Managing technical debt through software metrics, refactoring and traceability
Charalampidou, Sofia

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2019

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.
8 CONCLUSIONS & FUTURE WORK

This chapter presents the conclusions of the PhD dissertation. In Section 8.1, the research questions posed in Chapter 1 are revisited and answered according to the findings of the empirical and analytical studies reported in Chapters 2 to 7. In the same section, the contributions of this dissertation, compared to the state of the art, are summarised. Finally, the perspectives for future work are described in Section 8.2.

8.1 Answers to Research Questions and Contributions

In Chapter 1 we formulated the main problem statement of this dissertation (for more details on the three sub-problems see Section 1.4.1):

Although in the literature there is a variety of approaches for managing code, design and documentation TD, these approaches suffer from various limitations:

a) In code TD, tools for identifying, prioritizing, and resolving bad smells lack in accuracy,

b) In design TD, systematic support for identifying incorrectly instantiated patterns is lacking, as well as guidance on how to refactor the design,

c) In documentation TD, we lack tools for preventing the occurrence of insufficient, incomplete or outdated requirements documentation.
To address this problem, in Chapter 1 we decomposed it into a number of research questions that were derived based on the Design Science framework, and were answered in Chapters 2 to 7. Below we answer each research question based on the solutions that we have proposed and the empirical evidence that we have presented in the aforementioned chapters. This is organised per TD type and TD activity, according to the overview presented in Section 1.4.3.

8.1.1 Code TD Identification

We first addressed the lack of accuracy in the identification of bad smells. To this end, two research questions have been formed; RQ1.a questioned which metrics can be used for the identification of long method smells, and RQ1.b the accuracy of identifying long methods using these metrics. Both questions have been answered through a case study on 1,850 java open-source methods, which empirically explored the ability of size and cohesion metrics to predict the existence and the refactoring urgency of long method occurrences (Chapter 2). Based on the results of this study, we argue that cohesion is a quality property that should be used for the identification of extract method opportunities, and subsequently for the mining of long method bad smell instances. Specifically, the results of the study suggest that one size (i.e., LOC) and four cohesion metrics (i.e., LCOM1, LCOM2, CC and COH) are capable of characterizing the need and urgency for resolving the long method bad smell, with a higher accuracy compared to the existing literature. Based on our results CC and COH present the highest precision, and compared to the existing approaches for long method or extract method identification, our cohesion-based approach shows significantly higher precision (ranging from 68% to 96%, compared to 50% in terms of complexity (Marinescu 2004) and 38-66% of size metrics (Demeyer et al. 2000). The precision of size, based on our results is 81%. Summing up, based on the analysis we have performed: if one is interested in capturing as many long methods as possible, one should prefer size or not normalized cohesion metrics; whereas if one is interested to get as fewer false positives as possible, then one should prefer normalized cohesion metrics.

8.1.2 Code TD Prioritization

Continuing with the prioritization of code TD, we formulated a research question about how to prioritize three different kinds of bad smells, in order to repay them in the most efficient order (RQ1.c). TD prioritization can be performed on two levels: (a) selecting which kind of smell to resolve, or (b) which specific instance of the
code smell to refactor first. In this RQ, we focus on (a) and we explore the priority of refactoring Long Methods, Conditional Complexity and Code Clones, for repaying code TD, by assessing the associated interest probability. As a proxy of smell interest probability we use the frequency of smell occurrences and the change proneness of the modules in which they are identified. We note that for the special case of long methods, (b) has already been discussed in Section 2.5.2, where we discuss the urgency to apply the extract method refactoring for resolving the long method smell (RQ1.b). To achieve this goal in Chapter 3 we presented a case study which was performed on 47,751 methods extracted from two well-known open source projects. The results of the case study suggest that: (a) modules in which “code smells” are concentrated are more change-prone than smell-free modules, (b) there are specific types of “code smells” that are concentrated in the most change-prone modules. Specifically, the most frequently occurring bad smells (i.e., Code Clones) are placed in the least change prone parts of the system, whereas long methods, which are the rarest have been identified in the most frequently changing ones. (c) Interest probability of code clones seems to be higher than the other two examined code smells (i.e., 4.5%-14.0% per commit, compared to 1.0%-2.0% per commit for Long Methods and of 0.5%-3.5% per commit for Conditional Complexity). To conclude, although code clones are the kind of smell with the highest interest probability, this result does not come from the identification of the smell in change prone modules, but from the frequency of its occurrence. Additionally, the long method smell appears to be placed in design hotspots (i.e., parts of the code that change very regularly), and therefore they constitute an important kind of smell in TD management, since they are associated with the most frequent generation of interest.

8.1.3 Code TD Repayment

Upon the identification of a long method smell instance that needs to be refactored, a software engineer needs to explore potential refactoring suggestions and apply the most fitting one. To this end we defined two research questions related to the repayment of code TD. RQ1.d questions how to extract long method opportunities, while RQ1.e aims at investigating the benefit of repaying code TD by applying the proposed extract method refactoring approach. Both research questions have been answered in Chapter 4, where we introduce an approach (accompanied by a tool) that aims at identifying source code chunks that collaborate to provide a specific functionality, and propose their extraction as separate methods. The accuracy of the
proposed approach has been empirically validated both in an industrial and an open-source setting. In the former case, the approach was capable of identifying functionally related statements within two industrial long methods (approx. 500 LoC each), with a recall rate of 93%. The extraction of such sets of statements to separate methods has been validated as useful by the experts participating in our case study as the results strongly suggest that the use of method body cohesion metrics for identifying Extract Method opportunities is accurate. In the latter case, based on a comparative study on open-source data, our approach ranks better in terms of accuracy when dealing with very long methods (with F-measure 23-26.9%), compared to two well-known techniques of the literature (the best of which has an F-measure of 10.7-14.2%). To assist software engineers in the prioritization of the suggested refactoring opportunities the approach ranks them based on an estimate of their fitness for extraction. The ranking has been validated in both settings, and proved to be at least moderately correlated (correlation coefficient>0.4) to experts’ opinion.

8.1.4 Design TD Identification & Repayment

To address the second limitation of the problem statement (lack of systematic support for identifying incorrectly instantiated patterns, as well as for refactoring the design), we investigated the concrete example of a design pattern (i.e. Decorator) and we set two research questions. RQ2.a focuses on which parameters can be used for identifying improper Decorator pattern instantiations that can lead to design TD. RQ2.b aims at investigating the effect of its application on design TD, and how the knowledge about when a pattern is correctly instantiated (in terms of positive effect on quality) can be used for driving the refactoring (i.e., removing or adding a pattern instance). To answer these research questions, we presented in Chapter 5 a study which proposes a theoretical model for understanding the effect of patterns on 9 high level quality attributes (i.e., Size, Inheritance, Coupling, Cohesion, Polymorphism, Messaging, Complexity, Composition, and Abstraction). In particular, we model the effect of the pattern on quality as an equation of different size instantiation parameters (e.g., number of classes, number of methods, etc.) and we discuss cut-off points (i.e., values of the parameters) which when surpassed, the application of the pattern becomes either beneficial or harmful. Next, given the values of these parameters in the current instantiation of the pattern, we investigate if it indeed constitutes TD and thus a refactoring is necessary: if the values are in the harmful side of the cut-off point then the software engineer is prompted to re-
factor to a non-pattern version. For example, the results of the study suggest that Decorator instances should not evolve through the addition of components in composite objects, in the sense that this decreases system cohesion and therefore, modularity and maintainability are weakened.

8.1.5 Documentation TD Prevention

To address the third limitation of the problem statement (the lack of tools for preventing the occurrence of insufficient, incomplete or outdated requirements documentation), three research questions were formed. The first research question aimed at investigating whether an existing artifact traceability technique can be used to prevent requirements documentation TD (RQ3.a). Although the question was focused only on requirements traceability techniques, we decided to perform a broader study investigating existing literature in the field of software artifact traceability in general. In Chapter 6 we presented a mapping study on 155 primary studies on software artifact traceability, which are empirically evaluated, without setting any further restrictions in terms of investigating a specific domain or concrete artifacts. The study aims at exploring the goals of existing approaches, as well as the empirical methods used for their evaluation. The main contributions of this mapping study are the investigation of: (a) what type of artifacts are linked through traceability approaches; (b) what are the benefits of using the proposed artifact traceability approaches; (c) how is the benefit of these approaches measured; and (d) what are the research methods used. The results of the study suggest that requirements artifacts are dominant in the traceability domain, that the research corpus focuses on the proposal of novel techniques for establishing traceability, whereas the main benefits are the improvement of software correctness and extendibility. Finally, although many studies are including some empirical validation, there are still improvements to be made, and research methods that can be used more extensively. None of the existing studies can potentially be used for preventing requirements documentation debt. However, the findings of this study suggest that integrating requirements-to-code traceability into the IDE would be a promising approach for this purpose.

Pursuing this path, RQ3.b raises the question of how to implement requirements-to-code traceability to prevent the accumulation of documentation TD. To this end we decided to implement a requirements-to-code traceability tool, which would match the requirements defined in an industrial context (i.e. being integrated with the IDE and the current development processes). Next, RQ3.c aims at investigating how the
application of the selected requirements-to-code traceability tool influences documentation TD, in an industrial context. In Chapter 7 we reported on a qualitative case study in collaboration with a small/medium software company in order to: (a) analyze the current process and identify existing TD types, (b) collect the requirements and implement a tool that aims at preventing the accumulation of documentation TD, and (c) investigate whether the tool successfully meets its goal. The proposed tool integrates requirements specifications into the IDE, and enables the real-time creation of traces between requirements and code. The results of our study confirmed the existence of TD at the requirements specification level, in the sense that all types of documentation TD (outdated, insufficient or incomplete requirements) have been identified by the stakeholders. The stakeholders have promoted the integration of requirements specification in the IDE, rendering developers as responsible for their update and maintenance. This solution was well-accepted by developers, who considered it as a viable way to prevent the accumulation of further TD. Finally, the results indicated that the developers are motivated to use the developed tool, since they feel that they can develop, maintain and utilize requirements specifications and traces as part of their daily routine. Additionally, they consider that the extra burden is negligible, and they foresee two main benefits: that the creation of links between the requirements specifications and the place in the source code where they are implemented will have a great impact in terms of understandability of the code; and that the reduction of documentation TD will reduce the required maintenance effort, since the time required for identifying the affected parts of the code can be reduced.

In Table 8.1 we summarize the main contribution of the answer to each RQ compared to the state-of-the-art, and provide a reference to the corresponding chapter.

<table>
<thead>
<tr>
<th>Research Question</th>
<th>Chapter</th>
<th>Contributions &amp; Comparison to the state-of-the-art</th>
<th>Developed Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1.a</td>
<td>Chapter 2</td>
<td>It relates: (a) a variety of cohesion metrics with the existence of long methods, and (b) cohesion metrics to the prioritization of resolving long methods.</td>
<td>SEMI</td>
</tr>
<tr>
<td>RQ1.b</td>
<td></td>
<td>It compares size / cohesion metrics, as</td>
<td></td>
</tr>
</tbody>
</table>
### Conclusions & Future Work

In Chapter 7 we reported on a qualitative case study in collaboration with a small/medium software company in order to: (a) analyze the current process and identify existing TD types, (b) collect the requirements and implement a tool that aims at preventing the accumulation of documentation TD, and (c) investigate whether the tool successfully meets its goal. The proposed tool integrates requirements specifications into the IDE, and enables the real-time creation of traces between requirements and code. The results of our study confirmed the existence of TD at the requirements specification level, in the sense that all types of documentation TD (outdated, insufficient or incomplete requirements) have been identified by the stakeholders. The stakeholders have promoted the integration of requirements specification in the IDE, rendering developers as responsible for their update and maintenance. This solution was well-accepted by developers, who considered it as a viable way to prevent the accumulation of further TD. Finally, the results indicated that the developers are motivated to use the developed tool, since they feel that they can develop, maintain and utilize requirements specifications and traces as part of their daily routine. Additionally, they consider that the extra burden is negligible, and they foresee two main benefits: that the creation of links between the requirements specifications and the place in the source code where they are implemented will have a great impact in terms of understandability of the code; and that the reduction of documentation TD will reduce the required maintenance effort, since the time required for identifying the affected parts of the code can be reduced.

In Table 8.1 we summarize the main contribution of the answer to each RQ compared to the state-of-the-art, and provide a reference to the corresponding chapter.

<table>
<thead>
<tr>
<th>Research Question</th>
<th>Chapter</th>
<th>Contributions &amp; Comparison to the state-of-the-art</th>
<th>Developed Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1.a</td>
<td></td>
<td>Developed Tools: (a) a variety of cohesion metrics with the existence of long methods and their urgency for refactoring. It provides a method of higher accuracy (precision and recall), compared to the state of the art. Finally, it is one of the few tools that perform identification of long methods, instead of extract methods opportunities, see e.g., JDeodorand (Tsantalis and Chatzigeorgiou 2011a), JExtract (Yoshida et al. 2012), etc.</td>
<td>None. Reused existing tools</td>
</tr>
<tr>
<td>RQ1.c</td>
<td>Chapter 3</td>
<td>It investigates the relationship of change proneness and the existence of code smells in the context of technical debt management. The study is the first one that combines the frequency of occurrence of bad smells and the change frequency of the involved code, in a novel metric that can be used as a proxy for smell interest probability. The term smell interest probability is introduced by our study, along with possible use case scenarios from researchers and practitioners.</td>
<td></td>
</tr>
<tr>
<td>RQ1.d</td>
<td>Chapter 4</td>
<td>It proposes using the functional relevance of source code fragments for the identification of Extract Method opportunities. The proposed approach is the first one that is empirically validated with methods of hundreds of lines of code and in an industrial setting with professional software engineers.</td>
<td>SEMI</td>
</tr>
<tr>
<td>RQ1.e</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8.2 Future Work

Based on the findings, scope and limitations of the studies carried out during the PhD, several opportunities of future work could be identified. These opportunities
are described in the following, grouped into three main directions: (a) managing code TD, (b) managing design TD, and (c) managing documentation TD.

8.2.1 Future Work for Code TDM

With respect to code TDM, the scope of this thesis was its identification, prioritisation and repayment. In this section we present future work opportunities for advancing the state of the art in regard to all three activities.

In Chapter 2 we presented an approach for identifying extract method opportunities based on the use of size and cohesion metrics on class-level. An interesting future work would be the investigation of similar approaches based on method-level cohesion metrics or the exploration of the potential use of additional size metrics (e.g., number of accessible variables in a method) to indicate the existence and prioritization of extract method opportunities. Furthermore, researchers could explore the potentially improved predictive and ranking power of approaches that combine size and cohesion metrics (e.g. by using multivariate regression models, multi-criteria methods like the analytic hierarchy process (AHP), or Bayesian networks). Another idea would be to investigate the possibility of identifying thresholds, for the six metrics presenting the highest predictive power, that when surpassed, a method can be classified as in need for extract method refactoring. Finally, future work could investigate if method-level cohesion metrics can be used for the development of feature identification algorithms. The inherent relation between lack of cohesion and the number of functionalities that a software module offers might be a promising way for exploring the field of feature extraction.

In terms of prioritisation, in Chapter 3 we presented a methodology which provides a structured way to assess the interest probability of various types of technical debt. The methodology can be reused / tailored in many ways. First, applying it to more code smells that are described in the book of Fowler et al. (1999). Applying the method to more smells would provide a holistic evaluation of code smells, and would make the results of such a study more accurate in the sense that in the current study we considered as TD-free the modules that do not involve instances of the three bad smells under investigation. Second, tailoring it to fit different levels of granularity, such as requirements, or architecture. Such an analysis would be of great importance in the sense that TD is a multi-perspective notion that spans across all development phases. Third, applying the method to more projects would increase the reliability of the presented results and could possibly unveil differ-
ences in the interest probability of smell types in projects with different characteristics (e.g., size, maturity, history, levels of quality, etc.). An interesting special case of such an extension would be the application of the proposed approach to industrial projects, checking if there are differences compared to open-source ones.

In terms of repaying code TD the study presented in Chapter 4 led to some interesting implications and future work directions. First, the benchmark created for our comparative case study can be useful both in the domain of feature location and refactorings identification, which currently lack a set of methods with identified functionalities/extraction opportunities. The provision of this benchmark will enable a fair comparison of future approaches and reduce deviations in recall and precision, caused by using different systems as objects. Second, the fact that SRP and cohesion are successfully tailored to apply at the method level, opens new research directions on how other principles can be transferred to different levels of granularity, e.g., architecture or code. Finally, the approach can be tailored to fit the identification of additional refactoring opportunities. We believe that such a tailoring constitutes an interesting future work, since different refactoring opportunities require completely different identification algorithms, checking of preconditions, ranking approaches and evaluation strategies. For example, even for refactorings of similar purpose (e.g., extract parts of the code in different levels of granularity—i.e., extract methods, extract class, etc.) the required approaches should be different: in extract class you need to investigate the clusters of methods and attributes that should be placed in the new class, whereas in the extract method you need to investigate which lines of code are functionally relevant, do not violate AST preconditions, determine the number of parameters for the new method, etc. Thus, despite the fact that in both cases a cohesion-based approach is required, the same approach cannot be directly transferred from one code smell to the other.

8.2.2 Future Work for Design TDM

Within the scope of this thesis was also addressed the management of design TD, in terms of identification and repayment. In Chapter 5 we presented a theoretical model for understanding the effect of patterns on quality, which could lead into interesting future work like: (a) empirically investigating the accuracy of the theoretical results on OSS projects, (b) replicating the study with different alternatives so as to evaluate the sensitivity of our results to various alternative designs, (c) investigating the 3rd axis of change proposed by Ng et al. (2007) (i.e. the usefulness of the number of clients, as a predictor of software quality), to confirm whether
evolution through this axis is uniform in pattern and non-pattern solutions, and (d) comparing the effect of similar parameters of different patterns (e.g., if the addition of subclasses in Bridge has a similar effect to the addition of Leafs in Decorator.

8.2.3 Future Work for Documentation TDM

Finally, the third research area in the scope of this thesis was the management of documentation TD, focusing on requirements documentation TD prevention. In Chapter 6 we presented a systematic mapping study which explores the state-of-the-art in the field of software artifact traceability. Based on the findings of the study the performance of automated approaches and the cost of manual approaches are two major concerns related to traceability approaches, and therefore extra attention to these parameters would be advised. Additionally, researchers need to ensure that usability and management of traces drive the development of their methods and tools, so as to increase the chances of industrial adoption. In Chapter 7 we presented a qualitative case study conducted in an industrial context proposing and validating a tool-based approach for preventing documentation TD during requirements engineering. From a research point of view that study can be followed up with a longitudinal, quantitative case study to collect empirical evidence on the benefits of long-term use of the developed tool in the company. Additionally, a replication in the context of a different company would be useful, so as to check the generalizability of our findings in different processes.