A computational model of a natural language processing system for architectural technical debt management

A computational model of a natural language processing system for architectural technical debt management

B Perez[1,2], C Castellanos[2], and D Correal[2]

[1] Grupo de Investigación en Inteligencia Artificial, Universidad Francisco de Paula Santander, San José de Cúcuta, Colombia
[2] Grupo de Investigación en Tecnologías de Información y Construcción de Software, Universidad de los Andes, Bogotá, Colombia

E-mail: borisperezg@ufps.edu.co, cc.castellanos87@uniandes.edu.co

Abstract. Technical debt is a concept applied to decisions taken to favor rapid production, at the expense of future capacities to evolve or improve the product or service. This concept is transversal to all sciences, especially in experimental physics and systems. However, technical debt is difficult to manage because there is a lack of time, expertise, or knowledge on how to perform it. This paper proposes a computational model of technical debt management in software architecture, based on observations made in software teams in their daily work. This computational model exploits natural language processing and model-testing techniques on software project artifacts to build a model that allows localizing and visualizing the impact produced by the technical debt and its payment strategies. This proposal aims to support teams of architects to explain the current and future impact of the debt injected as a result of decisions made.

1. Introduction

Architectural technical debt (ATD) is a metaphor used to describe decisions taken by software architects to accomplish short-term goals but possibly negatively affecting the long-term health of the system [1]. These decisions could be made consciously to promote certain quality attributes over others, or could be made unconsciously due to a lack of field experience, or even because of some poorly defined requirements [2]. Architectural decisions are the most important source of technical debt [3].

ATD needs to be kept under control to keep the system healthy in the long term. However, ATD is difficult to manage (identification, measurement, prioritization, repayment, and monitoring) by software architect teams because there is a lack of time, lack of expertise and lack of knowledge on how to perform such activities. ATD required to establish what, where and when their items were injected, and their current and future impact in the overall system. Unfortunately, much of this knowledge usually vaporizes [4] but it could be possible to find some of them documented in multiple artifacts, such as chat logs and emails.

Several approaches have been proposed to support ATD identification and management [5–7]. However, it is not clear the types of unstructured sources and the required analysis process. Li, *et al.* in [8] proposed a decision-based ATD management (ATDM) approach, but requires

a strong presence of software architects. Therefore, there are no studies focused on ATDM at architecture-level only and exploiting the information over heterogeneous artifacts gathered from the different activities of the software architecture design process.

In the literature, several studies [9–11] have defined the main requirements to face for ATDM: (i) To discover and pay ATD early in the software lifecycle to save a significant amount of maintenance costs in projects, (ii) to develop approaches dealing with artifacts different to source code to manage ATD, and (iii) identification and assessments need to be performed with as little human intervention as possible.

Given the aforementioned requirements, the goal of this study is to manage architectural technical debt (ATD)'s life cycle of a software system project. This is done by supporting the identification, measurement, and tracking of the impact produced by the ATD injected and its payment practices on other architectural decisions. To this end, this proposal exploits several techniques such as natural language processing and model checking over a set of heterogeneous artifacts from a software development project. Results reported by this proposal could help software practitioners in keeping their software systems healthy, and thus, reducing costs related to maintainability and evolvability of software components.

The remainder of this paper is organized as follows: section 2 discusses related work. Section 3 provides details on the proposed model. Finally, section 4 discusses the proposal and conclusions.

2. Related work

Several approaches have been proposed to support ATD identification and impact measurement. Verdecchia in [5] proposed a technique focused mainly on complementing source code analysis with a review of issue trackers, questions and answer sites (*e.g.* Stack Overflow), documentation and other software artifacts. However, it is not clear if unstructured sources like emails, chat logs or video/audio transcriptions are also included. Also, the strategy for analyzing these sources is not described, nor how the impact of the identified ATD could be quantified.

Borrego in [6] stated that unstructured textual electronic media (UTEM), comments on repository hosting services, instant messaging, IM, email, can be used as a source of AK. He established a relationship between a UTEM and an architectural decision.

Musil, *et al.* in [7] employed sources that can be either organization-internal (*e.g.*, enterprise wikis, AK repositories, AKM tools) or external (*e.g.*, websites, blogs, archival publications) to be automatically indexed by the system.

Regarding ATDM, Li, Liang, and Avgeriou in [8] proposed a decision-based ATDM approach supporting the five activities of the ATD life cycle: identification, measurement, prioritization, repayment, and monitoring. Identification and measurement require a strong presence of software architects.

To the best of our knowledge, there are no studies focused on ATDM at architecture-level only and exploiting the information over heterogeneous artifacts gathered from the different activities of the software architecture design process.

3. Proposal

In this section, we present a semi-automated approach to manage ATD's life cycle. This proposal is based on natural language processing and model checking techniques on heterogeneous project artifacts to identify, measure and tracks the impact produced by injected ATD (consciously or unconsciously). It is focused on finding ATD items at architecture level only, measuring the impact of its repayment strategy based on architecture decisions and allowing the debt to be anticipated or corrected before the implementation process begins.

In this proposal, we acknowledge two kinds of ATD: decision ATD [12] and solution ATD [13]. The first is related to the ATD injected when a decision is made, consciously or unconsciously.

The latter is related to ATD injected as a consequence of the presence of architectural smells among architectural components. Our proposal consists of five stages (Figure 1):

- Stage 1. Extraction: focused on extracting all relevant information from heterogeneous artifacts and the explicit knowledge of the software architects on previously identified ATD items.
- Stage 2. Synthesis: focused on putting all this information together and analyzing it to identify information related to software architecture decisions.
- Stage 3. Analysis: focused on identifying and measuring ATD items based on information gathered from architectural decisions.
- Stage 4. Reporting: focused on presenting information to software architects related to the ATD identified.
- Stage 5. Planning & feedback: focused on giving feedback to the information related to ATD items for updating and learning.

These stages are iterative and need to be performed multiple times during the life cycle of the project. An iteration will depend on the software architects' team and could be performed only in some stages. The remainder of this section gives an in-depth review of the stages.

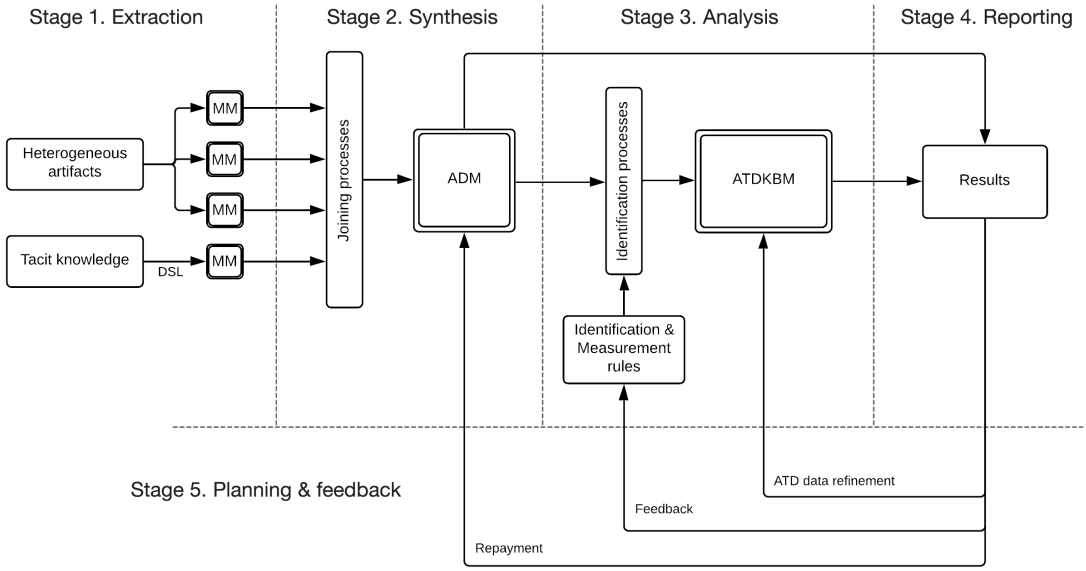

Figure 1. Framework design.

3.1. Stage 1. Extraction

This stage is focused on the semi-automatic extraction of information related to architectural decisions, specifically, its pros and cons. This stage is based on project heterogeneous artifacts, such as requirements document, architecture document, architectural viewpoints, chat logs (Slack, Microsoft Teams), emails (between architects and between architects and clients), commit log messages, wikis, issue trackers and transcribed video/audio meetings. Each artifact will require a specific procedure to extract its information and represent them into a basic structure. This will allow further refinement in the extraction process before going into the synthesis stage.

In addition to the architecture document, another valuable document is the report of the architectural evaluation process, such as architecture tradeoff analysis method (ATAM). The goal of architecture evaluation using ATAM is to understand the consequences of architectural

decisions with respect to the quality attribute requirements of the system. These consequences could be linked to ATD.

3.2. Stage 2. Synthesis

This stage is focused on joining the models populated in Stage 1 and enhance as much as possible the architectural decisions made in the project. Information from this stage will be the input for ATD analysis in stage 3.

The goal of this stage is to identify the pros and cons of architectural decisions based on text gathered from heterogeneous artifacts. Pros and cons could be done using sentiment analysis (SA) to classify the polarity (positive, negative, or neutral) of a given text. Positive sentiment is the pros of the decision discussed. Negative sentiment is the cons of the decision discussed. However, SA attempts to detect the overall polarity of a given text, regardless of the entities mentioned and their aspects. Aspect based sentiment analysis (ABSA) will be used in this stage to detect the main aspects of an entity and to estimate the overall sentiment of the given texts per aspect.

This stage includes an implementation of a named-entity recognizer (NER) to identify mentions to architectural decisions and their possible links with other architectural decisions. Latent Dirichlet allocation (LDA), a topic modeling technique, will also be used to discover the main topics of a document.

Architectural decisions will be represented using the decision-capturing metamodel presented by Zimmermann, *et al.* [14] and used within the architectural decision knowledge Wiki/architectural decision knowledge web tool.

3.3. Stage 3. Analysis

The goal of this stage is to build an ATD map over the architecture. It is required to acknowledge the ATD in the architecture and also to establish the source of it, i.e. the artifact where it was injected. It is important to remark that an architecture decision could be an ATD item at first, but then become a good decision over the lifetime of the project. For instance, designs that are over-engineered and thus costly to maintain in the first place may become appropriate with requests for new features and capabilities. Similarly, decisions that are good in the first place may become an ATD item in the future, for instance, because experienced developers leave the team, and inexperienced developers join the team. Changes in business direction may also create ATD that was not in the first place, for example, if the product should be market in regions that have export control restrictions for technologies used. Those aspects impact the "state of architectural TD items" over the lifetime of a project.

ATD can be either explicitly injected or implicitly injected and be made either consciously or unconsciously. How to address the acknowledgment of ATD using these scenarios is presented below:

- S1. Conscious & explicit: ATD is acknowledged and wrote down using a DSL proposed by Rebel.
- S2. Conscious & implicit: ATD could be identified based on information gathered and processed by stages 1 y 2 of our proposal.
- S3. Unconscious & implicit: ATD could be identified through architectural models by applying model checking techniques related to software architecture patterns.
- S4. Unconscious & unknown: ATD could be identified through artifacts dependency based on external knowledge bases.

3.3.1. Architectural technical debt knowledgde base model. ATD acknowledged by our proposal will be represented in a model conformed to the metamodel presented in Figure 2. This metamodel is an extension of the ATD conceptual model presented by Avgeriou, *et al.* [11]. Rebel metamodel will include additional elements such as ATD kind, repayment plan strategies and heterogeneous artifacts involved in the ATD identification. Repayment strategies will include an estimate of when repayment will take place into account, as well as refactoring decisions to be made, and under what conditions.

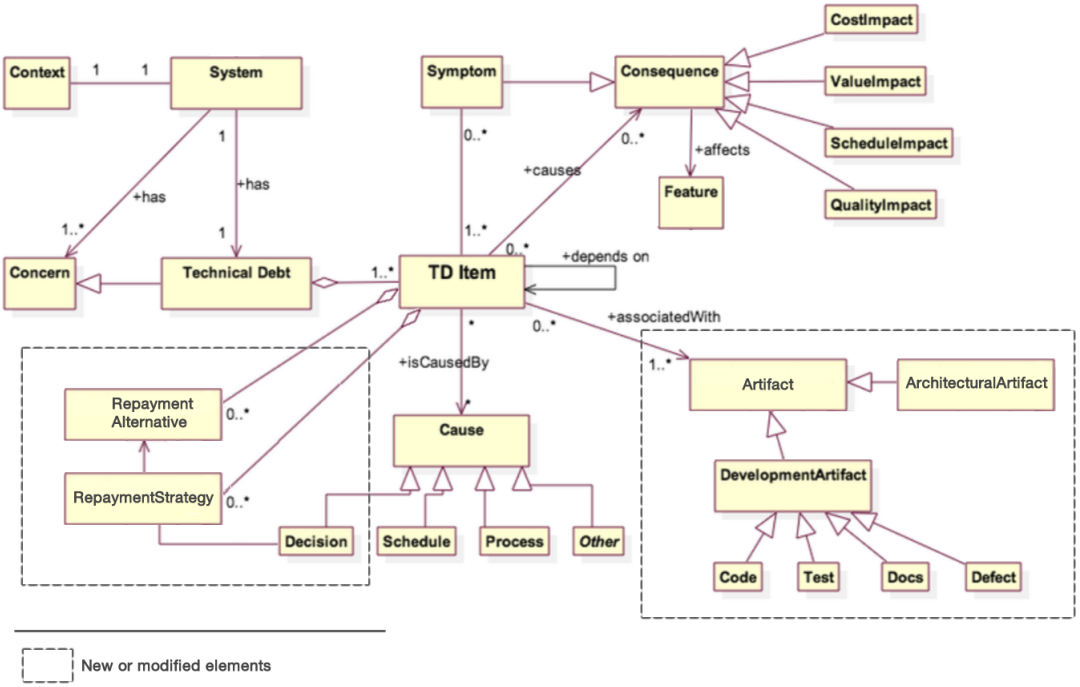

Figure 2. Proposed metamodel.

3.3.2. Impact measurement. ATD item impact measurement is based on three variables: (i) Implementation cost (IC), based on the architect's personal opinion about the relative implementation cost of fixing or repay the debt; (ii) Severity (S), measures the severity of ATD injected on the architecture; (iii) Number of architectural elements affected (IA), based on the proportional number of architectural elements affected by the ATD using a ten values scale. These variables are used to represent complexity, which is related to the cost of change or reworks [15]. In our work, this cost is interpreted as a relative value because it is based on estimates.

3.4. Stage 4. Reporting

This stage focused on presenting the ATD item candidate list, including a level of confidence, to the software architect in order to be checked for accuracy. A software architect uses this information to learn about ATD and to keep a track of it. We seek to provide the architect with information that he did not have before, and that he/she can now use to make more impact-conscious decisions. This stage is important because it helps to increase the awareness of the impact that ATD had on the system [1].

Displaying of ATD will be based on the following viewpoints: ATD Detail Viewpoint (presents the detailed information of individual ATD items), and ATD Decision Viewpoint (shows the

relationships between ATD items and architecture decisions). ATD Decision Viewpoint will be extended to include links among architectural decisions based on initial decisions and repayment decisions.

3.5. Stage 5. Planning and feedback

This information is used to provide feedback for the models used in the analysis stage to support the improvement of the accuracy of its activities. This stage is used to provide a mechanism to allow software architects to declare how they intend to pay the ATD. This information could be used to suggest repayment strategies.

It is important to remark that repayment strategies for ATD are also very much dependent on non-technical issues. For instance, if ATD is in a set of features that were considered in key product features, but market feedback shows these features are not as important for the customer value of the product, the pressure for resolving the architectural TD decreases significantly. Also, if the resolution of the ATD item may also break compatibility with the installed base, it may not be possible to fully resolve without increased effort external to the project or with losses in business and market attractiveness.

4. Conclusions

We have presented a model-driven approach for managing ATD's life cycle during the software architecture design process and focusing on identification at the architecture level. This proposal is based on an exhaustive analysis process over heterogeneous architect, in order to locate ATD injection motivators and its impact. Machine learning techniques and Model checking techniques are used to find ATD over the architecture. Reporting ATD items will allow architects to be conscious of any ATD associated with decisions and allow them to define strategies that eventually lead to solutions.

The importance of this proposal lies in the lack of research on ATD management using artifacts produced during the design of the software architecture. Findings provided by the SLR show us there is a gap in ATD identification where artifacts other than source code can be used to provide evidence of ATD. Our contributions required by the software architects community. This is an area of research ready to work and to go further. Prime "customers" of this work are seasoned architects with many years of experience.

This research needs to face several challenges. First, most of the software architecture decisions are not fully documented. A client does not pay for documentation. The rationale of decisions usually is divided among several artifacts, different ways to represent them and different ways to talk about them. Another challenge will be access to the heterogeneous artifacts. It is crucial to have access to all artifacts, including emails and chat logs. Company access is a nontrivial challenge. Also, more complex projects will require more artifacts to analyze and this will include different storing structures.

References

[1] Besker T, Martini A and Bosch J 2018 Managing architectural technical debt: A unified model and systematic literature review *Journal of Systems and Software* **135** 1

[2] Falessi D, Cantone G, Kazman R and Kruchten P 2011 Decision-making techniques for software architecture design: A comparative survey *ACM Computing Surveys* **43(4)** 33

[3] Ernst N A, Bellomo S, Ozkaya I, Nord R L and Gorton I 2015 Measure it? manage it? ignore it? software practitioners and technical debt (esec/fse 2015) *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (New York: Association for Computing Machinery) pp 50–60

[4] Jansen A, van der Ven J, Avgeriou P and Hammer D K 2007 Tool support for architectural decisions *Working IEEE/IFIP Conference on Software Architecture (WICSA'07)* (Mumbai: IEEE) pp 44–54

[5] Verdecchia R 2018 Architectural technical debt identification: Moving forward *IEEE International Conference on Software Architecture Companion (ICSA-C)* (Seattle: IEEE) pp 43–44

[6] Borrego G 2016 Condensing architectural knowledge from unstructured textual media in agile gsd teams *IEEE 11th International Conference on Global Software Engineering Workshops (ICGSEW)* (Irvine: IEEE) pp 69–72
[7] Musil J, Ekaputra F J, Sabou M, Ionescu T, Schall D, Musil A and Biffl S 2017 Continuous architectural knowledge integration: Making heterogeneous architectural knowledge available in large-scale organizations *IEEE International Conference on Software Architecture (ICSA)* (Gothenburg: IEEE) pp 189–192
[8] Li Z, Liang P and Avgeriou P 2014 Architectural debt management in value-oriented architecting *Economics-Driven Software Architecture* (Waltham: Morgan Kaufmann) pp 183–204
[9] Xiao L, Cai Y, Kazman R, Mo R and Feng Q 2016 Identifying and quantifying architectural debt *IEEE/ACM 38th International Conference on Software Engineering (ICSE)* (Austin: IEEE) pp 488–498
[10] Li Z, Avgeriou P and Liang P 2015 A systematic mapping study on technical debt and its management *Journal of Systems and Software* **101** 193
[11] Avgeriou P, Kruchten P, Ozkaya I and Seaman C 2016 Managing technical debt in software engineering *Dagstuhl Reports* **6(4)** 110
[12] Li Z, Avgeriou P and Liang P 2015 Architectural technical debt identification based on architecture decisions and change scenarios *12th Working IEEE/IFIP Conference on Software Architecture* (Montreal: IEEE) pp 65–74
[13] Martini A, Sikander E and Madlani N 2018 A semi-automated framework for the identification and estimation of architectural technical debt: A comparative case-study on the modularization of a software component *Information and Software Technology* **93** 264
[14] Zimmermann O, Koehler J, Leymann F, Polley R and Schuster N 2009 Managing architectural decision models with dependency relations, integrity constraints, and production rules *Journal of Systems and Software* **82(8)** 1249
[15] Nord R L, Ozkaya I, Kruchten P and Gonzalez-Rojas M 2012 In search of a metric for managing architectural technical debt *Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture* (Helsinki: IEEE) pp 91–100