2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW)

DOI 10.1109/EDOCW.2019.0016

Towards the Definition of Enterprise Architecture Debts

Simon Hacks*, Hendrik Höfert†, Johannes Salentin†, Yoon Chow Yeong‡, and Horst Lichter*
*Research Group Software Construction
RWTH Aachen University, Aachen, Germany
{hacks,lichter}@swc.rwth-aachen.de
† RWTH Aachen University, Aachen, Germany
{hendrik.hoefert,johannes.salentin}@rwth-aachen.de
‡ Universiti Teknologi Petronas
Perak Darul Ridzuan, Malaysia
yeongyoonchow@gmail.com

***Abstract*—In the software development industry, Technical Debt is regarded as a critical issue in terms of the negative consequences such as increased software development cost, low product quality, decreased maintainability, and slowed progress to the long-term success of developing software. However, despite the vast research contributions in Technical Debt management for software engineering, the idea of Technical Debt fails to provide a holistic consideration to include both IT and business aspects.**

Further, implementing an enterprise architecture (EA) project might not always be a success due to uncertainty and unavailability of resources. Therefore, we relate the consequences of EA implementation failure with a new metaphor – Enterprise Architecture Debt (EA Debt). We anticipate that the accumulation of EA Debt will negatively influence EA quality, and expose the business to risk.

***Index Terms*—Enterprise Architecture Management, Enterprise Architecture Debt (EA Debt), Term Definition**

1. Introduction

In the software development industry, Technical Debt is regarded as a critical issue in terms of the negative consequences such as increased software development cost, low product quality, decreased maintainability, and slowed progress to the long-term success of developing software [1], [2]. Technical Debt describes the delayed technical development activities for getting short-term payoffs such as a timely release of a specific software [3]. Seaman et al. [4] described Technical Debt as a situation in which software developers accept compromises in one dimension to meet an urgent demand in another dimension and eventually resulted in higher costs to restore the health of the system in future.

Furthermore, Technical Debt is explained as the effect of immature software artifacts, which requires extra effort on software maintenance in the future [5]. While the Technical Debt metaphor has further extended to include database design debt, which describes the immature database design decisions [6], the context of Technical Debt is still limited to the technological aspects.

There is extensive literature, which have been done on the concept of Technical Debt which can be sub-categorized into design debt [3], [7], (software) architectural debt [8], [9], code debt [10], documentation debt [11], etc. However, the concept of Technical Debt that particularly focuses on technical aspects demonstrates a lack of attention to attaining a holistic perspective to address the alignment between business and IT aspects. While enterprise architecture (EA) is gaining significant attention as a management instrument in business and IT [12].

Based on this observation, we see two possible threads. First, one can broaden the definition of Technical Debt to also cover business aspects. However, we believe that this will led to confusion if a term "Technical Debt" is not solely concerned with technical issues. Therefore, we decide for the second thread: introducing a term "Enterprise Architecture Debt" (EA Debt) that covers Technical Debt as well as business aspects. Since business and IT representatives potentially have different mindset and different goals [13], it is believed that EA Debt plays an important role to provide a common language for them. So far, we find two approaches, which try to conceptualize a similar idea. On the one hand, the Pragmatic EA Framework (PEAF) contains a concept of "Enterprise Debts" [14]. However, this is mainly related to the output of projects and misses an explicit definition of the term. On the other hand, there is a blog article[1] that uses the term "EA Debt" and highlight the need keeping track on "'partial' or 'messy' implementation of organizational changes". However, a clear definition is also missing there and we could not find any work that elaborates further on this.

For the purpose of introducing the metaphor, we formulate our research questions as follows:

(*RQ*) *What is an adequate definition of EA Debt?*

1. https://blogs.msdn.microsoft.com/nickmalik/2014/06/24/ea-debt/

In this paper, we argue for an EA perspective to expand the concept of Technical Debt to encompass business and IT aspects for a more holistic view. The core contribution of this study is a new proposed metaphor –EA Debt– that addresses the limitation of Technical Debt in providing a comprehensive enterprise architectural view. Mapping the debt concept to EA is also a remarkable initiative to deal with EA implementation tactfully.

The remainder of this paper is organized into the following parts: Section 2 introduces the concepts of Technical Debt and EA; those concepts are facilitated to come to a definition of EA Debt in Section 3, while we present three examples for EA Debt in Section 4.2; lastly, we express implications and future works in Section 5, and conclude with Section 6.

2. Key Concepts and Related Work

Before we deduce the term "EA Debt", we elaborate more on the terms Technical Debt (see Section 2.1) as well as EA and its quality issues (see Section 2.2).

2.1. Technical Debt

The metaphor *Technical Debt* was first introduced by Cunningham [2] and mentions what we today would call "refactoring". This first idea of not-quite-right code which we postpone making it right, is expanded by inter alia Kruchten et al. and Seaman et al. [15], [16] to display also other kinds of debts or ills of software development, such as test debt, people debt, architectural debt, requirement debt, documentation debt, or an encompassing software debt.

According to Kruchten et al. [15] Technical Debt refers to invisible elements, because visible elements for improving, like new features for evolution or repairing defects for maintainability issues, should not be considered as debt. Technical Debt should rather serve as a retrospect reflecting change of the environment, rapid success, or technological advancements as a possible cause for debt. However, "the debt might actually be a good investment, but it's imperative to remain aware of this debt and the increased friction it will impose on the development team" [15]. Hence, tools are required to increase the awareness to identify debt and its causes, and to manage debt-related tasks. Finally, the debt should not be treated in isolation from the visible elements of evolving and maintaining. Consequently, debt is "the invisible result of past decisions about software that negatively affect its future" [15].

Tufano et al. [17] encountered the same phenomenon and point out that most code smells are introduced at their creation. Furthermore, the code often gets smellier due to new artifacts build on top of suboptimal implementations. Even refactoring is often done wrong as it introduces further bad smells, which highlights the need for techniques and tools to support such processes [17].

McConnell [18], for example, tried to categorize different types of Technical Debt. He stresses that with this metaphor business and technical viewpoints can be emphasized, so that communication regarding specific problems can be enhanced. Technical Debt is used as a uniform communication tool, that allows us to measure and keep track of debt which eventually should help find a suitable solution to the upcoming challenges. In this case it should also reflect the different viewpoints, including the stakeholder's perspective, in order to allow an effective collaboration [18], [19].

Fowler even came up with a categorization of Technical Debt with his "Technical Debt Quadrant" to identify different types of it [20]. He distinguishes Technical Debt into reckless / prudent and deliberate / inadvertent debt. This reflects the different scenarios where debt is taken and hints at debt being taken unconsciously or negligent sometimes.

For him, the metaphor's primary task is to help "thinking about how to deal with design problems, and how to communicate that thinking" [20]. Nevertheless, it is used as a tool, which can reveal possible drawbacks of a current design decisions: his differentiation makes everyone aware that a certain decision could cause debt. This theoretical concept should then help to find a reasonable solution [21].

Further, Technical Debt tries to help to decide how to invest scarce resources: "Like financial debt, sometimes Technical Debt can be necessary" [22]. The value and present value play a role, including the difference between the actual state and an supposed ideal state as well as the time-to-impact. This involves a differentiation between "structural issues (the potential Technical Debt) and the effect it has on actual development (the effective Technical Debt)" [23], which could also be called problems and risks.

Overall debt has to be seen in its environment and it has to be determined if the debt was strategic or unintentional. Ultimately, this Technical Debt can be considered as an external software attribute, which needs to be quantified [22], [24]. As a consequence, one has to measure all criteria to estimate the debt and make a proper decision based on that information. It has been shown that reasonable decisions can be made more easily, if corresponding information (debt) is taken into account. Additionally, delaying a supposed "right" implementation has a significant impact on the cost of the project, so that an appropriate management of the debt concept is useful [25], [26], [27].

Technical Debt has shown its benefits in estimating deficits of software construction, providing a tool for decision making and increasing the awareness of said deficits [15], [16], [18], [28]. However, metrics that are commonly used in Software Engineering (cf. TD estimation [29]) should not only be used on single systems and software, but rather be generalized to be applicable and helpful for the entire enterprise and every scenario [8], [30], [31]. This encompasses then the different layers of EA, including business and IT with its systems and strategies in particular. Thus, it gets possible to estimate the consequences of EA implementation failures so that with the new metaphor of EA Debt improvements and measurement of the quality get possible. Furthermore, strategic decisions about future development based on urgency and dependency of debt can

be facilitated.

2.2. Enterprise Architecture

Following, we focus on *EA* and its quality issues. In accordance to ISO/IEC 25010 quality "is the degree to which a product or system can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific contexts of use" [32].

As described by Saint-Louis et al. [33] the definitions for EA diverge significantly, already hinting at differences in quality issues. Kappelman et al. [34] pointed out that "The 'enterprise' portion of EA is understood by some as a synonym to 'enterprise systems', yet by others as equivalent to 'business' or 'organization'. [...] Even less uniform is the understanding of the meaning of 'architecture'. The most common understanding of the term is a collection of artifacts (models, descriptions, etc.) that define the standards of how the enterprise should function or provide an as-is model of the enterprise" [34]. However, EA helps to face "ongoing and disruptive change" [33] by attempting to align IT and business strategy.

Despite the diverse perspectives and definitions, some articles focus on quality issues regarding EA. One could for example derive qualities that encompass IT system qualities, business qualities, and IT governance qualities [35]. Henderson and Venkatraman's Strategic Alignment Model (SAM) identifies Business Strategy, Business Structure, IT Strategy and IT Structure as four key domains of strategic choice, the so called "alignment domains" [36]. Based on this information an artifact-based framework for business-IT misalignment symptom detection was created in the article of the same name [37]. This framework also includes a first suggestion of catalogues for misalignment symptoms, EA artifacts, and EA analysis methods, so that also rather invisible or underestimated elements are considered. Moreover, a link between those categories is established, in order to know which misalignment symptoms affect which artifacts and how this can be analyzed.

Addicks and Appelrath [30] searched for key figures and their metrics in order to unitize the quality assessment of an application landscape. This approach can be applied to the EA as a whole, because business processes for example influence the application's quality. They stated that "all key figures must at least fit one of the following three conditions: (a) it must be used for indications of applications and be based on the applications attributes, (b) it must be an indicator of an application and its value is determined by attributes and relations from other EA artifacts (the applications enterprise context), and (c) it must indicate a landscapes quality and therefore use all attributes of applications and their enterprise context" [30].

Since many aspects influence the system's properties a unified meta model is difficult to create. However, Saat et al. [35] show that even different enterprise orientations with divergent focuses prefer certain qualities over others. In general, they show that the striven qualities differ across enterprises, nevertheless, there are some general qualities that are desired in most situations. Thus, a specialized prioritization and adaptation is needed for the EA and its IT/business alignment.

More general approaches are followed by for example artifact based viewpoints of Winter and Fischer [38] or the TOGAF Standard [39]. They refer to domains like Business / Process Architecture, Software / Data Architecture, Application / Integration Architecture and Technology / Infrastructure Architecture.

Ylimki goes even further and defines twelve critical success factors for EA. These factors obviously influence EA and its quality, although they are different to previously known aspects. Here high-quality EA is described with: "high-quality EA conforms to the agreed and fully understood business requirements, fits for its purpose (e.g. a more efficient IT decision making), and satisfies the key stakeholder groups (the top management, IT management, architects, IT developers, and so forth) expectations in a cost-effective way understanding both their current needs and future requirements" [40].

Moreover, there is an EA Model Quality Framework (EAQF) proposed by Timm et al. [41], which builds upon six principles to assess the quality of EA models, namely the principles of validity, relevance, clarity, economic efficiency, systematic model construction and comparison. Each of the principles is augmented with quality attributes that can be assessed in order to determine the quality addressing the EA Model's purpose, a specific view of it and the overall EA Model [41].

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

A further facet of EA quality is related to its complexity. Schmidt et al. [42] propose to calculate the entropy on certain parts of the EA. Therefore, they interpret the EA as a graph comprised of EA "things" as nodes and their relations as edges. Then, they calculate on the subset of the EA (e.g., the application layer) the entropy and compare it to the desired state. Last, they demonstrate their work on a case study of a insurance company transforming their IT landscape.

3. Defining EA Debt

As seen in Section 2.1, the definitions of Technical Debt are mostly descriptive, naming properties and different types of debt, rather than explicitly defining the term. Nevertheless, Technical Debt is considered as a tool pointing at possible risks, measuring and tracking deficits, and aiming at supporting the process of finding suitable solutions. The focus mainly lies on increasing awareness of also invisible or structural elements on a technical level and giving a uniform

basis for discussions and communication. Further, a holistic vision is often missing, but needed to steer Technical Debts in the broader context of the whole organization.

To outline the findings regarding EA and its quality issues from Section 2.2, we conclude that there are different definitions of EA around. Although it is more often described and its quality aspects are mentioned with respect to a specific viewpoint and enterprise orientation, the descriptions diverge. Hence, only a common understanding of EA and a basis for communication and discussion is established.

Therefore, the requirements differ for each enterprise, making a uniform approach according to EA models and quality issues for a certain level of detail impossible. Consequently, our definition of EA Debt determines a technique providing some crucial factors in order to estimate an EA's quality for its specific purpose on a high abstraction level and increase the awareness for possible sub-optimal aspects that may cause severe drawbacks in the future. The metaphor of debt should serve as a common basis for communication and discussion, as well as pointing out differences in as-is situations and proposed ideal to-be situations [34]. The need for such a basis was observed before as Niu et al. [43] identified four possible situations regarding communication:

1) Consensus where stakeholders use terminology and concepts in the same way.
2) Correspondence where they use different terminology for the same concepts.
3) Conflict where they use the same terminology for different concepts.
4) Contrast where stakeholders differ in terminology and concepts.

Furthermore, flexibility and robustness are identified as important factors in the dynamically changing and evolving environment of the enterprise and its structures. Hence, it can be used as a powerful tool, like Technical Debt, and help to manage a complex enterprise, because it allows to get a holistic overview and keep track of existing debt. All in all this should fulfill the purpose and function of EA according to Kappelman et al. [34].

In order to achieve a scenario-unaware definition of EA Debt, we have to state what we refer to when we are talking about EA. Saint-Louis et al. [33] conducted a SLR (Systematic Literature Review) and found multiple definitions of EA. Therefore, we regard EA as a set of artifacts, which are aggregated.

In the TOGAF Standard version 9.2 Business Architecture, Data Architecture, Application Architecture and Technology Architecture are identified as the four architecture domains [39]. Those roughly match the different layers of Winter and Fischer [38], although they named Process Architecture in particular.

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

1) EA layers
   - 1.1. Business Architecture
   - 1.2. Data Architecture
   - 1.3. Application Architecture
   - 1.4. Technical Architecture
2) Other influencing factors
   - 2.1. Stakeholders
   - 2.2. Guidelines and Standards

Considering EA as a set of artifacts, it can contain more than the mentioned artifacts, but it is easy to add and remove artifacts later, so the metaphor of EA Debt can be adapted to individual situations.

Winter and Fischer [38] point out that "Most of the artifacts [...] in EA can be represented as aggregation hierarchies". Taking this aggregation hierarchies and the definition of Technical Debt by Cunningham [2] into account, EA Debt in general can be understood as an aggregation of taking debts in each layer. But, just adding up each artifact would not give a concrete overview of the current situation, it could even whitewash huge issues in the whole EA. Therefore, every artifact, and also every part an artifact consists of, should be weighted. Obviously, there is no uniform weighting function, because it depends on the concrete EA and also the organization. On top of that, we need to take into consideration that the interrelations along artifacts can cause debt, too. An artifact might be optimized and work perfectly, but if there is a huge overhead caused by interfaces respectively multiple lines of reporting, the EA might not work perfectly.

Distilling the aforementioned properties of Technical Debt and EA, we come to following definition:

> **Definition** (EA Debt)**.** *Enterprise Architecture Debt is a* **metric that depicts the deviation** *of the* **currently present state** *of an enterprise from a* **hypothetical ideal state**.

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

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

Here a focus on mainly invisible elements can be helpful, because these factors may not even be recognized or their impacts are underestimated, like a too complex EA or wrong design decisions. By increasing the awareness for such issues the overall inadvertent debt can be reduced. Assuming a prudent management, this would lead to debt mostly being taken consciously and planned strategically. This means that

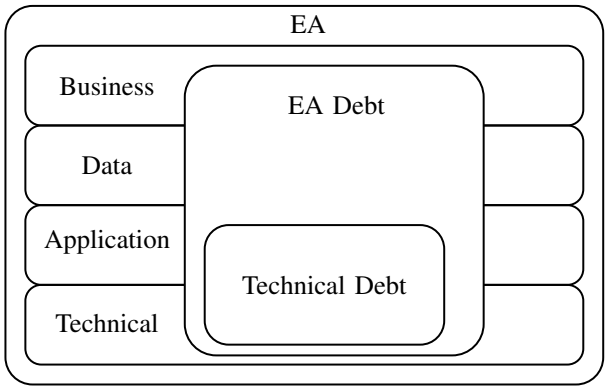

Figure 1. Placing of Technical Debt within the concept of EA Debt

possible repercussions are weighed and less trade-offs are accepted, which unnoticeably impair the enterprise.

Hence, the identified differences between the current state and the supposed ideal situation can be managed, such that the sometimes necessary debt, namely deliberate prudent debt, holds the largest share. Further, we can relate Technical Debt to EA Debt as a sub-domain of it (cf. Figure 1) that is related to the software and technology aspects. However, there are also issues related to EA Debts on software and technology aspects that are still not covered by Technical Debt. Those issues can be inter alia their relations among each other or the adherence to overarching business goals.

4. Demonstrating EA Debt

Applied standards and guidelines are an indication for reliable artifacts and therefore lower EA Debt. The EAQF [41] assesses the quality of an EA model with six principles. These principles can be used to construct an ideal situation and detect the differences to the current as-is situation. In this way also rather invisible or unconsidered elements can be found, which then form a good entry point for EA Debt estimations. Yet again, the awareness of invisible or hardly noticeable elements has to be increased, because they in particular entail high risk of further debt in the future.

Őri [37] proposed an artifact-based framework that aims at detecting misalignment in an undesired state of the enterprise, which can be used to detect also EA Debt. Based on the strategic alignment perspectives Strategy Execution, Technology Transformation, Competitive Potential, and Service Level the framework decomposes those perspectives into corresponding perspective components, namely the "alignment matches". Then, they are connected to typical misalignment symptoms, using a misalignment symptom catalogue as a reference. After that, relevant containing artifacts are identified, again using an artifact catalogue. Finally, suitable EA analysis types are collected respecting the affected artifacts. This article already sets up catalogues based on other research that can be used as a reference [37].

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

Some observations are made by Schmid [23] regarding shortcomings of Technical Debt that can be transformed and extended to EA Debt: "Technical Debt should be evaluated with respect to future evolution. [...] We need to differentiate between the structural issues (the potential Technical Debt) and the effect it has on actual development (the effective Technical Debt). [...] There is nothing like a technical-debt-free system." In general for the entire organization and our concept of EA Debt this implies:

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

4.1. Possible impacts on EA Debt

As mentioned before there are multiple artifacts that have an impact on EA Debt, identifying them should be an outcome of EA Debt evaluation. Having a look at the development of artifacts over time [30], known to cause EA Debt can be one way to approach the search of new impacts to the current EA Debt situation. Finding a slow or even stagnating development in an artifact A means there is almost no effort to lower the debt and, therefore, artifacts relying on A will increase the overall EA Debt. Starting a search at A can help finding impacts on EA Debt. On top of the already mentioned artifacts, that are able to cause EA Debt, there are more hidden or invisible aspects. We propose some of them:

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

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

Another impact, that has to be taken into consideration is that EA Debt itself should not be ignored and deferring the evaluation can, but not has to, increase EA Debt exponentially. Ideally, everyone working on artifacts of an EA

should at least know the concept, so everyone is aware that his acting can cause EA Debt.

4.2. Examples

We come up with two made up and simplified examples and one real-world example. The first example shows the broaden understanding of Technical Debt applied to the application layer. The second example presents an issue related to the process layer, and the third sketches the situation in a company that already conducts something that can be subsumed under EA Debts assessment.

4.2.1. Example 1 – EA Debts (Application Layer). A company is situated in the banking market and implemented its business critical applications on their mainframe until the end of the last decade. Due to a change in their IT-strategy, future applications should be developed using cloud environments. As the application landscape is comprised by more than 300 applications, a big bang scenario, where all applications are moved to the cloud, is unfeasible. Consequently, the applications are moved to the cloud step by step according to their application life-cycle rating.

The central EA department has defined a target landscape and a road map describing the way to get to the target landscape. This road map includes also two applications a and b, which should be moved from the mainframe to the cloud. Both applications depend on each other, which means that they use interfaces of each other. As it is planned that both applications should be moved to the cloud simultaneous, the interfaces of both applications can be developed within the target landscape.

Due to unforeseeable delays in the project that implements application b, it is not expected that a and b can still go live at the same time in the cloud. However, the interfaces of b are indispensable for the use of a. Therefore, a is developed in a way that it relies on the interfaces of the mainframe implementation of b. As this interface implementation is obviously not part of the target landscape, this will introduce EA Debts into the EA.

Nonetheless, there is no feasible alternative and, therefore, there is the need to take this additional effort, which leads to a worse quality of the overall EA. This could be solved by classical approaches like a cost-benefit analysis. However, the concept of EA Debt can help to create awareness for this quality issue along all stakeholders and that this EA Debt should be repaid as soon as application b has been moved to the cloud.

4.2.2. Example 2 – EA Debts (Process Layer). We consider a automotive supplier that produces engines for high-class cars. Due to several mergers, the organizational structure of the company is complex and is also reflected in their processes. This leads to an overall process that is not very efficient and the management of the company is already aware of this. However, the processes could not be optimized completely free as there are bargaining agreements that guarantee employees there actual job for a certain time.

This situation describes a trade-off between economical needs (optimize the processes) and legal requirements (bargaining agreement), which ends up in a not optimal situation from an EA point of view. However, to be aware of this fact, this issue should be rated as an EA Debt to make it visible and allow a conscious handling of it.

4.2.3. Example 3 – A Real-world Example. The following example reflects our insights from a real-world company that tries to identify the mismatch between their target landscape and their actual one. Further, they like to measure the proceeding. As the described issue is not related to technical issues, but to the alignment of the reality to EA defined goals, “Technical Debt” would be the wrong term. However, the company does not use the term “EA Debt”, but it could be subsumed beyond it.

The EA is divided into several hot-spots that are connected to certain areas of the IT-strategy. For every hot-spot, the enterprise architects identify need for action on a timeline. Every quarter, the enterprise architects conduct interviews with the responsible for each hot-spot and try to grasp the actual proceeding of the actions. The responses are given in a qualitative and a quantitative manner and were aggregated to report the overall process to the management. The management uses this input to steer the actual projects and to sell demands for actions to the business.

4.2.4. Findings. We showed, that EA Debt can be a tool, like monetary debt, to grow, but it can also be dangerous, if it potentially can not be repaid and consequently does not pay off. In contrast to monetary debt, EA Debt can suddenly occur, but does not have to. This means that enterprises have to be careful with taking EA Debt and even more with not lowering it. Still EA Debt points at those risks and problems to facilitate a sustainable development. Another example, illustrating an as-is situation with EA Debt in an EA would be rather uninteresting, because there are no universal guidelines yet, how to deal with EA Debt (see future work, Section 5) but every enterprise has to deal with EA Debt themselves. Nevertheless lowering EA Debt should be the main goal in such a situation. With the two examples, we demonstrate a way, how to use the concept of EA Debt as a tool to prevent bad decisions regarding the whole EA.

5. Implications and Future Work

This section discusses the constructive implications for real-life practice and future research, as well as the limitations found in this study. The objective of this session is to provide a future direction for real-world application and further research which has the potential to highly contribute to the EA domain.

By introducing a new metaphor that addresses a current gap in Technical Debt, the theoretical and empirical contribution can in turn benefit both the academia and EA practitioners, respectively, for research and real-life practices. The contribution of our work implies:

1) A realization of EA Debt concept for EA practitioners, IT representatives, and management. It allows EA practitioners to conceptualize the components that might affect the success of EA implementation;
2) A tool for EA practitioners to critically identify and examine EA Debts across four architectural layers on the basis of TOGAF;
3) A method for enterprise architects to effectively communicate the EA problems to management for the success of EA implementation;
4) A new research area for practices, approaches, models, or tools relevant to the context of EA Debt, such as EA Debt measurement, EA Debt identification, EA Debt monitoring, etc.

In practice, several existing methods can be used to facilitate the identification, detection, and measurement of EA Debt to base the estimates. EA Debt can be identified and measured by evaluating the performance of EA models using existing EA analysis tools proposed by Buschle et al. [44]. Moreover, Enterprise Coherence Assessment (ECA) proposed by Wagter et al. [45] can be used as an instrument to measure enterprise's level of coherence during enterprise transformation and thus incoherence between important aspects of the enterprise can be identified as EA Debt. Also, Őri [46] and Carvalho and Sousa [47] proposed approaches for business-IT misalignment detection.

Nevertheless, future work would suggest to develop a comprehensive framework, which is unique to the context of EA Debt. This framework should also elaborate on the integration of the established debt cycle into the processes that are already established in EA.

6. Conclusion

Implementing EA is essential to enhance the business-IT alignment in a holistic manner. However, academia, software developers, and organizations have been focusing on Technical Debt, which deals with the quality issues on code, application and system level. Considering the importance of EA in creating value to organizations, we asked for an adequate definition of EA Debt (RQ). Therefore, this work has extended the benefits of Technical Debt concept to the definition of EA Debts. In the pragmatic world, EA projects, programs or initiatives are implemented to realize the target architecture and EA Debt is expected to incur along the EA implementation process due to limited resources. We have asserted that the accumulation of EA Debt over time is likely to negatively affect the quality of an EA in responding to the complex business environment.

As we opened with the definition of EA Debt a new area of research, future work can elaborate on several different topics. First, the overall concept of EA Debt needs to be further evaluated, due to case studies, surveys, and so on. Second, it should be investigated how existing approaches can contribute to this new field, like EA quality assessments, EA best practices, etc. Third, existing concepts of code smell detection can be transferred to the domain of EA Debts to create an initial catalog of potential EA smells. Fourth, research should be conducted to identify EA smells, which solely arise in the field of EA and have no counterpart in existing Technical Debt domains. Fifth, we considered EA Debts in a "classical" environment and did not discuss the influence the concept can have on an "agile" environment. However, we assume that EA Debts can be a valuable instrument to inject EA thinking into agile projects. Last, research can elaborate on management methods for EA Debt, like how to decide which EA debt to repay next or how to involve EA stakeholders.

References

[1] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498–1516, 2013.

[2] W. Cunningham, "The WyCash portfolio management system," *ACM SIGPLAN OOPS Messenger*, vol. 4, no. 2, pp. 29–30, 1993.

[3] N. Zazworka, C. Seaman, and F. Shull, "Prioritizing Design Debt Investment Opportunities," *Proceeding of the 2nd workshop on Managing technical debt - MTD '11*, pp. 39–42, 2011.

[4] C. Seaman, Y. Guo, N. Zazworka, F. Shull, C. Izurieta, Y. Cai, and A. Vetrò, "Using technical debt data in decision making: Potential decision approaches," in *3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, 2012, pp. 45–48.

[5] Y. Guo and C. Seaman, "A portfolio approach to technical debt management," in *Proceeding of the 2nd working on Managing technical debt - MTD '11*, 2011, p. 31.

[6] M. Albarak and R. Bahsoon, "Prioritizing technical debt in database normalization using portfolio theory and data quality metrics," in *Proceedings of the 2018 International Conference on Technical Debt - TechDebt '18*, 2018, pp. 31–40.

[7] N. Zazworka, M. A. Shaw, F. Shull, and C. Seaman, "Investigating the impact of design debt on software quality," in *Proceedings of the 2nd Workshop on Managing Technical Debt*. ACM, 2011, pp. 17–23.

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

[9] A. Martini, J. Bosch, and M. Chaudron, "Architecture technical debt: Understanding causes and a qualitative model," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2014, pp. 85–92.

[10] F. A. Fontana, V. Ferme, M. Zanoni, and R. Roveda, "Towards a prioritization of code debt: A code smell intensity index," in *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*. IEEE, 2015, pp. 16–24.

[11] H. F. Soares, N. S. Alves, T. S. Mendes, M. Mendonça, and R. O. Spínola, "Investigating the link between user stories and documentation debt on software projects," in *2015 12th International Conference on Information Technology-New Generations*. IEEE, 2015, pp. 385–390.

[12] M. Lange, J. Mendling, and J. Recker, "An empirical analysis of the factors and measures of Enterprise Architecture Management success," *European Journal of Information Systems*, vol. 25, no. 5, pp. 411–431, 2016.

[13] M. Kuznetcov, "Measuring Architectural Technical Debt," Ph.D. dissertation, Radboud University Nijmegen, 2014.

[14] K. L. Smith and T. S. Graves, *An Introduction to Peaf: Pragmatic Enterprise Architecture Framework*. Pragmatic EC, 2011.

[15] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, vol. 29, no. 6, pp. 18–21, 2012.

[16] C. Seaman and Y. Guo, "Measuring and monitoring technical debt," *Advances in Computers*, vol. 82, pp. 25–46, 2011.

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

[18] S. McConnell, "Managing technical debt slides," 2007.

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

[20] M. Fowler, "Technical debt quadrant," https://martinfowler.com/bliki/TechnicalDebtQuadrant.html, last accessed on 2018-10-16.

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

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

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

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

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

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

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

[28] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, "An enterprise perspective on technical debt," in *Proceedings of the 2Nd Workshop on Managing Technical Debt*, ser. MTD '11. New York, NY, USA: ACM, 2011, pp. 35–38.

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

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

[31] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the principal of an application's technical debt," *IEEE Software*, vol. 29, no. 6, pp. 34–42, 2012.

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

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

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

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

[36] J. C. Henderson and H. Venkatraman, "Strategic alignment: Leveraging information technology for transforming organizations," *IBM Systems Journal*, vol. 38, no. 2.3, pp. 472–484, 1999.

[37] D. Őri, "An artifact-based framework for business-it misalignment symptom detection," in *The Practice of Enterprise Modeling*, J. Horkoff, M. A. Jeusfeld, and A. Persson, Eds. Cham: Springer International Publishing, 2016, pp. 148–163.

[38] 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*. IEEE, 2006, pp. 30–30.

[39] The Open Group, "TOGAF." [Online]. Available: http://www.opengroup.org/togaf

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

[41] 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*, vol. 4, 2017, pp. 14–21.

[42] C. Schmidt, T. Widjaja, and A. Schütz, "Messung der komplexität von it-landschaften auf der basis von architektur-metamodellen: Ein generischer ansatz und dessen anwendung im rahmen der architektur-transformation," in *GI-Jahrestagung*, 2013.

[43] N. Niu, L. D. Xu, J. C. Cheng, and Z. Niu, "Analysis of architecturally significant requirements for enterprise systems," *IEEE Systems Journal*, vol. 8, no. 3, pp. 850–857, 2014.

[44] M. Buschle, J. Ullberg, U. Franke, L. Robert, and T. Sommestad, "A Tool for Enterprise Architecture Analysis using the PRM Formalism," in *Conference: Information Systems Evolution (CAiSE)*, 2010.

[45] R. Wagter, E. Proper, and D. Witte, "A Practice-Based Framework for Enterprise Coherence," in *Working Conference on Practice-Driven Research on Enterprise Transformation (PRET)*, vol. 120, 2012, pp. 77–95.

[46] D. Őri, "Business-IT Misalignment Symptom Detection based on Enterprise Architecture Analysis," in *1st Enterprise Engineering Working Conference Forum co-located with the 7th Enterprise Engineering Working Conference (EEWC)*, 2017.

[47] G. Carvalho and P. Sousa, "Business and information systems Mis-Alignment Model (BISMAM): An holistic model leveraged on misalignment and medical sciences approaches," in *BUSITAL*, vol. 336, 2008, pp. 104–119.