	1	2	3	4	5	6
German		A			B		C
Maths			A	B			
Physics		A				B	

Each focus area has certain capabilities that can be enabled. The overall maturity is determined by the least mature area.

SWC Software Construction | RWTH AACHEN UNIVERSITY

1. The idea of maturity models
2. Properties of maturity models
3. Types of maturity models
4. The meta-model
5. Conclusion

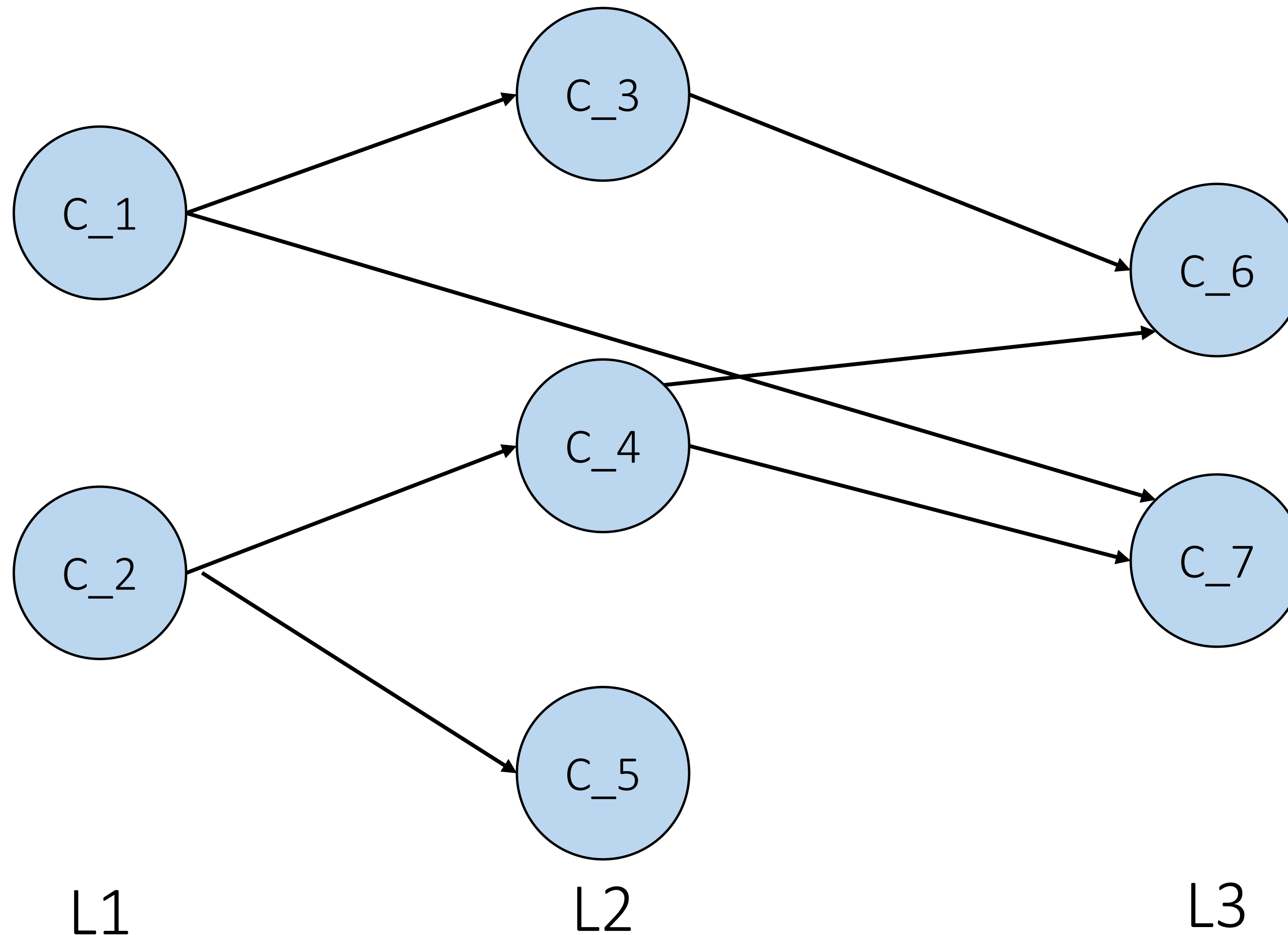
SWC Software Construction | RWTH AACHEN UNIVERSITY



Meta model

- Structure is similar across different types of maturity models
- Measurement of maturity differs but can be computed by traversing the *dependsOn* relationship

Computation of levels and maturity



- Marking algorithm determines levels
- Maturity can either be handled in a discrete or continuous manner

1. The idea of maturity models
2. Properties of maturity models
3. Types of maturity models
4. The meta-model
5. Conclusion

SWC Software Construction | RWTH AACHEN UNIVERSITY

Conclusion

- Maturity models share a common structure
- Mapping models to that common structure allows comparing arbitrary models
- “Levels” can be computed model independent
- Maturity can be computed with a model fitting function (discrete/continuous, levels/KPIs)

SWC Software Construction | RWTH AACHEN UNIVERSITY

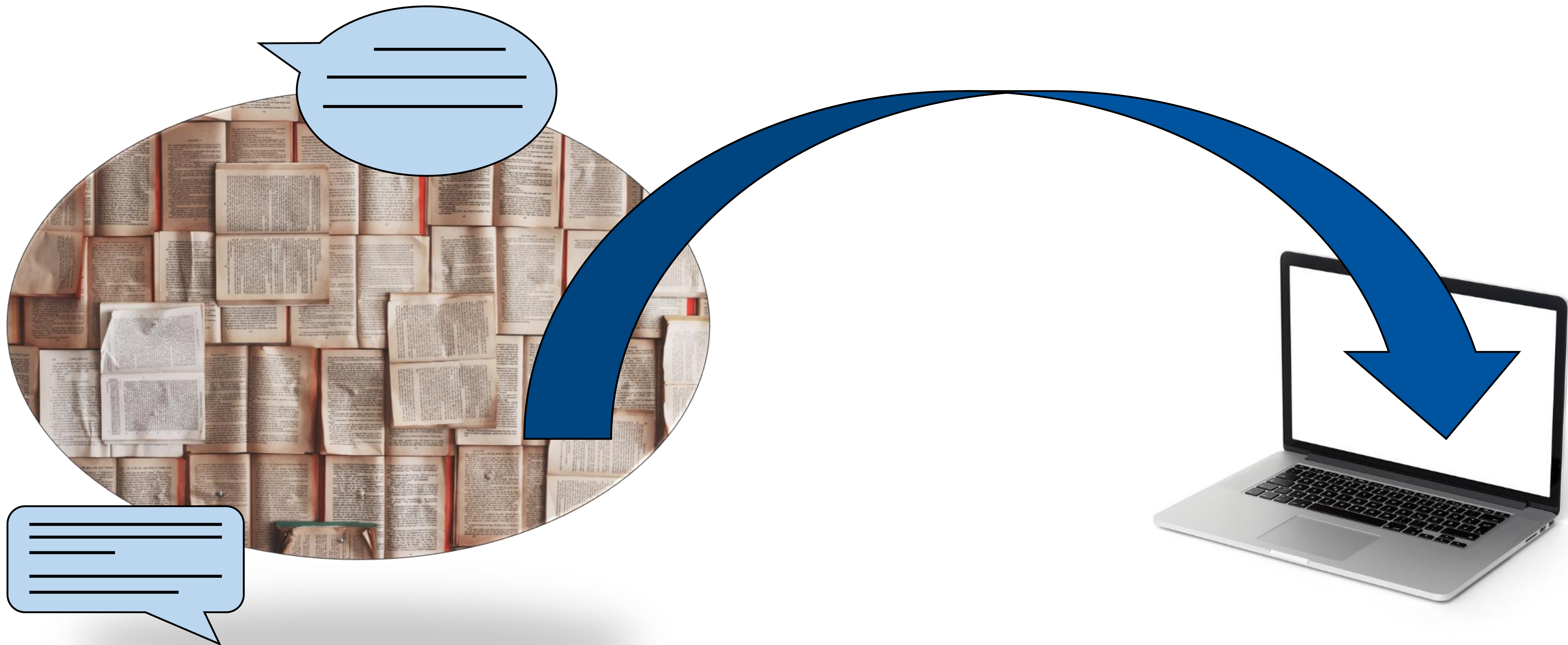
Nina Rußkamp,
nina.russkamp@rwth-aachen.de

Collaboration in Software Design

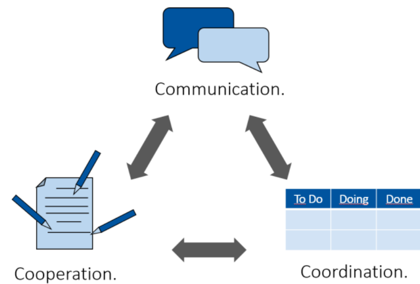
How to create UML models collaboratively

SWC Software Construction | RWTH AACHEN UNIVERSITY

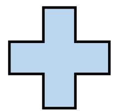
Imagine…



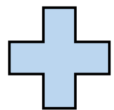
Outline



Collaboration

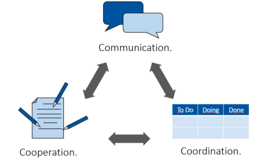


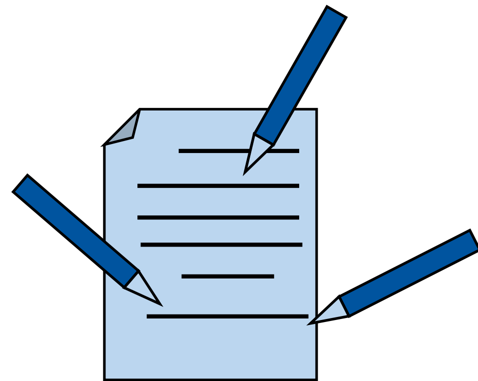
Collaborative UML Modelling



Collaborative UML Modelling Tools

Collaboration

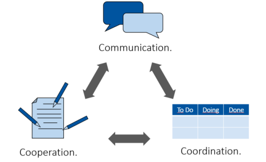




Cooperation.

Software Construction
RWTH AACHEN UNIVERSITY

Collaboration



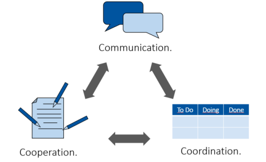
Communication.

Cooperation.

To Do	Doing	Done

Coordination.

Collaboration

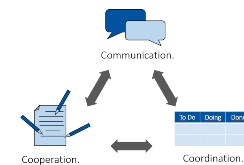


Process

State of Mind

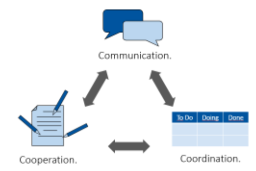
Product

SWC Software Construction | RWTH AACHEN UNIVERSITY

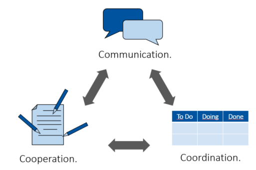


Collaboration. Why?

Limited Resources.



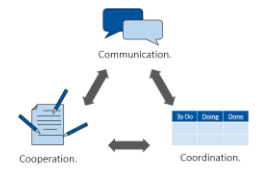
Collaboration. Why?



Limited Resources.

Collaboration. Why?

Software Complexity.

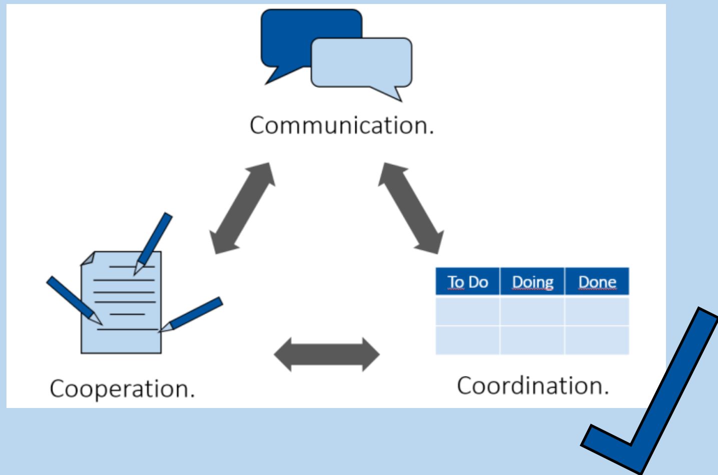
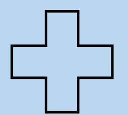
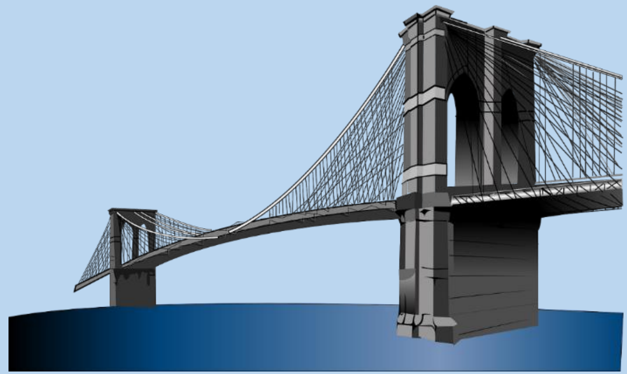
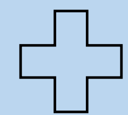


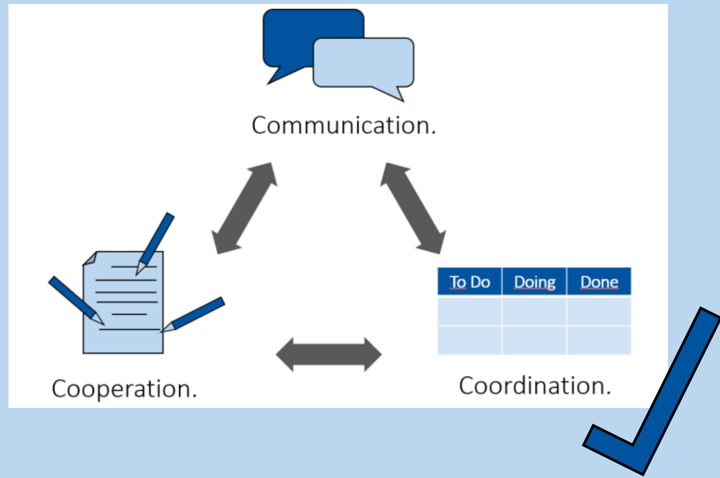
Limited Resources.

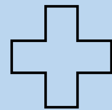
Collaboration. Why?

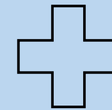
Software Complexity.

More than one person needed.

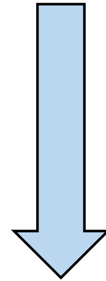

Communication.
Cooperation.
To Do
Doing
Done
Coordination.





Communication.
Cooperation.
To Do
Doing
Done
Coordination.





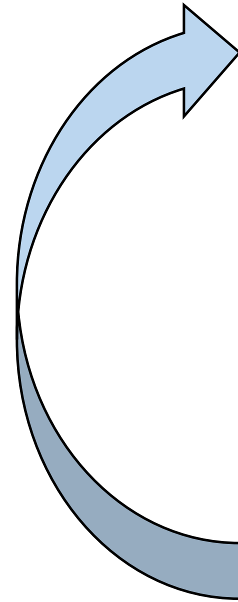
UML Modelling

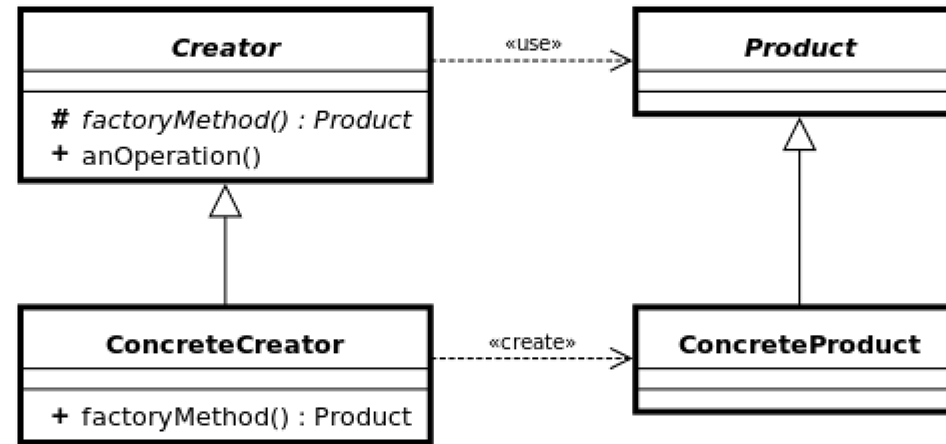


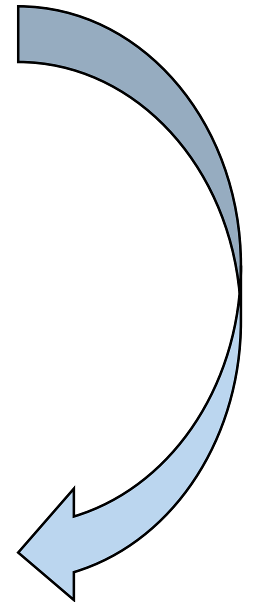
Software Solution.

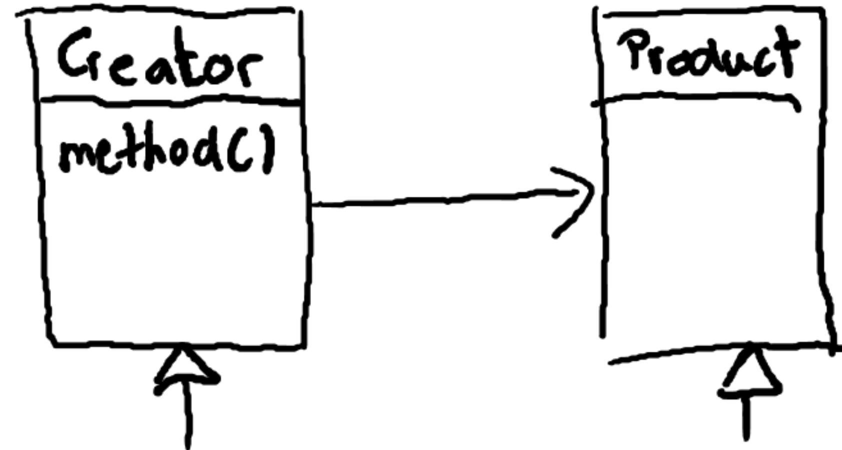
Software Problem.

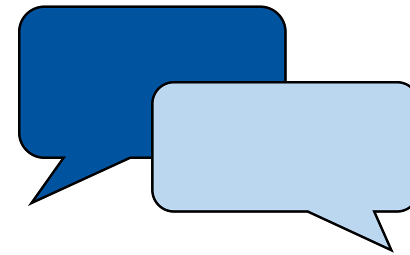
UML Modelling



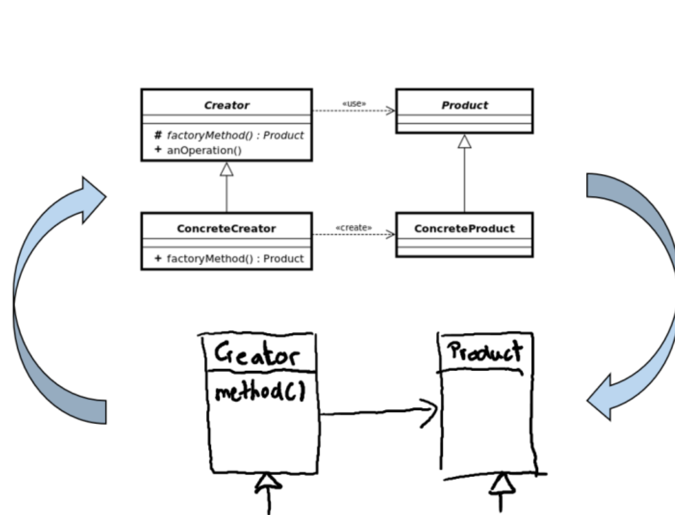


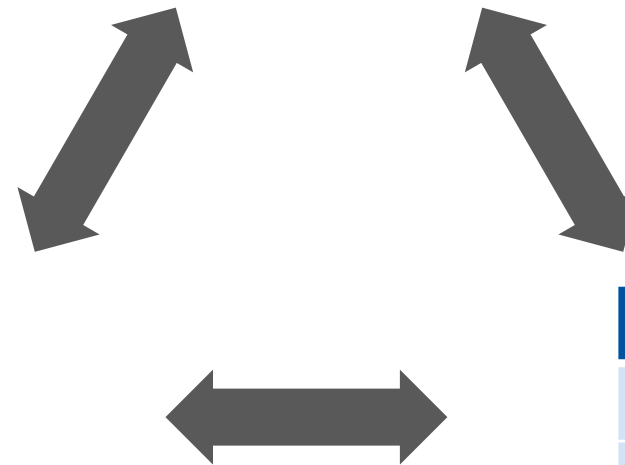






Communication.

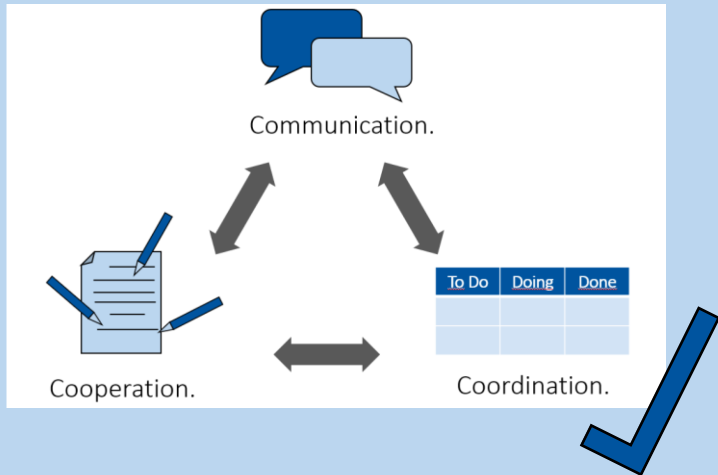


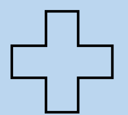


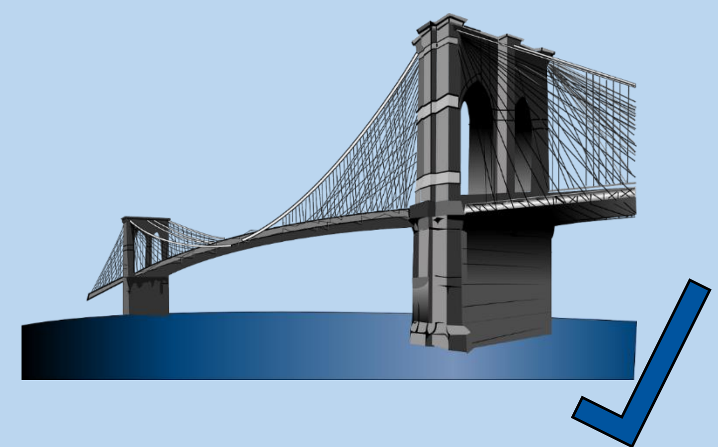
To Do	Doing	Done

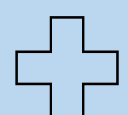
Cooperation.

Coordination.

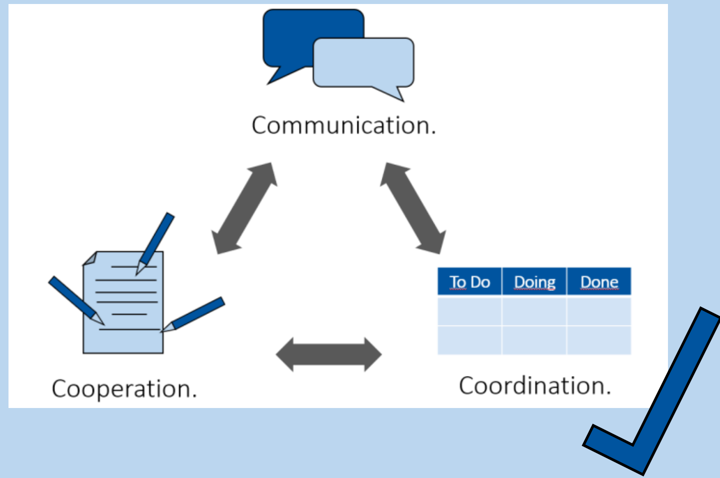


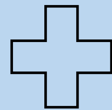


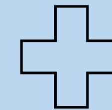


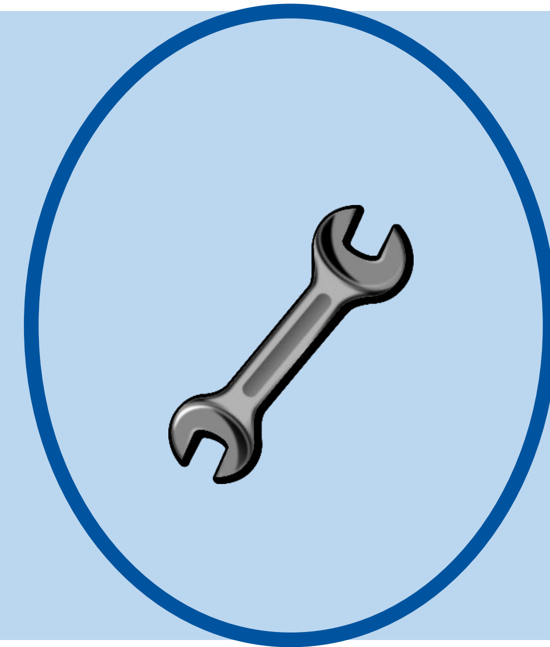


SWC Software Construction
RWTH AACHEN UNIVERSITY









SWC Software Construction
RWTH AACHEN UNIVERSITY

Demo

OctoUML

B. Vesin, R. Jolak, and M. R. V. Chaudron. OctoUML: An environment for exploratory and collaborative software design. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 7-10, 2017.

File Edit View

Class Diagram 1

Black

Zoom

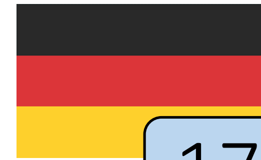
https://www.youtube.com/watch?v=xodDi8M25tM

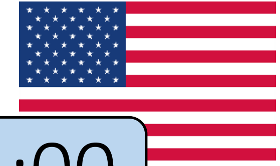
Such a fancy tool. But?

Distributed Collaboration?

Such a fancy tool. But?

Distributed Collaboration?





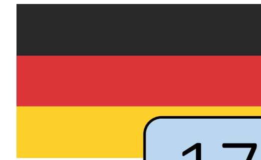
17:00

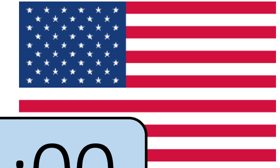
11:00

Asynchronous Collaboration?

Such a fancy tool. But?

Distributed Collaboration?



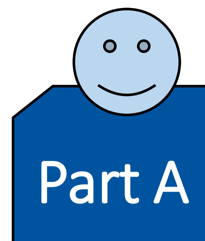


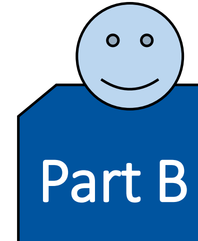
17:00

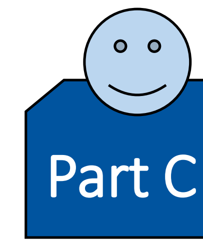
11:00

Asynchronous Collaboration?

Such a fancy tool. But?







Exclusive Collaboration?

CamelDesignCollaboration - s1_alice_2008_09_03_1/s1.posterboard - Eclipse SDK

File Edit Navigate Search Project Run Camel Window Help

Bird's eye view

PosterBoard Birdview
1

Navigator
.project
s1.posterboard
s1.whiteboard
Project navigator
2

Outline
umlExample1
Component_0
Actor_0
Actor_1
UseCase_0
Model outline
3

Team View
bob
charlie
dave
Team View
4

s1.posterboard
Posterboard area with adjusting palette
6

Package_0
Class_1
Class_2
+ class_1
+ class_2
[*]
[*]
+ class_2
[*]

Palette
Select
Marquee
Solid connection
Dashed connection
Markers
My Color Marker
Black Marker
Red marker
Blue marker
Shapes
Ellipse
Rectangle

Chat View
Chat view
5

alice :Added dependency between class 1 and class 2
bob :Did we need that?
charlie :we need that because otherwise functionality of the posterboard is limited

charlie
bob
Send

s1_alice_2008_09_03_1/s1.posterboard

CAMEL

M. Cataldo, C. Shelton, Y. Choi, Y.-Y. Huang, V. Ramesh, D. Saini, and L.-Y. Wang. CAMEL: A Tool for Collaborative Distributed Software Design. In *2009 Fourth IEEE International Conference on Global Software Engineering*, pages 83-92. IEEE, 2009.

Utopia – Collaborative UML Modelling Tool

Part A

Part B

Part C

Communication

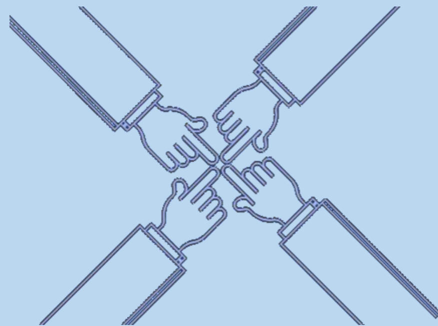
Coordination

Cooperation

Reality – Collaborative UML Modelling Tools

OctoUML

Cooperation



CAMEL

Communication

Coordination

Cooperation

Software Construction
RWTH AACHEN UNIVERSITY

Collaboration is COMPLEX
HAWAII 2015
SEA VOYAGE
CALIFORNIA - MALIBU BEACH
SHARKBAY
SURF SHOP
Summer
PARADISE
Holidays

Collaboration
IMPROVES
UML Modelling

Collaboration is COMPLEX

Collaboration
IMPROVES
UML Modelling

Collaboration is
COMPLEX

Collaboration Tools are
INSUFFICIENT

Collaboration IMPROVES UML Modelling

Collaboration is COMPLEX

Collaboration Tools are INSUFFICIENT

THANKS!

SWC Software Construction | RWTH AACHEN UNIVERSITY

One more question.

COLLABORATION – MUCH ADO ABOUT NOTHING?

COLLABORATION – MUCH ADO ABOUT NOTHING?

Ali Ariff (374675)

Aspects of Software Complexity

Full-scale Software Engineering Seminar 2018

SWC Software Construction
RWTH AACHEN UNIVERSITY

Motivation

- Software complexity definition is vague
- Known aspects:
  - Code
  - Architecture
- Goal: Discover more aspects
- Why: Grasp software complexity deeper
- Assess:
  - Microservice
  - Domain Driven Design

Current Definition of Software Complexity

- Complexity is the degree to which a system or component has a design or implementation that is difficult to understand and verify

_public_activity.html.haml — HuddleHR

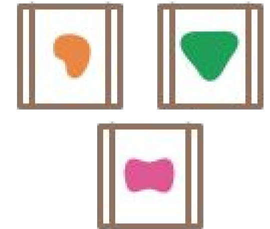
```
- if @activities.count > 0
  %ul.media-list
    - @activities.each do |activity|
      -# First, make sure to skip activities that are empty...
      - unless activity.parameters[:old_value] == nil && activity.parameters[:new_value] == ""
        %li.media
          - if activity.owner.present?
            - if activity.owner.profile
              %span.pull-left.media-object
                - activity.owner == current_user ? @owner = activity.owner : @owner = current_user
                = link_to image_tag(activity.owner.user_picture("thumb"), :class => "media-object img-
                polaroid", :alt => activity.owner.full_name), user_profile_path(activity.owner)
          - else
            %a.pull-left{:href => "#"}
              = image_tag "/assets/silhouette.gif", :class => "img-polaroid"
          .media-body
            %p.media-heading
              / Display who did the action
              - if activity.owner
                = activity.owner == current_user ? (link_to "You", user_profile_path(current_user)) : (
                link_to activity.owner.full_name, user_profile_path(activity.owner))
              - else
                HuddleHR made some automated changes to your account.
              / START PARTIAL
              / CREATE
              - if activity.key.split(".").last == "create"
                added
                / DESTROY
              - elsif activity.key.split(".").last == "destroy"
                deleted a
              - else
                / UPDATE
                - if activity.parameters[:old_value].present? && activity.parameters[:new_value].present?
                  updated
                  / Add new value for element
                - elsif activity.parameters[:old_value].blank? && activity.parameters[:new_value].
                present?
                  added
                  / Remove value for element
```


SWC Software Construction | RWTH AACHEN UNIVERSITY

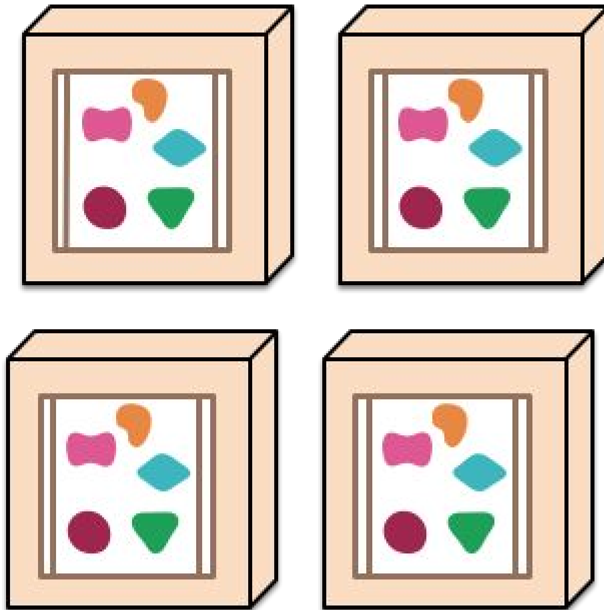
Microservice

A monolithic application puts all its functionality into a single process...

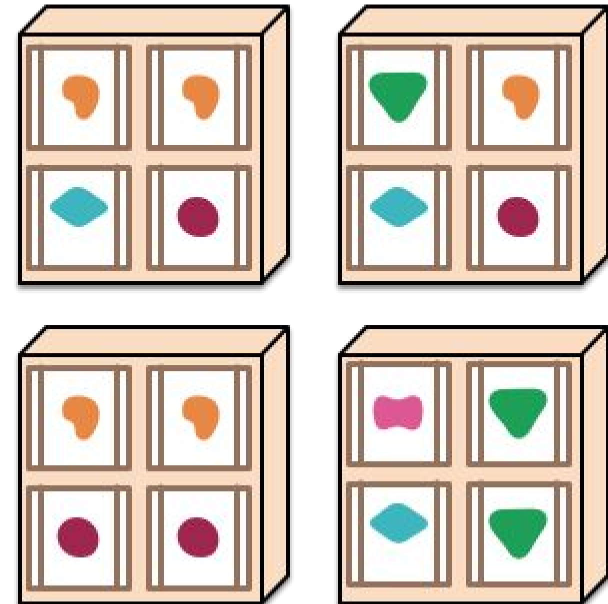
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers



... and scales by distributing these services across servers, replicating as needed.



https://martinfowler.com/articles/microservices.html

Findings in Microservice

- Architecture
- Operation
- Skill
- Security

Architecture & Operation

- Architecture
  - High Cohesion
  - Loose Coupling

- Operation
  - Test
  - Deploy
  - Monitor
  - Config

SWC Software Construction | RWTH AACHEN UNIVERSITY

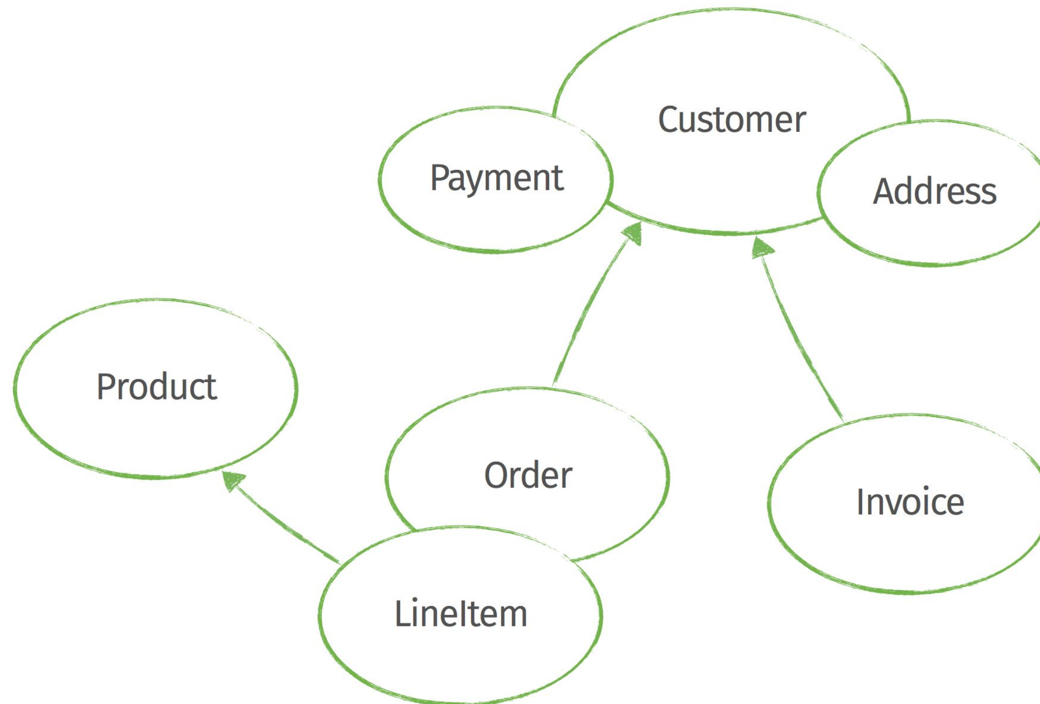
Skill & Security

- Skill
  - One service one team
  - Various skill-set required in a team

- Security
  - Hardening & securing distributed system
  - Maintain update various technology stack

Domain Driven Design (DDD)

- DDD is an approach to develop software for complex needs by deeply connecting the implementation to an evolving model of the core business concepts



Findings in DDD

- Domain
- Process
- Organizational
- Code

SWC Software Construction | RWTH AACHEN UNIVERSITY

Domain & Process

- Domain
  - Sphere of knowledge
  - Domain model
  - Bounded context

- Process
  - Isolation within the domain model
  - Stakeholder required to be collaborative

Organizational & Code

- Organizational
  - Communication & coordination
  - Remove dependency of each other

- Code
  - Code structure
  - Big codebase, more bugs & errors
  - Splitted to smaller parts

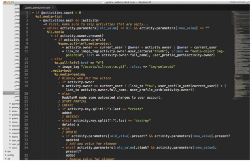
Conclusion & Future Work

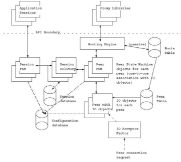
- Hidden aspects are discovered after assessment
- Not only code & architecture
- Increase understanding of overall software complexity
- Guidance to measure overall software complexity

Summary

Current Definition of Software Complexity

- Complexity is the degree to which a system or component has a design or implementation that is difficult to understand and verify





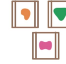
3

SWC Software Construction | RWTH AACHEN UNIVERSITY

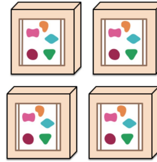
Microservice

A monolithic application puts all its functionality into a single process…

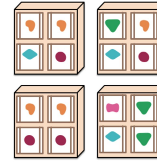
A microservices architecture puts each element of functionality into a separate service…



… and scales by replicating the monolith on multiple servers



… and scales by distributing these services across servers, replicating as needed.

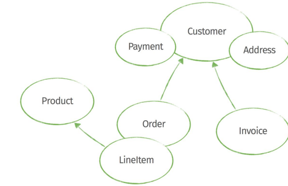


4 https://martinfowler.com/articles/microservices.html

SWC Software Construction | RWTH AACHEN UNIVERSITY

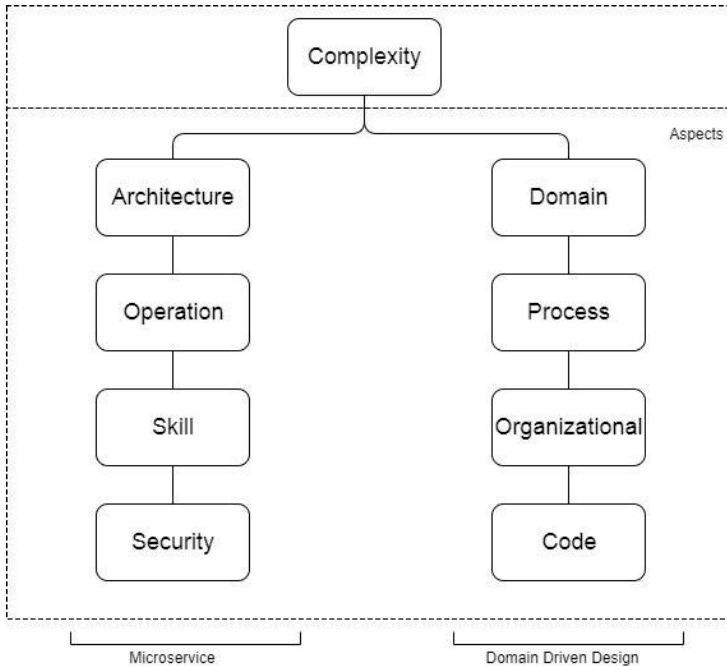
Domain Driven Design (DDD)

- DDD is an approach to develop software for complex needs by deeply connecting the implementation to an evolving model of the core business concepts



8

SWC Software Construction | RWTH AACHEN UNIVERSITY



Conclusion & Future Work

- Hidden aspects are discovered after assessment
- Not only code & architecture
- Increase understanding of overall software complexity
- Guidance to measure overall software complexity

12

SWC Software Construction | RWTH AACHEN UNIVERSITY

Radman Sheikh
radman.sheikh@rwth-aachen.de

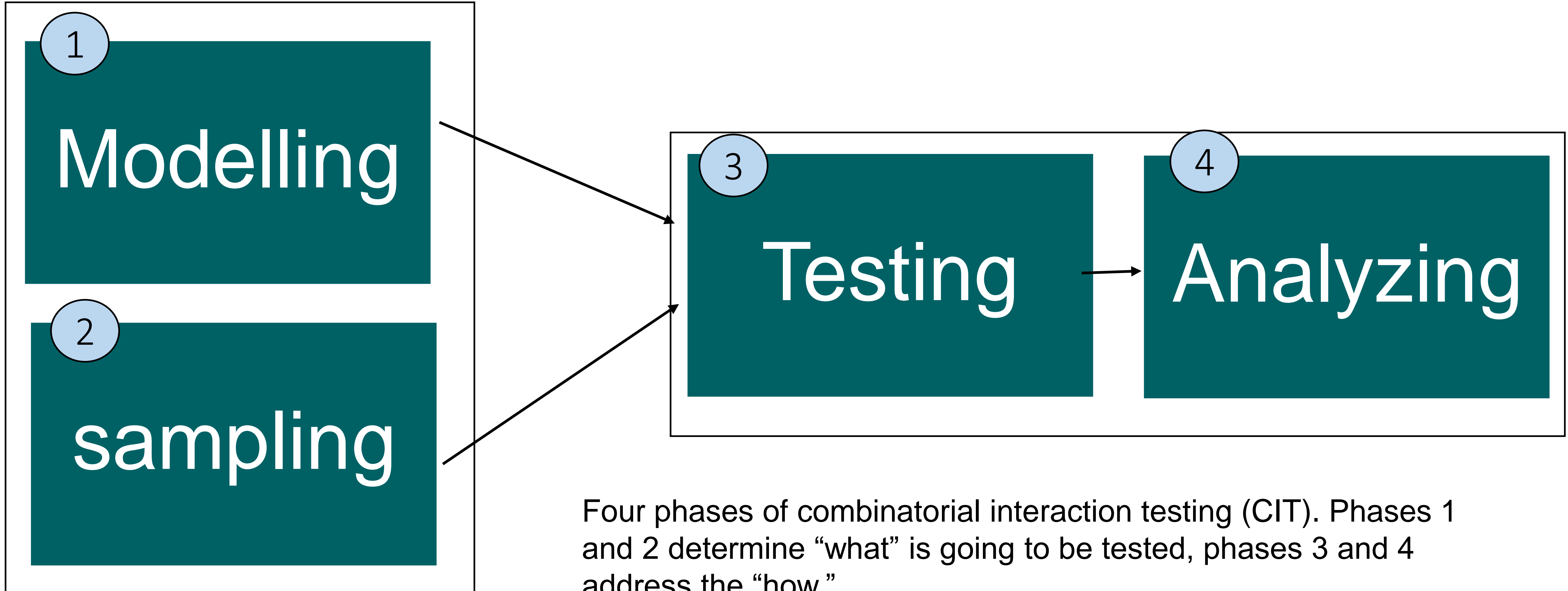
Applications of Combinatorial Testing

How to analyze and categorize ?

SWC Software Construction | RWTH AACHEN UNIVERSITY

Who have heard of combinatorial testing before?

What is combinatorial Testing?



SWC Software Construction | RWTH AACHEN UNIVERSITY

Some Questions

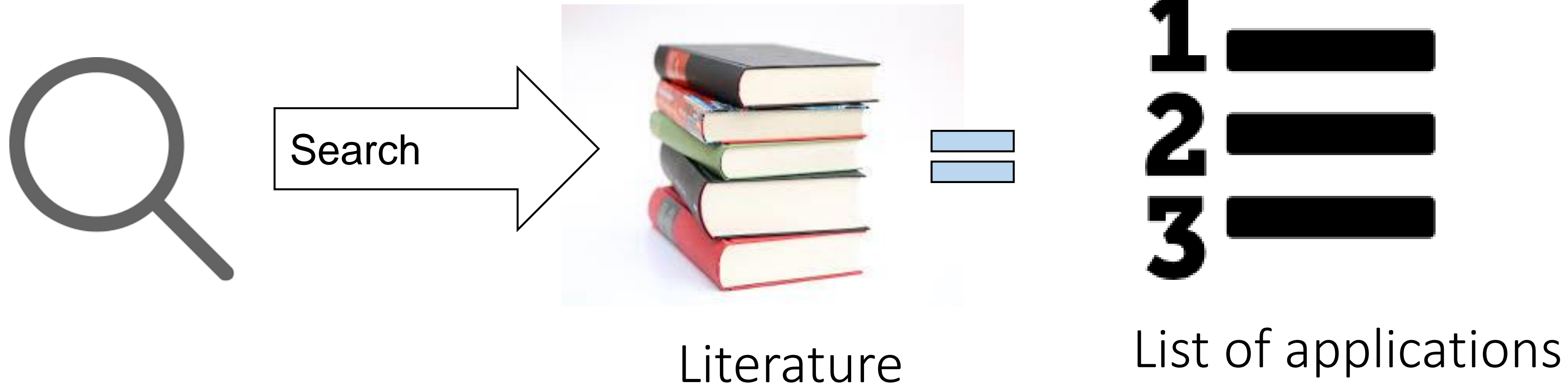
Before going into details lets ask ourselves some questions

- Can we apply CT to real world application.
- If we can apply CT on applications , can we categorize them
- What will be the result of categorization

Methodology of Finding CT Applications

- Significance of Combinatorial Testing

- Methodology adopted for finding CT application



Literature

List of applications

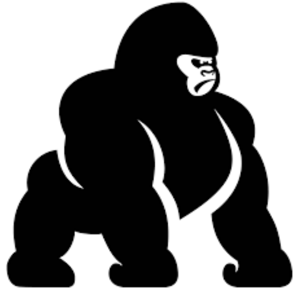
Why we did and needed Literature review?

Categorization of Application

- 46 literature sources considered
- 43 applications were discovered
- Focus of Search
- How Literature was found?

Categorization of Application

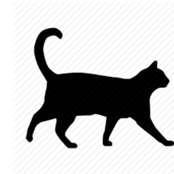
- The categories which were discovered are



Operating Systems

Web Applications

Mobile and space Applications

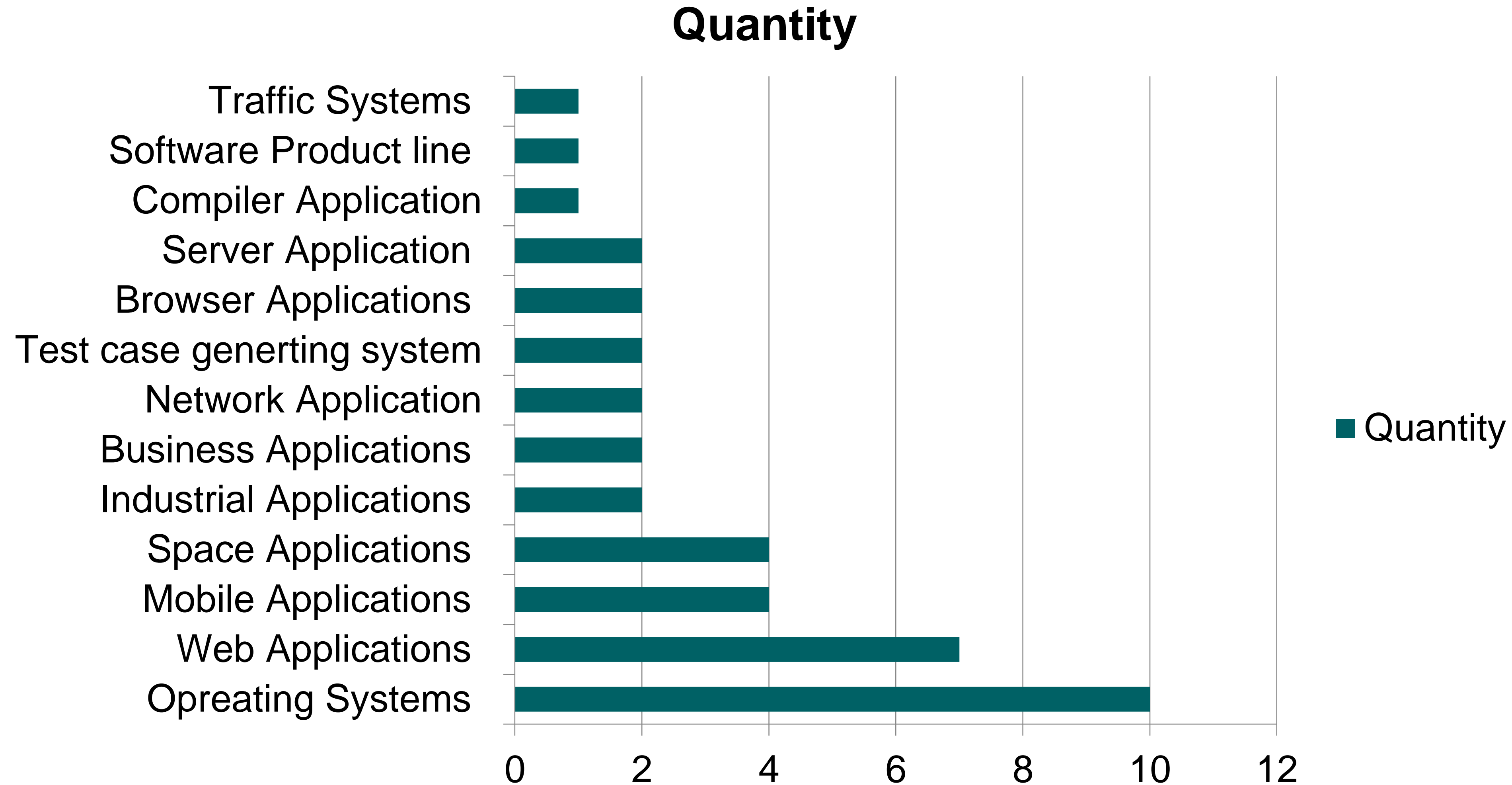


Business,
Network,
industrial,
server,
Browser and
Test case generating systems applications

Traffic
Inventory ,
Database systems,
Compiler and
software product line

SWC Software Construction | RWTH AACHEN UNIVERSITY

Categorization of Application



Factors Behind creating a category

Classification in literature

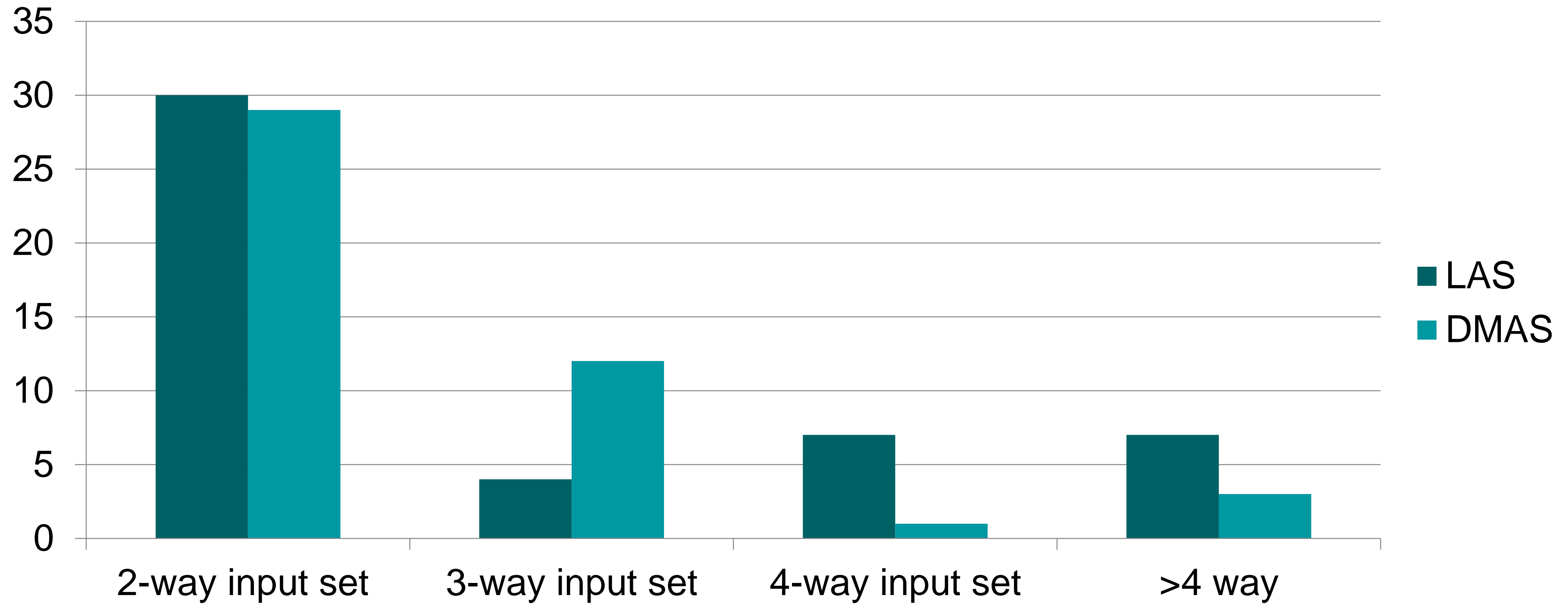
Application's Contribution

Keywords

Applications Background

What about the results ?

Combinatorial Testing on Business Applications



- Undetected bugs
- The variation of n-way input test set

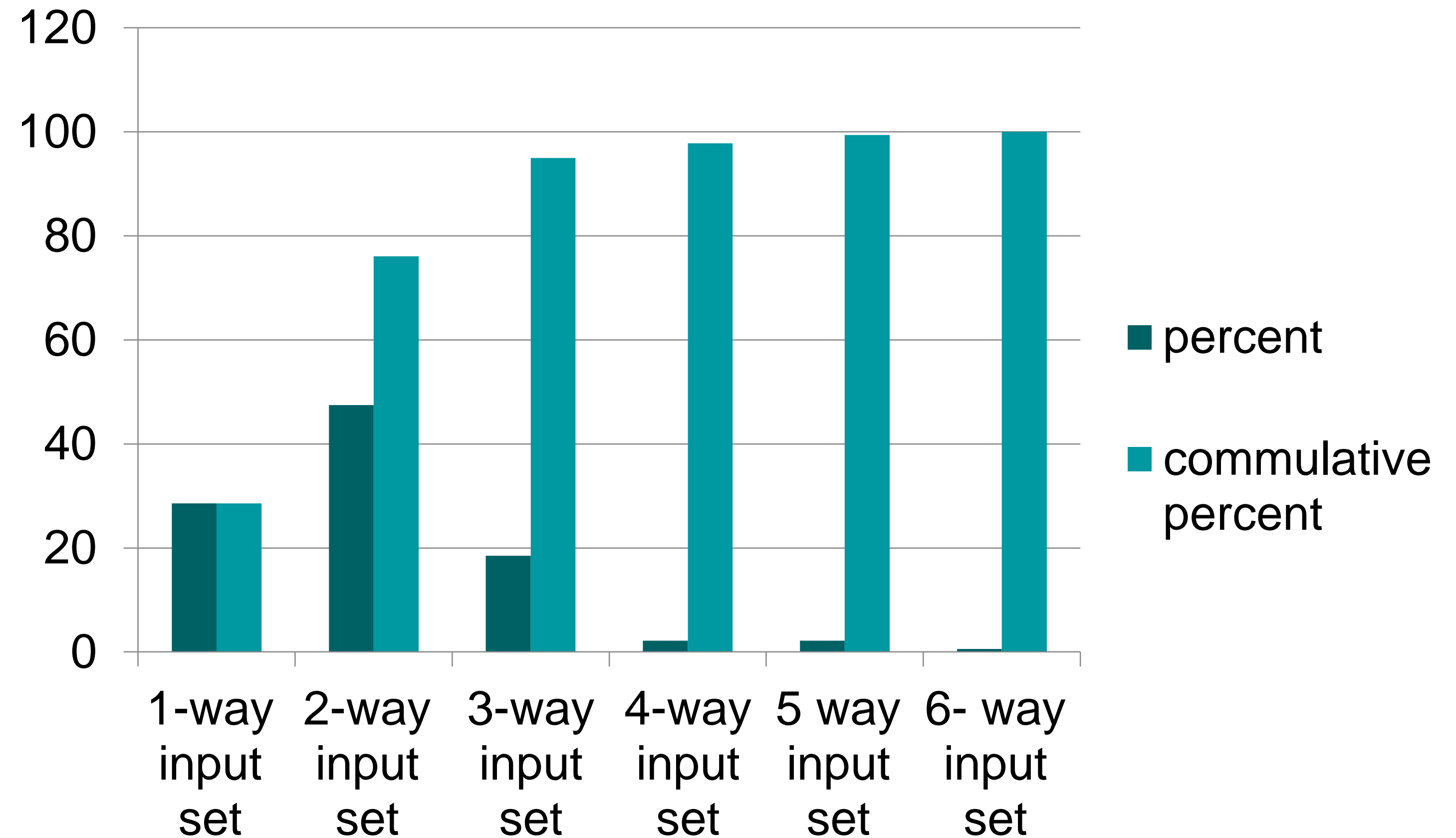
SWC Software Construction | RWTH AACHEN UNIVERSITY

Security Testing of Web Applications

- 5 web applications tested
- XSS exploitation was tested using an attacking vector
- BURP and ZAP penetration tools were used
- 58 tests were run on each tool
- The result was quite surprising

Bug Detection in Browser Applications

- 194 bugs reviewed and emulated
- 1-6 way input set were constructed
- The result is



Testing of Mobile Application

- Device versatility problem
- Optimal subset selection vs random selection
- Effectiveness measurement EC
- Conclusion

Testing ACTS through ACTS

- Testing a real life system
- What is ACTS
- Samples Tests and test results
- Basics for combinatorially testing a real life system
- Conclusion

System level testing of industrial application

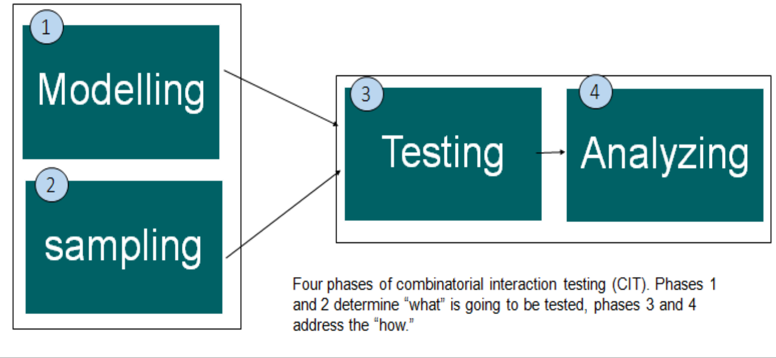
- Using Combinatorial testing to do system level testing on two industrial Applications.
- What is charm and EDM
- Building an abstract model
- Measuring quality through quality matrix
- Factor that should be kept in wind while Combinatorial Testing an industrial applications.

Detecting deadlocks in Grid networks

- What about combinatorial testing of network applications
- Question: can combinatorial methods help us find attacks on networks?
- SUT is "Simured" network simulator
- Main Goal :detect configurations that can produce deadlock
- Compare effectiveness of random vs. combinatorial inputs.
- To identify the deadlocks the researchers asked some questions
- Conclusion

Conclusion and future suggestions

Software Construction
RWTH AACHEN UNIVERSITY



Some Questions

Before going into details lets ask ourselves some questions

- Can we apply CT to real world application.
- If we can apply CT on applications, can we categorize them
- What will be the result of categorization

4

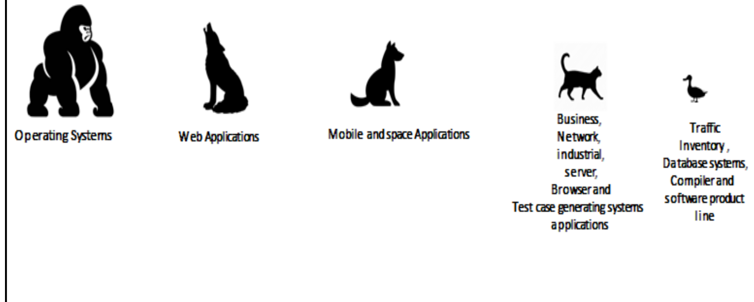
Methodology of Finding CT Applications

- Significance of Combinatorial Testing
- Methodology adopted for finding CT application

5

Categorization of Application

- The categories which were discovered are



Thank you!

Software Construction | RWTH AACHEN UNIVERSITY

Lukas Schade,
lukas.schade@rwth-aachen.de

Software Architecture in an Agile Environment

Software Construction
RWTH AACHEN UNIVERSITY

Agile Software Development

modern light fast

successful

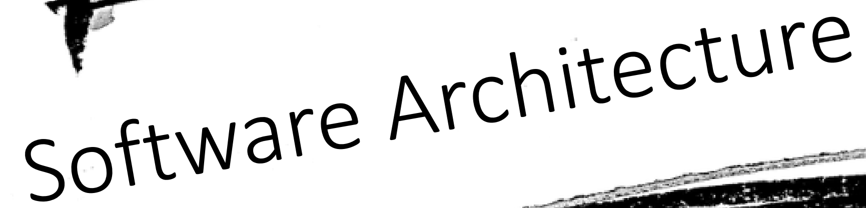
Lukas Schade, lukas.schade@rwth-aachen.de

Software Architecture

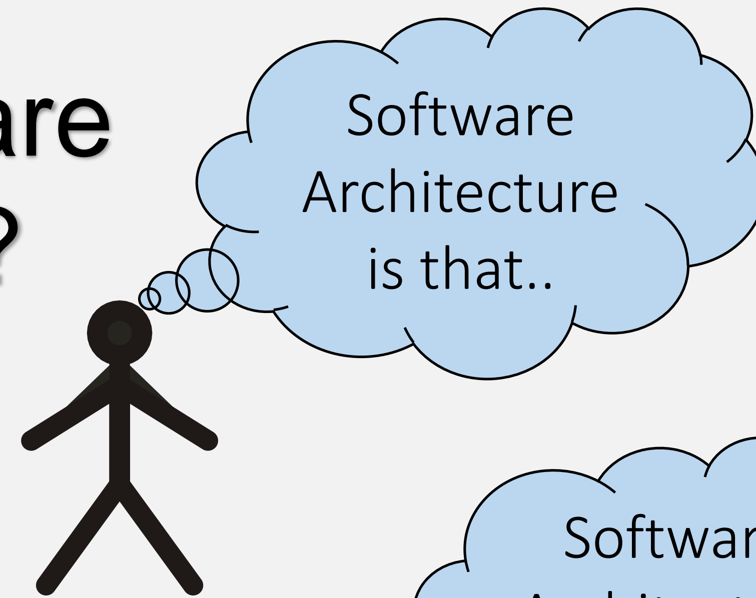
outdated heavy slow

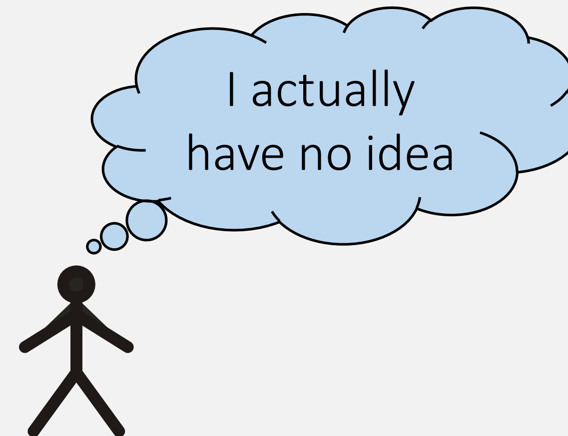
impractical

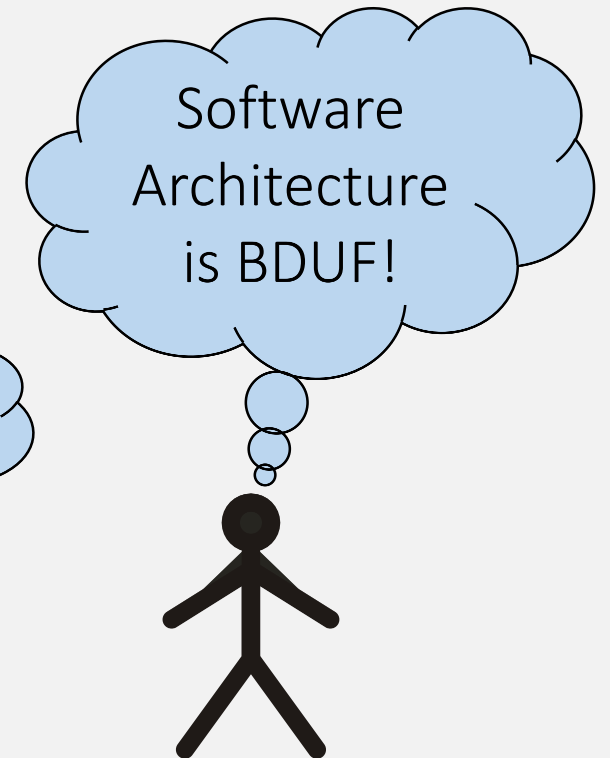
Lukas Schade, lukas.schade@rwth-aachen.de


Software Architecture
Agile Software Development

What is Software Architecture?







Software Architecture as…

the result of a design process

Lukas Schade, lukas.schade@rwth-aachen.de

Manifesto for Agile Software Development

…

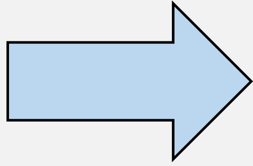
Working software over comprehensive documentation

…

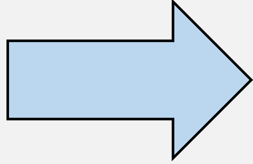
Responding to change over following a plan

http://agilemanifesto.org/

Lukas Schade, lukas.schade@rwth-aachen.de



Big design phase upfront is unwanted



Big comprehensive plan is unwanted

Lukas Schade, lukas.schade@rwth-aachen.de

Lukas Schade, lukas.schade@rwth-aachen.de

Architecture is developed iteratively

→ No big design phase

→ No big Plan

→ Allows responding to changes fast

→ Working software at an early stage

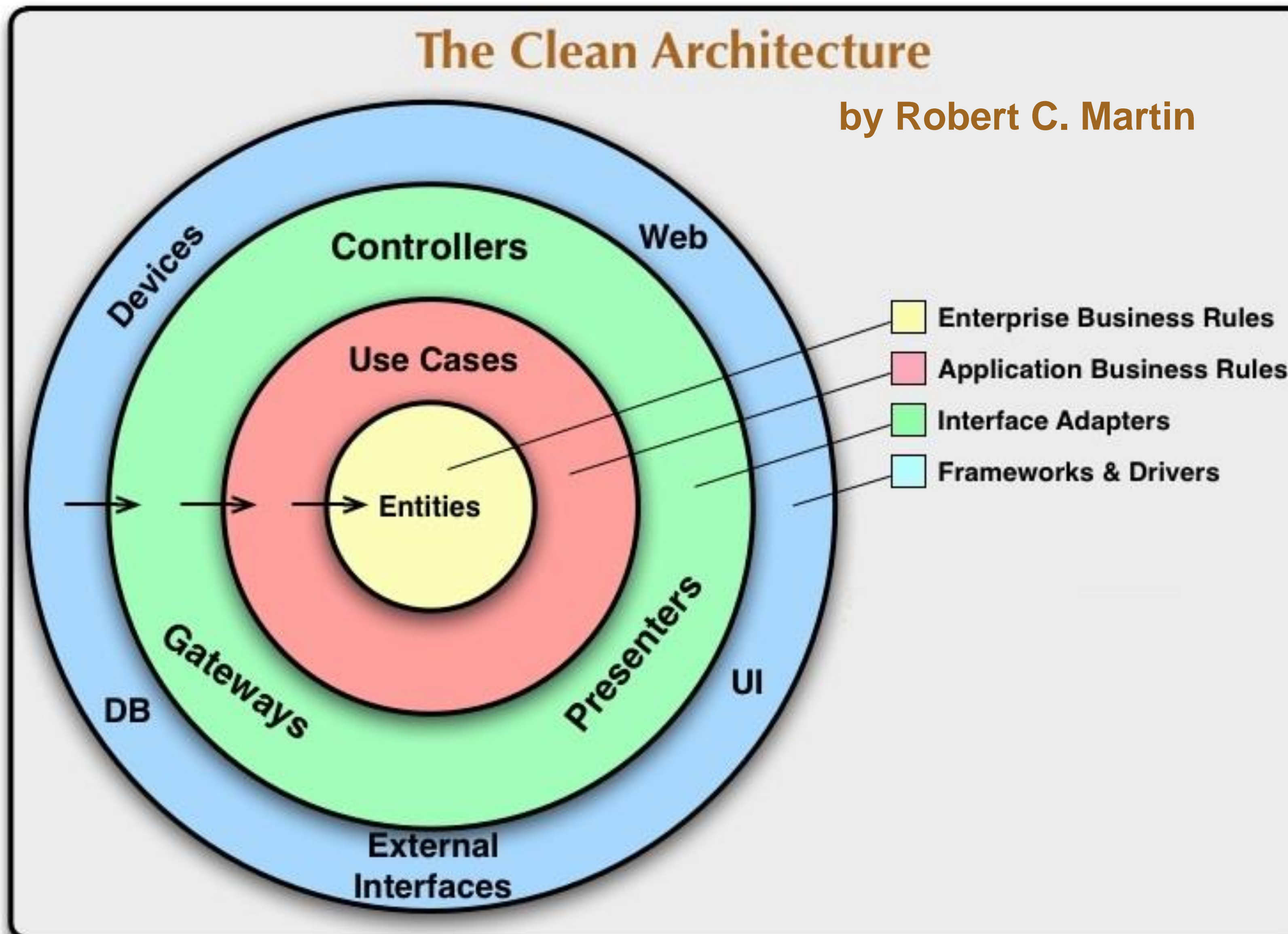
Lukas Schade, lukas.schade@rwth-aachen.de

Architecture needs to be:

adaptable

open for changes

Lukas Schade, lukas.schade@rwth-aachen.de



https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

Walking Skeleton

Contains key architectual decisions

Is not complete, but already “walks”

Lukas Schade, lukas.schade@rwth-aachen.de

Software Architecture

Agile Software Development

Architecture is developed iteratively

The Clean Architecture
by Robert C. Martin

Entities
Use Cases
Controllers
Gateways
Presenters
Devices
Web
DB
UI
External Interfaces

Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

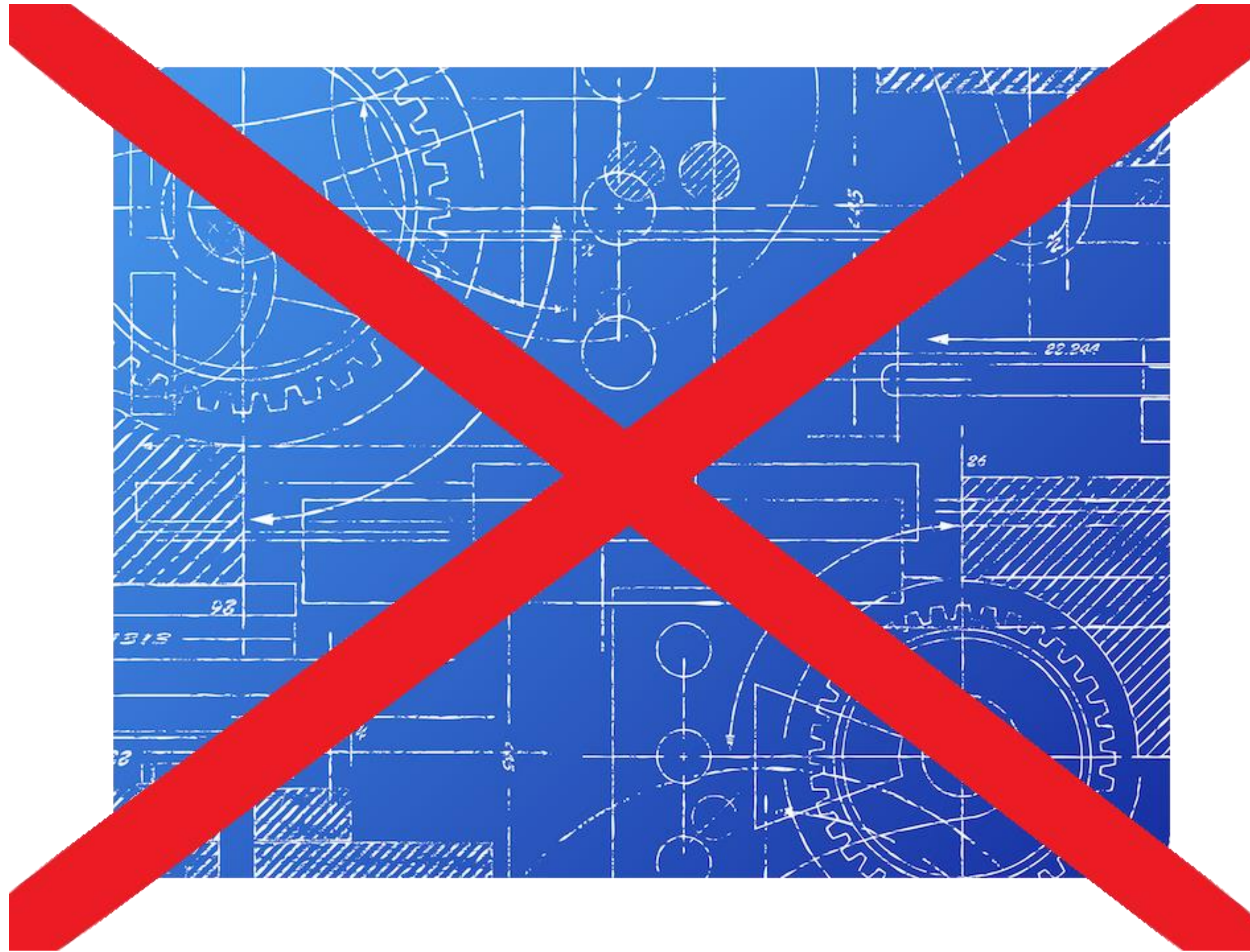
Walking Skeleton

Contains key architectual decisions

Is not complete, but already "walks"

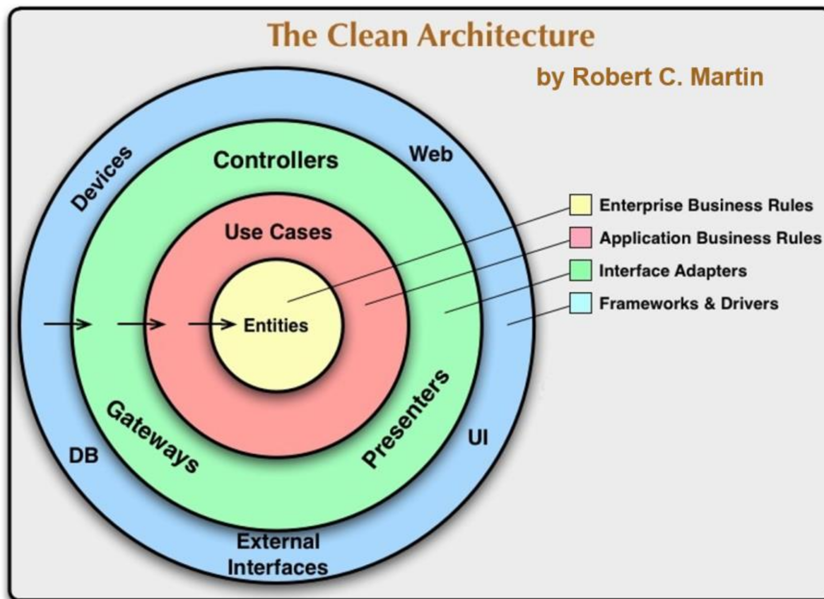
Software Construction RWTH AACHEN UNIVERSITY

Software Architecture
Agile Software Development



Architecture is developed iteratively

Lukas Schade, lukas.schade@rwth-aachen.de



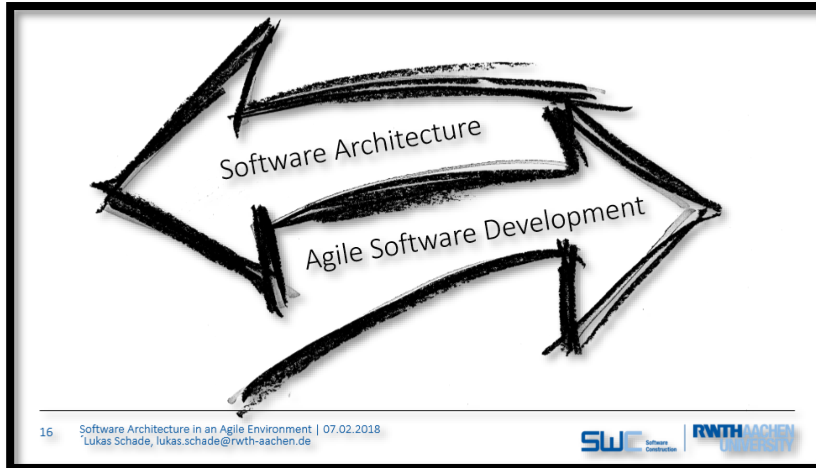
https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

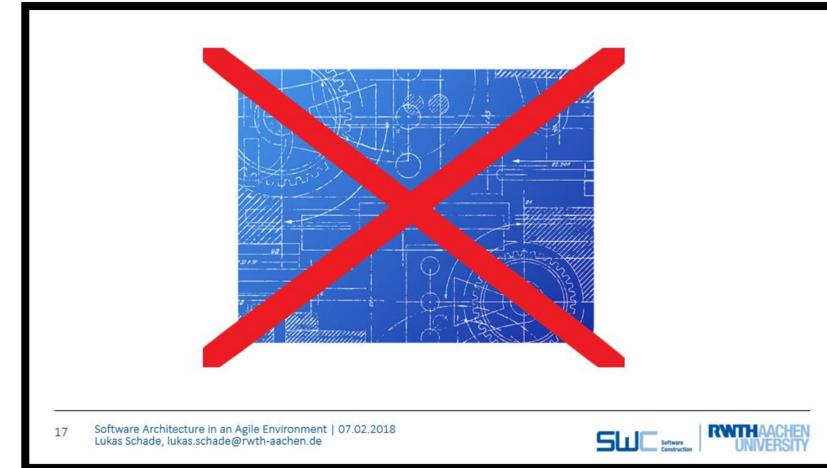
Walking Skeleton

Contains key architectual decisions

Is not complete, but already “walks”

Software Architecture in an Agile Environment





Architecture is developed iteratively

18 Software Architecture in an Agile Environment | 07.02.2018
Lukas Schade, lukas.schade@rwth-aachen.de

The Clean Architecture
by Robert C. Martin

Controllers, Use Cases, Entities, Gateways, Presenters, Devices, Web, DB, UI, External Interfaces

Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

Walking Skeleton

Contains key architectural decisions

Is not complete, but already "walks"

19 Software Architecture in an Agile Environment | 07.02.2018 Lukas Schade,
lukas.schade@rwth-aachen.de

Thank you!

Walking Skeleton

Contains key architectual decisions

Is not complete, but already “walks”

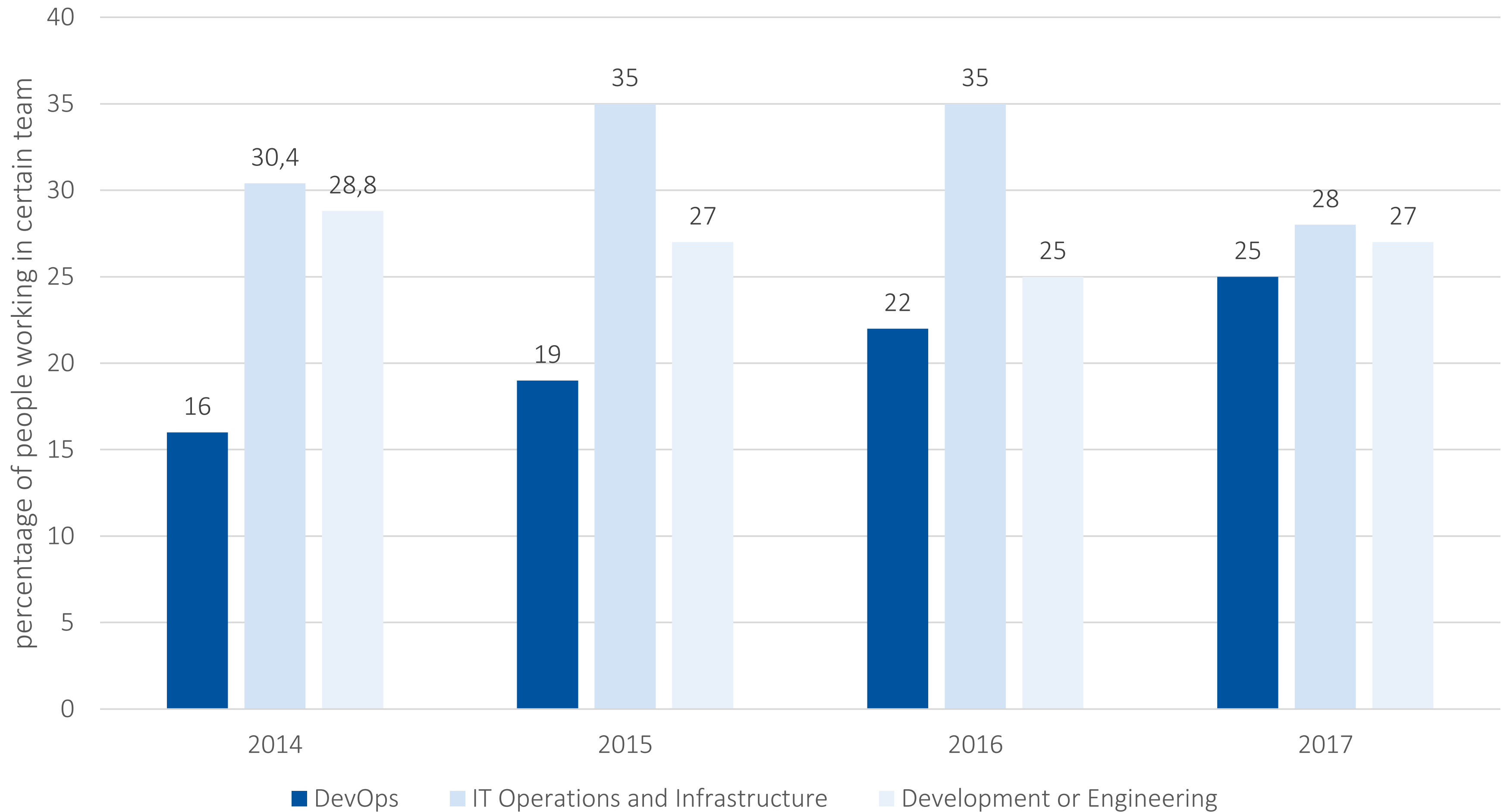
Lukas Schade, lukas.schade@rwth-aachen.de

Leon König
leon.koenig@rwth-aachen.de

Towards a Quality Model for DevOps

SWC Software Construction
RWTH AACHEN UNIVERSITY

Demographics in Industry



Leon König, leon.koenig@rwth-aachen.de
SWC Software Construction
RWTH AACHEN UNIVERSITY

DevOps – background and definition

- Clipped compound of **Dev**elopment and **Op**eration**s**
- Conflicts between development and operation teams
- Integration of existing movements

Agile Transformation

Lean Software Development

Continuous Deployment

Continuous Delivery

Leon König, leon.koenig@rwth-aachen.de

SWC Software Construction | RWTH AACHEN UNIVERSITY

A DevOps Quality Model

A selection from existing quality models

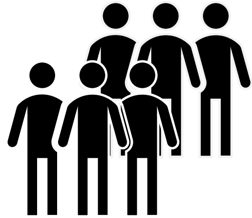
Leon König, leon.koenig@rwth-aachen.de

Goal-Question-Metric

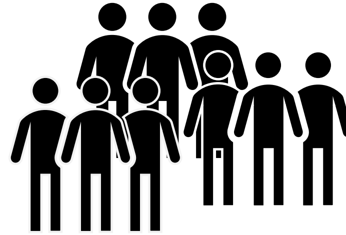
1. Formulate conceptual goals
2. Specify conceptual goals and assessment by questions
3. Derive metrics to answer questions in a quantitative way

SWC Software Construction | RWTH AACHEN UNIVERSITY

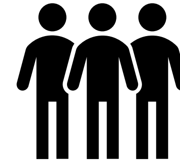
Conceptual goals



Dev team



DevOps team



Ops team

DevOps

Process

inter-team collaboration
intra-team collaboration
development process

Technical implementation

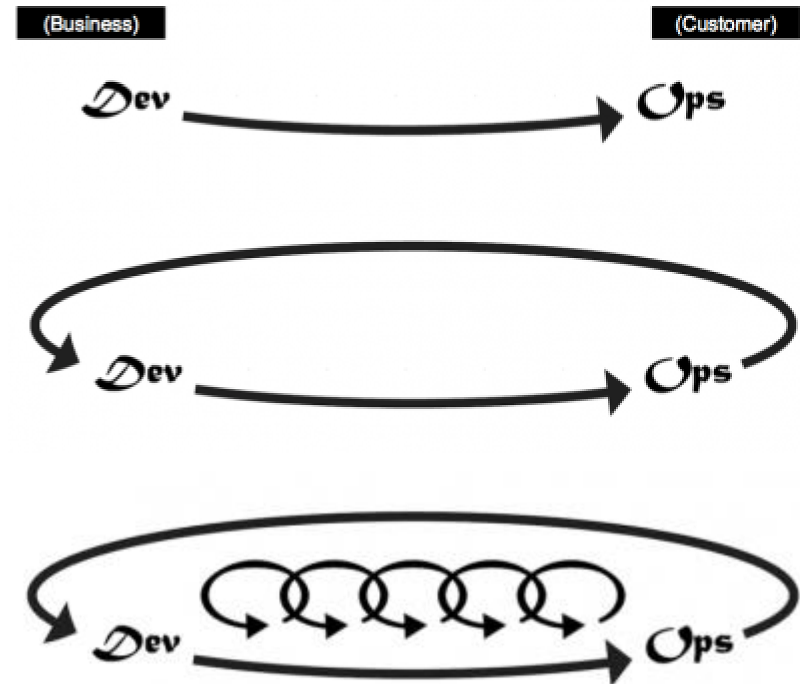
Deployment pipeline

Leon König, leon.koenig@rwth-aachen.de
SWC Software Construction | RWTH AACHEN UNIVERSITY

Process view

The three ways of DevOps by Kim et al.

- Principles of flow and system thinking
- Principles of feedback
- Principles of continual learning and experimentation



Source: https://itrevolution.com/the-three-ways-principles-underpinning-devops/ (25.01.2018)

Leon König, leon.koenig@rwth-aachen.de

Table 1: DevOps process quality

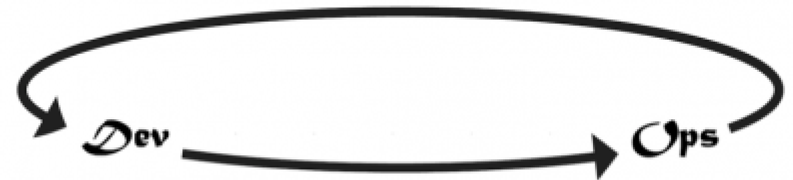
Question	Metrics	Source
What is the current performance of the development and deployment process regarding flow of value to the customer?	What is the number of releases per month?	[10]
	How many features are released per month?	[10]
	What are the costs per release?	[14]
	What is the average lead time of a feature?	[6]
	What is the average cycle time of a feature?	[18]
	What is the average change lead time?	[8, 16]
	What is the fastest possible feature lead time?	[10]
	What is the current revenue per user story?	[14]
	How old is the oldest done feature which is already developed and tested but not deployed to production yet?	[10]
	What is the mean time to repair a detected failure?	[16]
	What is the current change failure rate?	[16]
	What is the average badge size?	[6]
	Are teams organized around products or KPIs?	[12]
What is the current performance of the development and deployment process regarding flow of feedback?	What is the mean time to detect a failure?	[6]
	Does the development team have access to logs and stacktraces of the production systems for debug purpose?	[8]
	What is the number of incidents as a result of a feature release?	[18]
	Are metrics on system, availability and performance available?	[8]
	Does the monitoring system alert appropriate persons on failures?	[8]
	Does every person in the organization have access to visualized feedback and metrics of all systems?	[8, 4, 15]
	What is the average and maximum time between customer touch points?	[18]
What is the current performance of the development and deployment process regarding continual learning and experimentation?	Are code reviews performed to gain a uniform understanding?	[8]
	How much time is spent on rework after a feature release?	[18]
	What amount of time is used to store new knowledge?	[18]
	What amount of time does it take to retrieve knowledge acquired earlier?	[18]
	How much time does the team spend on reflection of their work after a project or sprint?	[18]
	How much time does the team spend on reflection of their working process?	[18, 15]
	Which percentage of deliberate introduced faults for experimentation does the team discover?	[18]
	Do people collaborate across team borders?	[4, 15]

Leon König, leon.koenig@rwth-aachen.de

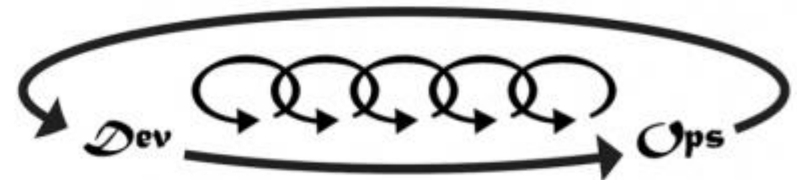
Process view

- Average lead time of a feature
- Oldest done feature
- Average badge size

- Mean time to detect
- Number of incidents as result of release
- Monitoring system features



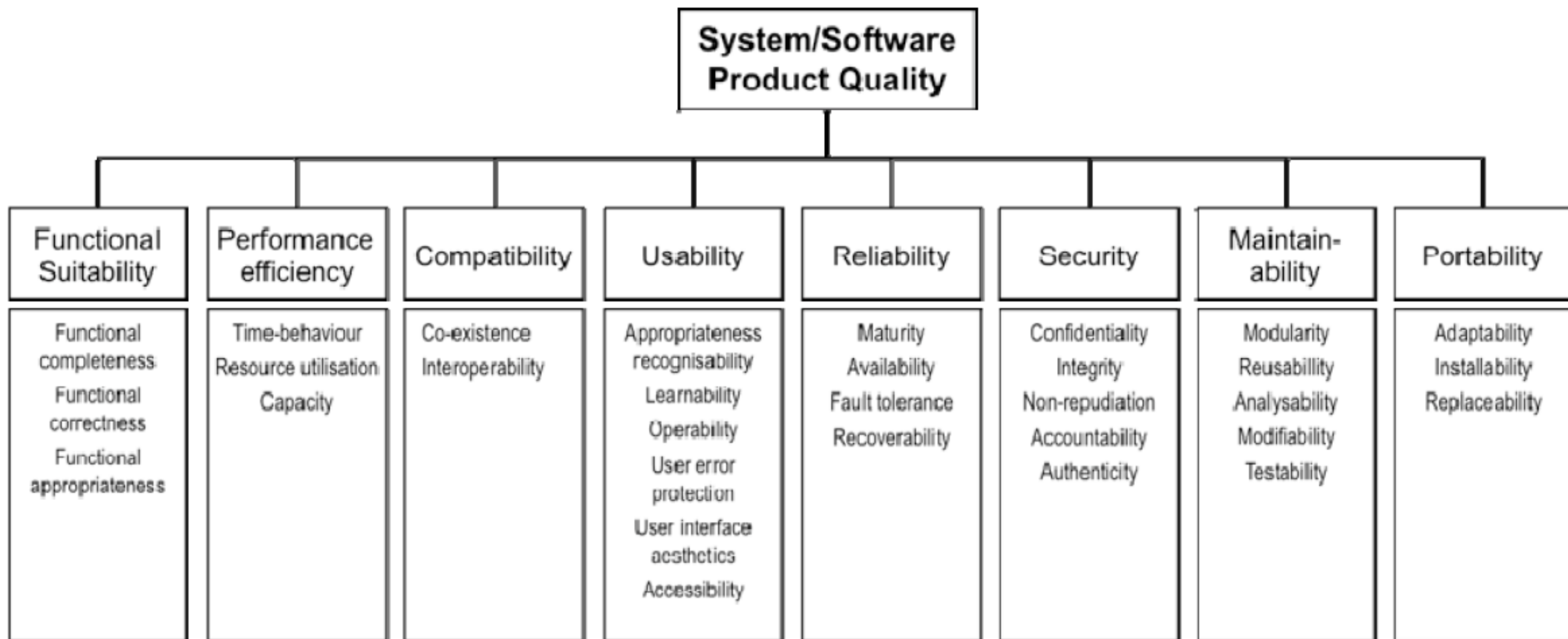
- Code reviews
- Time to store and retrieve knowledge
- Percentage of discovered faults



Leon König, leon.koenig@rwth-aachen.de

Pipeline is product of separate development process

ISO 25010 - Systems and software Quality Requirements and Evaluation (SQuaRE)



Source: ISO/IEC 25010:2011 – Figure 4 – Product quality model

Leon König, leon.koenig@rwth-aachen.de

Technical implementation view

• Fully automated deployments • Static code metrics • Zero-downtime deployments	**Functional Suitability**
• Equal deployments to different environments • Orchestrated deployments • Library / API management	**Compatibility / Maintainability**
• Change failure rate • Unit / Integration / Performance tests • Infrastructure as Code	**Reliability**

Leon König, leon.koenig@rwth-aachen.de

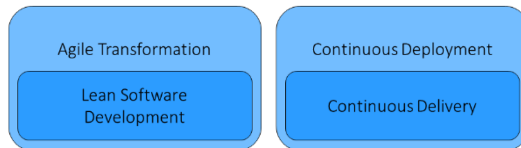
Table 2: Technical implementation quality

Characteristic	Metrics	Source
functional suitability	To what degree does the pipeline support a fully automated deployment?	[8]
	Are artifacts tagged and managed appropriate?	[11] [4, 15]
	Are static code metrics (CheckStyle, Sonar, FindBugs) generated?	[8] [11] [4] [15]
	Does the pipeline allow deployment roll-backs?	[11]
	Does the architecture support zero-downtime deployments?	[8] [3] [4] [15]
	Are deployments disconnected from releases?	[4, 15, 6]
	Does the pipeline support self-healing mechanisms on failures?	[12]
performance efficiency	What is the average lead time of a feature?	[6]
	What is the average cycle time of a feature?	[18]
	What is the fastest possible feature lead time?	[10]
	What is the mean time to repair a detected failure?	[16]
	How long does it take to deploy a new feature?	[8]
compatibility	Are artifacts deployed to any environment in an equal way?	[4, 15]
	Are deployments orchestrated?	[4] [15]
usability	Are testresults and metrics illustrated in a graphical way?	[8, 15]
reliability	What is the current change failure rate?	[16]
	Are database changes automated and versioned in an auditable way?	[3, 4]
	Are automated tests performed on build level in a continuous way?	[8]
	Are integration tests run in a continuous way?	[8, 4, 15]
	Are performance tests run in a continuous way?	[8, 4]
	Are deployments scripted and automated?	[8]
	Are system tests performed after every deployment automatically?	[8]
	Is the infrastructure provisioned by versioned *Infrastructure as Code*?	[8, 4]
security	Are changes to sourcecode or Infrastructure as Code auditable?	[8]
	Are pipeline runs traceable and accountable?	[8] [4, 15]
maintainability	Is the architecture component based or orchestrated?	[4, 15]
	Is an appropriate library and API management used?	[15]

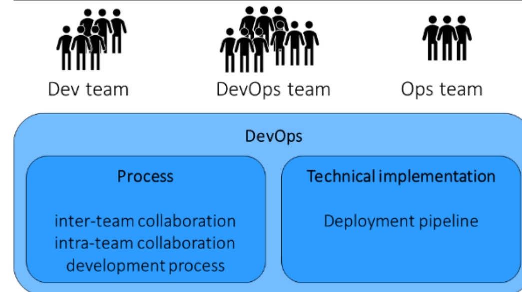
Leon König, leon.koenig@rwth-aachen.de

DevOps – background and definition

- Clipped compound of Development and Operations
- Conflicts between development and operation teams
- Integration of existing movements



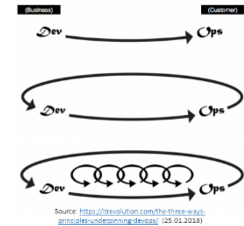
Conceptual goals



Process view

The three ways of DevOps by Kim et al.

- Principles of flow and system thinking
- Principles of feedback
- Principles of continual learning and experimentation

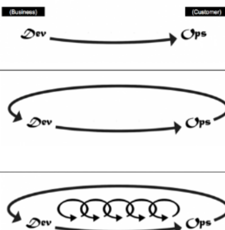


Process view

- Average lead time of a feature
- Oldest done feature
- Average badge size

- Mean time to detect
- Number of incidents as result of release
- Monitoring system features

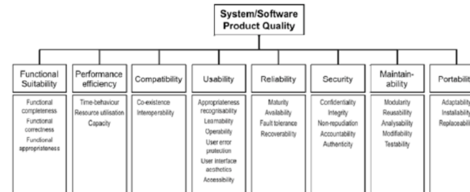
- Code reviews
- Time to store and retrieve knowledge
- Percentage of discovered faults



Technical implementation view

Pipeline is product of separate development process

ISO 25010 - Systems and software Quality Requirements and Evaluation (SQuaRE)



Technical implementation view

Metrics	Quality characteristic
• Fully automated deployments • Static code metrics • Zero-downtime deployments	Functional Suitability
• Equal deployments to different environments • Orchestrated deployments • Library / API management	Compatibility / Maintainability
• Change failure rate • Unit / Integration / Performance tests • Infrastructure as Code	Reliability

Conclusion / Outlook

- Result of selection from existing quality models
  - Much literature available on agile and continuous delivery / deployment
- Some gaps identified

- Verify quality model

Leon König, leon.koenig@rwth-aachen.de