Managing technical debt in software architecture

Li, Zengyang

Managing Technical Debt in Software Architecture

PhD Thesis

By **Zengyang Li** (李增扬) at the Software Engineering and ARCHitecture (SEARCH) group of the University of Groningen, The Netherlands.

ISBN: 978-90-367-7918-0

The research described in this thesis has been carried out at the Faculty of Mathematics and Natural Sciences, University of Groningen, The Netherlands.

university of groningen / faculty of mathematics and natural sciences

The author acknowledges the financial support by the AFR PhD Grant from the Fonds National de la Recherche Luxembourg, under the contract No. 895528.

university of groningen

Managing Technical Debt in Software Architecture

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. E. Sterken
and in accordance with
the decision by the College of Deans.

This thesis will be defended in public on

Monday 29 June 2015 at 11.00 hours

By

Zengyang Li

born on 17 October 1982
in Jiangxi, China

Supervisors
Prof. P. Avgeriou
Prof. P. Liang

Assessment committee
Prof. A.N. Chatzigeorgiou
Prof. P. Kruchten
Prof. A.C. Telea

Samenvatting

In de afgelopen jaren is er significante belangstelling geweest voor technische schuld. Het concept van technische schuld ging eerst over kwesties met interne kwaliteit maar is nu uitgebreid naar de hele levenscyclus van software, waar ook softwarearchitectuur en testen onder valt. Op het architectuurniveau wordt architectuurtechnische schuld voornamelijk opgelopen door architectuurbeslissingen die opzettelijk of onopzettelijk systeembrede kwaliteitsattributen in gevaar brengen, vooral onderhoudbaarheid en evolueerbaarheid. Gezien de fundamentele invloed van softwarearchitectuur op kwaliteitsattributen, inclusief onderhoudbaarheid en evolueerbaarheid, heeft technische schuld op het architectuurniveau een grotere en bredere impact op deze kwaliteitsattributen dan andere typen technische schuld, zoals technische schuld op het niveau van broncode. Daarom moet architectuurtechnische schuld op een systematische en toepasselijke manier gemanaged worden, om zo de gezondheid van de software architectuur te verbeteren, en om de kosten van systeemevolutie en -onderhoud op de lange termijn te optimaliseren.

Het meeste onderzoek naar technische schuld focust zich op technische schuld op het niveau van broncode, terwijl technische schuld op architectuurniveau en het managen daarvan minder goed verkend blijft. Op dit moment is er een gebrek aan een algeheel proces voor het systematisch managen van architectuurtechnische schuld, evenals een gebrek aan een benadering voor concrete activiteiten rondom architectuurtechnische schuld, in het bijzonder voor het identificeren, meten, en documenteren van architectuurtechnische schuld.

Voorafgaand aan het onderzoeken van de belangrijkste problemen in het managen van architectuurtechnische schuld moesten we eerst het concept van technische schuld en de huidige staat van onderzoek naar het managen van technische schuld goed begrijpen. Dit kon ons helpen om een degelijk begrip van architectuurtechnische schuld te krijgen en ons inspireren om een toepasselijke aanpak voor het managen van architectuurtechnische schuld te bedenken. Hiervoor hebben we een systematische literatuurmappingonderzoek over technische schuld en het managen van deze schuld uitgevoerd. De hoofdresultaten zijn als volgt samengevat. (1) Technische schuld is geclassificeerd in tien types volgens de stages in het softwareontwikkelproces, en technische schuld in broncode was het meest bestudeerde type technische schuld. (2) Rente, principe, en risico zijn de meest gebruikte begrippen voor het beschrijven en uitleggen van het concept technische schuld. (3) In de meeste studies wordt beargumenteerd dat technische schuld een negatief effect heeft op de onderhoudbaarheid van een software systeem. (4) Acht management activiteiten rondom architectuurtechnische schuld zijn geïdentificeerd; onder deze activiteiten hebben het identificeren, meten, en herbetalen van van technische schuld het meeste aandacht gekregen, terwijl het representeren/documenteren van technische schuld het minste aandacht kreeg. (5) Van de 29 tools die gebruikt worden voor het managen van technische schuld zijn er

slechts vier die puur bedoeld zijn voor het managen van architectuurtechnische schuld; de meeste tools ondersteunen alleen het managen van technische schuld in broncode en ontwerp, terwijl er weinig tools ondersteuning bieden voor het managen van andere typen technische schuld, bijvoorbeeld architectuurtechnische schuld.

Na het verkrijgen van een begrip over de huidige stand van zaken in het onderzoek naar technische schuld hebben we geprobeerd te verkennen hoe architectuurtechnische schuld kan worden gemanaged. Als eerste heben we een voorstel gedaan voor een conceptueel model van architectuurtechnische schuld welke is gebaseerd op het begrip van technische schuld, alsmede een sjabloon voor elementen van architectuurtechnische schuld welke is gebaseerd op dit model. Vervolgens hebben we een managementproces voor architectuurtechnische schuld ontwikkeld welke gebruik maakt van het conceptuele model, om zo het maken en evalueren van beslissingen vanuit een waarde-georiënteerd perspectief in architectuurontwerp te faciliteren. Het managementproces voor architectuurtechnische schuld bestaat uit zes activiteiten: identificeren, meten, prioriteren, herbetalen, in de gaten houden, en documenteren van architectuurtechnische schuld. Een voorbeeld in het bedrijfsleven waarin de voorgestelde benadering (het proces en model) voor het managen van architectuurtechnische schuld in architecturale synthese en evolutie wordt gebruikt laat zien hoe architectuurtechnische schuld kan worden gemanaged tijdens het werken aan een architectuur. Dit onderzoek kan op de lange termijn worden gebruikt om een beheersbare en voorspelbare balans tussen de waarde en kosten van architectuurontwerp te realiseren.

De eerste stap in ons management proces voor architectuurtechnische schuld is om architectuurtechnische schuld te identificeren. Bestaande benaderingen voor het identificeren van architectuurtechnische schuld zijn voornamelijk gebaseerd op de analyse van broncode, en hebben daarom bepaalde tekortkomingen: (1) ze kunnen alleen problemen in de systeemimplementatie identificeren; (2) ze kunnen alleen worden toegepast als een systeem geimplementeerd is in broncode; en (3) ze missen een mechanisme voor het bevestigen of een geïdentificeerde architectuurtechnische schuld een echte architectuurtechnische schuld is of niet. Om deze problemen aan te pakken hebben we een benadering voor het identificeren van architectuurtechnische schuld voorgesteld welke is gebaseerd op architectuurbeslissingen en wijzigingsscenarios. Door middel van een casestudy in een telecommunicatiebedrijf heben we de effectiviteit en gebruikersvriendelijkheid van deze benadering geëevalueerd. De resultaten laten zien dat de voorgestelde benadering nuttig is en simpel te gebruiken voor het identificeren van technische schuld, en dat het ook ondersteuning bied voor het plannen van releases en voor het meten van de rente van architectuurtechnische schuld.

Het is moeitlijk om architectuurtechnische schuld precies te meten, maar het spreekt voor zich om een indicatie te geven van de totale hoeveelheid architectuurtechnische schuld in een software systeem. Indicatoren van architectuurtechnische schuld kunnen laten zien wat de richting is van wijzigingen in

de architectuurtechnische schuld van opeenvolgende versies van een software systeem. Eén indicator van architectuurtechnische schuld is het gemiddelde aantal gewijzigde componenten per commit: een hoger aantal gewijzigde componenten per commit is een indicatie van meer architectuurtechnische schuld in een software systeem. Het is echter moelijk, en soms onmogelijk, om het aantal gewijzigde componenten per commit te berekenen omdat de data (i.e., het logbestand met commits) niet altijd aanwezig of toegangkelijk is. We stellen voor om meetwaardes over de modulariteit van software te gebruiken welke direct kunnen worden berekend op basis van broncode als een plaatsvervaner voor het aantal gewijzigde componenten per commit, om zo een indicatie van architectuurtechnische schuld te geven. We hebben de correlatie tussen het aantal gewijzigde componenten per commit en modulariteitmetrieken gevalideerd door meerdere holistische casestudies binnen dertien open source projecten. De resultaten van dit onderzoek suggereren dat twee modulariteitmetrieken, namelijk de index van pakketwijzigingsimpact en de index van pakket-doel-focus, een significante corelatie hebben met het aantal gewijzigde componenten per commit, en deze metrieken kunnen daarom gebruikt worden als alternatieve indicators voor architectuurtechnische schuld.

Nadat architectuurtechnische schuld is geïdentificeerd en gemeten moet de kennis over architectuurtechnische schuld expliciet worden gedocumenteerd waardoor de rest van de activiteiten rondom het managen van architectuurtechnische schuld wordt gefaciliteerd. Bestaand onderzoek over het documenteren van architectuurtechnische schuld is echter gelimiteerd en neemt niet alle zorgen van belanghebbenden over architectuurtechnische schuld weg. We stellen zes architectuur viewpoints voor welke gerelateerd zijn aan architectuurtechnische schuld. Elk viewpoint is een raamwerk voor een aantal zorgen over architectuurtechnische schuld. Al deze zorgen zijn systematisch uit literatuur gehaald in het voorgenoemde systematische literatuurmappingonderzoek over architectuurtechnische schuld. De zes viewpoints helpen gezamenlijk om een omvatten begrip over architectuurtechnische schuld in een software systeem te krijgen, en dit geeft ondersteuning voor het maken van architectuurbeslissingen. Om de effectiviteit van viewpoints over architectuurtechnische schuld bij het documenteren van architectuurtechnische schuld te evalueren hebben we een casestudy uitgevoerd bij hetzelfde bedrijf waar de voorgenoemde benadering voor identificatie van architectuurtechnische schuld was gevalideerd. De resultaten van de casestudy laten zien dat gedocumenteerde views over architectuurtechnische schuld de documentatie van architectuurtechnische schuld effectief kunnen faciliteren.

De voorgenoemde benaderingen voor het managen van architectuurtechnische schuld hebben hun beperkingen terwijl ze niet alle activiteiten in het managementproces van architectuurtechnische schuld afvangen. Wij verkenden de toepassing van kennis-gebaseerde benaderingen in software architectuur door een systematisch literatuurmappingonderzoek, om zo het huidige management van architectuurtechnische schuld (wat een type architectuurkennis is) te verbeteren. We zagen (1) de representatie van architectuurtechnische schuld en architectuur in een formele vorm, en het verder toepassen van redeneringstechnieken die gebaseerd zijn op deze formele representaties, kan worden gebruikt voor het identificeren, meten,

prioriseren, en in de gaten houden van architectuurtechnische schuld, en (2) dat het delen en hergebruik van kennis over architectuurtechnische schuld en gerelateerd architectuurkennis ondersteuning kan bieden aan het identificeren, documenteren, prioriseren, herbetalen, en in de gaten houden van architectuurtechnische schuld.

Abstract

Technical debt (TD) has received significant attention in the past few years. The concept of TD was initially concerned with internal quality issues in coding, and currently it has been extended to the whole software lifecycle, such as software architecture and testing. At the architecture level, architectural technical debt (ATD) is mainly incurred by architecture decisions that intentionally or unintentionally compromise system-wide quality attributes, particularly maintainability and evolvability. Considering the fundamental influence of software architecture on quality attributes, including maintainability and evolvability, TD at the architecture level (i.e., ATD) has greater and wider impact on these quality attributes than other TD types, such as code-level TD. Thus, ATD needs to be systematically managed in an appropriate manner, in order to improve the health of the software architecture and optimize the cost of maintenance and evolution of the system in the long term.

Most research on TD focuses on TD at the source code level while TD at the architecture level and its management remain under-explored. Currently, there is a lack of an overall process for systematically managing ATD, as well as approaches for concrete ATD management activities, particularly for ATD identification, measurement, and documentation.

Before investigating the key problems in ATD management, we first needed to obtain a comprehensive understanding on the concept of TD and the current state of research on TD management (TDM). This could help us to build a solid understanding on ATD and inspire us to come up with appropriate approaches for ATD management. To this end, we conducted a systematic mapping study on TD and its management. The main results are summarized as follows. (1) TD is classified into ten types according to the stages of the software development lifecycle, and code TD was the most studied TD type. (2) Interest, principal, and risk are the most frequently-used notions to describe and explain the TD concept. (3) Most studies argue that TD negatively affects the maintainability of the software system. (4) Eight TDM activities were identified; among the activities, TD identification, measurement, and repayment received the most attention, while TD representation/documentation received the least. (5) Among the 29 tools used for managing TD, only four are dedicated tools for TDM; most tools only support managing code and design TD, while few tools support the management of other types of TD, e.g., ATD.

After having gained an understanding on the state of the art on TD research, we tried to explore how to manage ATD. First we proposed a conceptual model of ATD based on the understanding on TD, and an ATD item template based on this model; then we developed an ATD management process that utilizes this conceptual model, in order to facilitate decision-making and decision-evaluation in a value-oriented perspective in architecture design. The ATD management process is comprised of six activities: ATD identification, measurement, prioritization, repayment, monitoring, and documentation. An industrial example using the proposed approach (the

process and model) of ATD management in architecture synthesis and evaluation shows how ATD can be managed in architecting. The contribution of this work provides a controllable and predictable balance between the value and cost of architecture design in the long term.

In our ATD management process, the first step is to identify ATD. Existing ATD identification approaches are mainly based on source code analysis and thus suffer from certain shortcomings: (1) they can only identify issues at the system implementation; (2) they can only be employed after the systems is implemented in code; and (3) they lack a mechanism to confirm whether the identified ATD is real ATD or not. To address these issues, we proposed an ATD identification approach based on architecture decisions and change scenarios. We evaluated the effectiveness and usability of this approach, through an industrial case study in a large telecommunications company. The results show that the proposed approach is useful and easy to use for ATD identification, and it also supports release planning and ATD interest measurement.

It is difficult to precisely measure ATD, but it makes sense to indicate the amount of the total ATD in a software system. ATD indicators can show the change direction of the ATD in sequential versions of the software system. One indicator of ATD, is the average number of modified components per commit (ANMCC): a higher ANMCC indicates more ATD in a software system. However, it is difficult and sometimes impossible to calculate ANMCC, because the data (i.e., the log of commits) are not always available or accessible. We proposed to use software modularity metrics, which can be directly calculated based on source code, as a substitute of ANMCC to indicate ATD. We validated the correlation between ANMCC and modularity metrics through a holistic multiple case study on thirteen open source software projects. The results of this study suggested that two modularity metrics, namely Index of Package Changing Impact (IPCI) and Index of Package Goal Focus (IPGF), have significant correlation with ANMCC, and therefore can be used as alternative ATD indicators.

After ATD is identified and measured, the knowledge about ATD needs to be explicitly documented thereby facilitating the rest of the activities in ATD management. Existing work on ATD documentation is rather limited and it cannot address all stakeholders' concerns on ATD. We proposed six architecture viewpoints related to ATD (ATD viewpoints in short). Each viewpoint frames a number of concerns on ATD. All these concerns were systematically extracted from literature in the aforementioned systematic mapping study on TD. The six ATD viewpoints together help to get a comprehensive understanding of ATD in a software system, thereby providing support for architecture decision-making. To evaluate the effectiveness of the ATD viewpoints in documenting ATD, we conducted an industrial case study in the same company where the aforementioned ATD identification approach was validated. The case study results show that the documented ATD views can effectively facilitate the documentation of ATD.

The aforementioned approaches for ATD management have their limitations while they do not cover all the activities in the ATD management process. We explored the

application of knowledge-based approaches in software architecture through a systematic mapping study, in order to improve the current management of ATD, which is a type of architectural knowledge. We found that (1) the representations of ATD and architecture in a formal form and further applying reasoning techniques based on these formal representations can support ATD identification, measurement, prioritization, and monitoring, and (2) the sharing and reuse of the knowledge on ATD and related architectural knowledge can support ATD identification, documentation, prioritization, repayment, and monitoring.

Acknowledgments

The journey of finishing this PhD thesis is full of challenges. During this journey, a lot of people have provided me with their help, support, and encouragement.

First of all, I am extremely grateful to my supervisor Paris Avgeriou, who provided me with the opportunity to pursue a PhD degree at the University of Groningen. I especially thank him for the freedom, guidance, support, and patience that he gave me during my PhD research. In the occasions when I got lost and almost felt hopeless, he always had good suggestions and gave me confidence to continue. Paris shared with me not only his experience of being a successful academic but also his wisdom towards work and life. Besides benefiting from his professional supervision in research, I also learned a lot from him on time management, task decomposition, healthy life styles, etc. Paris is the best supervisor that I can expect.

I deeply thank my co-supervisor Peng Liang, also an ideal supervisor, who helped me a lot with my research. I highly appreciate his active guidance and constant encouragement. Whenever I suffered from difficulties and doubts in my research, I always turn to him for help. He was always willing to discuss with me in detail and with patience. His suggestions and comments have been constructive and valuable. Peng is not only my co-supervisor in research but also a best friend in my life. We have known each other for almost ten years. I sincerely expect our future fruitful collaboration and lifelong friendship.

I express my gratitude to Prof. Nicolas Guelfi for his great support in my PhD position application, for hosting my visit to the Lab of Advanced Software Systems (LASSY) at the University of Luxembourg, and for our pleasant collaboration on architectural technical debt.

I am grateful to my reading committee members Prof. Alexander N. Chatzigeorgiou, Prof. Philippe Kruchten, and Prof. Alexandru C. Telea. I appreciate their valuable time and comments on this thesis, and their attendance at my defense ceremony.

Special thanks go to Klaas Andries de Graaf for translating the abstract of this thesis into Dutch. I am grateful to Mojtaba Shahin for our collaboration on architectural knowledge.

I have been proud of being a member of the Software Engineering and Architecture (SEARCH) research group at the University of Groningen. Many thanks go to my former and present colleagues during all these four years. Especially, I thank my best officemate Daniel Feitosa for his kind help during these years. I thank Apostolos Ampatzoglou for our nice collaboration on architectural technical debt measurement. I am grateful to Matthias Galster, Dan Tofan, Christian Manteuffel, Sara Mahdavi-Hezavehi, Apostolos, and Daniel for their careful and critical reviews of my papers. I especially thank Matthias and Dan for sharing their wonderful ideas with me and for their kind encouragement when I encountered difficulties in my research. Thanks go to Chen Yang for his help during my empirical studies. I am

grateful to Vinicius H. S. Durelli, Uwe van Heesch, Ahmad Waqas Kamal, Trosky B. Callo Arias, and Klass-Jan Stol for their friendship. Thanks go to Jan Salvador van der Ven, David Ameller, and Ramón Urias for our nice chats when we shared the office during their visits at the SEARCH group.

I am very grateful to Prof. Bing Li and Prof. Keqing He at Wuhan University, China, for their help in many ways during these years.

Thanks go to my intern and thesis students Deniz Marlali, Can Menekse, Yavuz Selim Emir, and Jeroen David van Leusen for their contributions to the tools that support my research.

I express my gratitude to Esmee Elshof, Ineke Schelhaas, Annette Korringa, Lourens Boomsma, Janieta de Jong-Schlukebir, and the International Service Desk for helping me out with lots of things, making easier my life at the University of Groningen.

Many thanks go to Fan Zhang, Shuo Zhang, Weiguo Xia, and Yanping Zhao for the good times we had together in Groningen. I thank Jiapan Guo for the pleasant conversations we had on our research, life, and future plans. Thanks also go to Yi Wan for his help with my personal affairs in Groningen when I was in China for job search. I am grateful to Lina Ye for her inspiring suggestions on my research when we met in Lille, France during the QoSA 2014. I thank Prof. Liming Zhu, Lianping Chen, Ran Mo, and Jiang Pan for our nice gathering in Montréal, Canada during the WICSA 2015.

Special thanks go to Anita Verhoeven for her friendship and hospitality. My wife and I were invited to her home for many times, and enjoyed the time she shared with us. I especially appreciate the tasty meals she served, and her kindness to show us around to enjoy the traditional festival celebrations in The Netherlands and invite us to visit many nice sites around Groningen. All of these helped me to know better about the people, the culture, and the country.

Last but not least, I specially thank my wife Hui for her wholehearted and constant love and support along the journey. I thank her for encouraging me to pursue a PhD degree at the University of Groningen and sharing her experience in doing successful research. I am thankful for the numerous happy and relaxing moments that we enjoyed together after busy work. I also thank my other family members for their support and understanding these years.

Zengyang Li
Groningen
May, 2015

Content