A Research Plan to Characterize, Evaluate, and Predict the Impacts of Behavioral Decay in Design Patterns

Derek Reimanis
Department of Computer Science
Montana State University
Bozeman, MT 59717-3880
1+ (406) 994-4780
derek.reimanis@cs.montana.edu

ABSTRACT

We propose a research plan to further the understanding of design pattern evolution. Current research into design pattern evolution focuses on the structural elements of decay, which is realized as *structural grime*. We plan to expand the current state of research by introducing the notion of *behavioral grime*, or unwanted artifacts that appear at run-time in a pattern. This form of grime may be transparent to the current analysis models. We seek to classify types of grime into taxonomy, evaluate each type in terms of impacts on technical debt and quality in the pattern and system as a whole, and predict future occurrences of behavioral grime. Studies are designed for each of these respective goals. The results of this research will further the understanding of design patterns, assisting practitioners and researchers alike.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification – *Formal Methods*; D.2.8 [**Software Engineering**]: Metrics – *product metrics*; D.2.11 [**Software Engineering**]: Software Architectures – *Patterns*

General Terms

Measurement, Design, Experimentation, Verification.

Keywords

Software Behavior, Software Architecture, Design Patterns, Formalization, Software Decay, Technical Debt

1. INTRODUCTION

Design patterns embody recurring solutions to common object-oriented problems in software development. Patterns are design decisions that are reusable, maintainable, and attempt to minimize re-design in the future [12]. However, the evolution of design patterns is controversial. The original intent of the pattern may become obscured for many reasons, including new developers contributing to a pattern, or the unforeseen changes to elements participating in the pattern. Empirical work has shown that the structure of a pattern has the potential to decay as the pattern ages [14] [15] [17] [18] [19]. Furthermore, research has shown that the structural decay of patterns results in decreased system quality and increased technical debt [8].

Although significant work has been made towards understanding design pattern structural decay, little work has been made towards understanding behavioral decay. Behavioral decay refers to the deterioration of the runtime design of a system. Behavioral decay is complementary to structural decay, yet a large gap and dearth of research is evident. The exploration of behavioral decay in design patterns will yield greater insights into the benefits and detriments of utilizing design patterns.

This paper is organized as follows: Section 2 discusses related work. Section 3 outlines the current challenges in the field, including research gaps and relevant problems. Section 4 outlines research objectives. Section 5 describes the approach. Section 6 identifies the threats to the validity of the proposed study, and section 7 provides concluding remarks.

2. BACKGROUND AND RELATED WORK

2.1 Technical Debt

Technical debt (TD) is a metaphor coined by Ward Cunningham to describe the gap between the current state of a software system and the ideal state [7]. TD captures the effects of decisions that sacrifice good design principles for on-time delivery. Many times these decisions take the form of shortcuts or workarounds in code that complete the task at hand, but at the expense of decreased quality. *Principal* and *interest* are two attributes of TD. Given a task to implement, principal refers to the cost in effort to complete the task. Interest refers to the gap between maintenance costs under ideal conditions versus conditions where maintenance is higher due to accrued debt from tasks where TD is not repaid. Effectively managing TD is multi-faceted problem, where the need to implement new features must be leveraged with the need to refactor.

Tom et al. performed a systematic literature review of the current state of TD in academic literature [29]. The study reports that many of the difficulties of managing TD are a result of poor problem definition and representative models. As an outcome of this study, Tom et al. propose a fundamental framework of TD; this work follows this framework.

Tom et al.'s framework identifies architectural technical debt (ATD) as a specific type of TD that focuses on items originating from the design or architecture of a software project. These are items such as modularity violations [30], architecture dependency issues [26], and design pattern decay [4] [14] [15] [17] [18] [19].

Several operational models for estimating TD have recently surfaced in the field [6] [13] [23] [24] [25], however no single method has surfaced as a clear better approach, possibly because they fail to capture domain specific information in a system.

2.2 Software Quality

Software quality has been categorized into a set of characteristics, each of which is composed of related sub-characteristics. The ISO-IEC 25010 Software Quality Specification formalizes a set of eight characteristics to form an abstract model for measuring quality [16]. These characteristics, or attributes, are evaluated to the extent to which a system realizes that characteristic. Several domain-agnostic quality models that realize this specification have been developed. Two quality models, QMOOD and a robust alternative QUAMOCO, have surfaced as operational quality models [2] [31].

2.3 Software Behavior

Preliminary research reveals that software behavior can be of two types; *internal* and *external*. Internal behavior refers to the interior mechanisms and API calls that occur during system runtime. Internal behaviors are not necessarily seen except at the point in time in which they are executing. In this manner, internal behaviors are more a temporary artifact that exists only for the duration of their execution. External behavior refers to the external and observable result that the system produces. These may be represented as system goals, and are the consequences of internal behaviors. That is, internal behaviors cause external behaviors.

2.4 Software Decay

Code decay is a term that refers to the case where code is "harder to change than it should be" [9]. Similarly, software decay refers to software that is more difficult to change than it should. Several types of software decay have been identified, including code smells, anti-patterns, and design pattern decay [4] [10] [17] [18] [19]. Design pattern decay refers to implementations of design patterns that gain undesired elements or lose desired elements as they evolve. In this sense, the benefits that the pattern offers are lost as its design becomes obfuscated. Studies have found that design pattern decay negatively impacts testability and understandability of systems [4] [17].

Previous work in design pattern decay has focused on the structure of patterns [8] [14] [15] [17] [18] [19]. These are realized as unwanted or missing artifacts that do not follow the structural specification of the pattern. When these artifacts obscure the implementation of a pattern while still maintaining some of the integrity of the original pattern, they are referred to as *design pattern grime*. Alternatively, when these artifacts obscure an implementation of a pattern to such an extent that the integrity of the pattern is entirely lost, they are referred to as *design pattern rot*. Empirical studies have only confirmed the existence of pattern grime.

Further work has classified the types of design pattern grime into three disjoint categories: *class grime, modular grime,* and *organizational grime* [15] [17] [18] [19]. Of these, Schanz and Izurieta expanded the modular grime category, identifying *strength, scope,* and *direction* as attributes of modular grime [27]. Additionally, Griffith and Izurieta expanded the class grime category, identifying *strength, scope,* and *direction/context* as attributes of class grime [15].

2.4.1 Design Pattern Specification

The process of identifying pattern grime consists of recognizing differences between a pattern instance and a pattern's specification. A common language used to specify patterns is the Role-Based Meta-Modeling Language (RBML) [22]. RBML is realized in the Unified Modeling Language (UML 2.0)[1] and is an abstract language that generalizes each actor in a pattern to a single common role. Depending on the type of pattern, there will be a number of possible roles. For example, the Observer pattern has a *Subject* role and an *Observer* role. Observer pattern instances have classes that fulfill both these roles.

Dae-Kyoo Kim has shown that RBML alone is not sufficient for specifying patterns because it lacks constraint templates that limit the capabilities of roles [21]. In order to combat this, the Object-Constraint Language (OCL) is used to provide necessary constraints to RBML models.

3. CURRENT RESEARCH CHALLENGES

3.1 Research Gaps

The current knowledge base of design pattern grime features only structure-based disconformities, or grime that is captured from a static snapshot of a pattern instance. This works seeks to extend the knowledge base of pattern grime by considering behavior-based disconformities, or grime that is captured during the runtime execution of a design pattern. In an effort to achieve this goal, the authors have identified the following research gaps.

1. **Characterization of Behavioral Grime**: Structural grime is incapable of capturing whether or not a design pattern is behaving as intended. A pattern instance may have no structural grime, but the runtime execution of the pattern may not match the expected runtime execution of the pattern. Cases such as this are not captured by the current knowledge base of pattern grime. This notion forms the basis for this research. Given this, the characterization of behavioral grime is a gap that needs clear definitions.
2. **Behavioral Grime Taxonomy**: To the best knowledge of the authors, no attempt has been made at categorizing the types of behavioral grime in the context of design patterns.
3. **Impacts on Quality**: Previous studies have identified the impact of structure-based grime on quality attributes, showing that testability and maintainability are negatively impacted from structural grime [15] [19]. However, no attempt has been made at quantifying the impact of behavioral grime on these quality attributes and the additional quality attributes featured in the ISO 25010 software quality specification.
4. **Impacts on Technical Debt**: Dale and Izurieta showed that the injection of modular grime into patterns increases the technical debt of the pattern [8]. No work has sought to capture the impact of behavioral grime on technical debt.
5. **Relationships between Behavioral and Structural Grime**: Several questions arise that are concerned with the relationships between behavioral and structural grime. For example: How are structural grime and behavioral grime related? Is the appearance of structural grime causal to the existence of behavioral grime? Is the reverse true? Are there cases where structural grime exists but behavioral grime does not?

[1] http://www.uml.org/

6. **Tool Support**: Currently, there is no known tool support to operationalize behavioral concepts. Implementing a tool is an important contribution to the community.
7. **Predicting Pattern Decay:** No research has looked into predicting when a pattern is prone to decaying, or even if certain patterns are more prone to decay. Bridges to these two research gaps would give valuable insight to developers regarding the implementation of patterns, and even when to be aware that a pattern might be near decaying/rotting.

3.2 Operational Gaps

A pilot study was performed, in the form of a controlled experiment; in which realizations of observer patterns were studied. Our dataset consisted of three instances of the observer pattern that we created ourselves; one instance behaved as defined, one instance featured *Subjects* that waited a significant amount of time before updating their *Observers* when their state changed, and the final instance featured *Subjects* that did NOT update their *Observers* when their state changed. These three instances exemplify cases where, respectively, (1) a pattern behaves properly, (2) a pattern behaves properly but a disharmony exists during its lifetime, and (3) a pattern behaves significantly different from its intended usage. The SonarQube [13] tool, used to estimate Technical Debt, and the inCode tool[2], used to identify design flaws, were run across the pattern instances. Neither of these tools identified a major difference between the three pattern instances, suggesting that state-of-the-art tools used to identify issues are not capable of detecting problems concerning design pattern behavior. This experiment highlights the need to explore this area further.

3.3 Proposed Contributions

To address current gaps, the following contributions are proposed:

1. The formal characterization of behavioral grime in design patterns
2. The development of taxonomy to classify behavioral grime
3. The development of empirical studies to capture the impacts of grime on TD and quality
4. The identification of patterns that are prone to behavioral grime
5. The creation of a tool that aids in the detection of behavioral grime
6. The development of a method that allows predictive capabilities for recognizing grime

3.4 IDoESE Feedback Sought

Advice on the following topics is sought:

1. **Overall Scope**: Whilst all topics presented in this paper are interesting and necessary research items, advice on the estimation of work and its feasibility is sought. For the scope of a doctoral-level degree, is this plan too ambitious? If so, what parts should be prioritized?
2. **Automation**: Currently, there is very little automation of these processes. This is a result of exploring a new area of research. To what extent should we focus on operationalizing behavioral detection and quantification?
3. **Pattern Dataset**: The only available dataset of design pattern instances is the Perceron's dataset [1]. This dataset only features instances of 10 unique pattern types, all from the Java programming language. This means that this research has limited generalizability. Is it necessary or worth the effort to look at more pattern types and/or patterns instances from other languages?

[2] https://www.intooitus.com/products/incode

4. OBJECTIVES

4.1 Research Objectives

RG1: *Investigate* design pattern instances *for the purpose of* identifying and characterizing internal and external behavioral grime *with respect to* proper pattern behavior as defined by the design pattern specification *from the perspective of* the software system *in the context of* design patterns in open source and commercial software.

RQ1.1: Does the behavior of a design pattern instance deviate from the expected behavior of that pattern type?

Rationale: This is the basic question of this research. If it is possible to identify design pattern instances where the actual behavior deviates from expected behavior, then the need to further explore this phenomenon is apparent.

RQ1.2: Do common types of behavioral grime exist within multiple instances of a single pattern type?

Rationale: If common grime types can be identified within a specific pattern, other instances of that pattern may be circumspect to the same type of grime.

RQ1.3: Do common types of behavioral grime exist across multiple instances of different pattern types?

Rationale: If common types of behavioral grime exist across different types of patterns, we will have attained some level of generalizability that applies to a larger set of pattern types.

RG2: *Express* the difference between structural and behavioral grime *for the purpose of* illustrating the importance of studying behavioral grime *with respect to* design pattern instances *from the perspective of* design pattern instances *in the context of* open source and commercial software.

RQ2.1: To what extent can patterns have both structural and behavioral grime?

Rationale: Consider the grime quadrant in Table 1. Columns indicate whether structural grime exists in a pattern, and rows indicate whether behavioral grime exists in the same pattern. Current research has identified design patterns with grime, but those patterns are constrained by cases A and B. This research needs to be expanded to discover patterns that fall in cases C and D. This will illustrate that this work is novel.

RQ2.2: Does the current knowledge base of structural grime instances include cases of behavioral grime?

Rationale: There may be behavioral grime in many of the patterns that exhibit structural grime.

RQ2.3: What is the relationship between behavioral grime and structural grime?

Table 1 -- Grime quadrant of possible grime types. For a given pattern, rows correspond to at least once instance of behavioral grime existing in the pattern, and columns correspond to at least one case of structural grime existing in the pattern.

	Structural grime does not exist	Structural grime exists
Behavioral grime does not exist	**Case A**	**Case B**
Behavioral grime exists	**Case C**	**Case D**

Rationale: Intuitively, it appears a relationship exists between behavioral and structural grime. Discovering the precise nature of this relationship will help developers understand pattern decay in the future.

RG3: *Quantify* the impact of *grime* in internal and external design pattern behavior *for the purpose of* capturing the effects on system quality and TD *with respect to* proper pattern behavior as defined by the design pattern specification *from the perspective of* the software system *in the context of* design patterns in open source and commercial software.

RQ3.1: To what extent does behavioral grime affect the quality attributes of a design pattern?

Rationale: This research question seeks to quantify the impact behavioral grime has on the quality of the pattern.

RQ3.2: Is the quality of certain types of behavioral grime worse than other types?

Rationale: This question attempts to identify the forms of behavioral grime that are worse than others.

RQ3.3: To what extent does behavioral grime affect the TD of a software project?

Rationale: In essence, TD captures the financial impact of behavioral grime. Understanding this impact is crucial for developers and project managers alike so decisions regarding release timelines or refactorings can be made.

RQ3.4: Is the TD of certain types of behavioral grime worse than other types?

Rationale: This question attempts to identify the forms of behavioral grime that are worse than others.

RQ3.5: Are the current TD estimation and quality measurement tools capable of capturing behavioral grime?

Rationale: Behavioral grime may have an impact on the TD estimate and quality of the pattern. If the current tools are not sufficient in capturing these impacts, then the tools need to be extended in order to reflect the impact.

RG4: *Investigate* the evolution of internal and external behavior in design patterns *for the purpose of* capturing trends of behavioral grime over time *with respect to* proper pattern behavior *from the perspective of* the software system *in the context of* pattern in open source and commercial software.

RQ4.1: Can common trends of behavioral grime be captured as a pattern evolves?

Rationale: This question identifies if patterns are more prone to certain behavioral grime types. If we can predict which patterns tend towards building behavioral grime, then development efforts can be more pro-active in addressing pattern evolution.

RQ4.2: Can behavioral grime be predicted?

Rationale: This question focuses on the possibility that underlying mechanisms may exist that allow us to predict when a pattern will accumulate behavioral grime in the future.

4.2 Research Metrics

Following the GQM approach [3], several metrics are identified that will be used to answer the research questions.

M1: *Structural Grime Count (SGC)* – The total amount of grime accumulated in a single pattern realization that is identified from structural models. This metric will be used to answer RQs 2-4.

M2: *Behavioral Grime Count (BGC)* -- The total amount of grime accumulated in a single pattern realization that is identified from behavioral models. This metric will be used to answer RQs 2-4.

M3: *Technical Debt Principal (TDP)* – A measure of the cost required to complete a task. This metric will be used to answer RQ 3.

M4: *Technical Debt Interest (TDI)* – A measure of differences in cost required to complete tasks under ideal conditions versus the current condition of the system. This metric will be used to answer RQ 3.

M5: *Pattern Quality (PQ)* – An aggregated measure of the eight quality characteristics featured in the ISO 25010 software quality specification [16]. Each quality characteristic is further broken down into a number of (sub)-characteristics. This metric reflects an aggregation of the (sub)-characteristics. This metric will be used to answer RQ 3.

M6: *Probability to Deviate (PD)* – The probability that a pattern will accumulate grime in the future, given its pattern type, past and current SGC, BGC, TDP, TDI, and PQ. This metric will be used to answer RQ 4.

4.3 Working Hypotheses

H1: There exist instances of behavioral grime that are not captured by current structural grime models.

H2: Common forms of behavioral grime exist within the same pattern type.

H3: Common forms of behavioral grime exist across different pattern types.

H4: Including behavioral grime in the current grime models will allow the detection of pattern rot.

H5: Quality and TD

H5.1: Behavioral grime has a negative effect on the quality of the (a) pattern realization, and (b) software system as a whole.

H5.2: Behavioral grime has a negative effect on the TD calculation of the (a) pattern realization, and (b) software system as a whole.

H6: Given the pattern type, and past and current measurements of SGC, BGC, TDP, TDI, and PQ, it is possible to predict whether a pattern will accumulate grime in the future, with a degree of uncertainty.

5. APPROACH

5.1 Data Collection

Design pattern instances will be collected across a variety of open source and commercial software systems. The Perceron's dataset features 4500 pattern instances from Java open source software systems [1]. The patterns featured in this database will be downloaded to provide an initial set of design pattern instances. Additionally, design patterns will be manually extracted from a commercial software system owned by a local firm with an established relationship.

Models of each design pattern instance will be captured using UML class diagrams and UML sequence diagrams[3]. Class diagrams capture the structural elements of the pattern instance, and sequence diagrams capture the behavioral elements of the pattern instance. Additionally, pattern specifications for each pattern type will be captured in UML class and sequence diagrams, using RBML and OCL.

The PQ, TDI, and TDP of each pattern instance will be calculated. These metrics will be calculated for both individual pattern instances and the entire software system that the pattern originates from. This data will be stored in a relational database.

5.2 Research Approach

Once the data collection process is complete, a variety of case studies and experiments will be used to answer the research questions. Juristo and Moreno's guide on experimentation in software engineering will be used to initially construct experiments [20].

RQ1.1-3 will be evaluated using a case study, wherein the taxonomy of design pattern grime will be extended to incorporate behavioral grime types. All pattern instances will be categorized according to their behavioral and structural conformance from the grime quadrant of Table 1. We will manually sort through each category, identifying design pattern violations. Violations that share similarities (OCL or RBML) will be grouped.

RQ2.1-3 will be evaluated using a case study. Conformance checking algorithms will be implemented that validate the structural conformance and behavioral conformance according to the work done by [21] [28]. All available pattern instances will be categorized into one of the four groups defined in Table 1. A binomial regression model will be fitted from the sample in order to answer RQ2.3.

RQ3.1-5 will be evaluated using a controlled experiment. Patterns will be blocked according to pattern type and then randomly selected from the available dataset. Patterns will be evaluated for TD and quality using a suite of static and dynamic analysis tools, as discussed in section 2. After measurements are recorded, forms of grime will be randomly selected and injected into patterns. After injecting, we will re-evaluate the TD and quality measurements. To analyze the data, two ANOVA tests will be utilized. RQ3.1-4 will be answered by fitting a two mean model, containing a mean for non-injected patterns and a mean for injected patterns. That is, the respective TD and quality measurements from all tools that analyzed non-injected patterns will be averaged. Respectively, the same analysis will be done for injected patterns. RQ3.5 will be answered by fitting a separate means model; that is, each quality analysis tool will have a mean. Variance will be measured over all the analysis tools, for each of non-injected and injected patterns.

RQ4.1-2 will be evaluated using an observational study. Patterns will be divided by pattern type and assessed for the existence of grime across their lifetime in terms of project releases. For each release, a record will exist documenting whether that pattern instance has grime or not. Further, an ARIMA analysis will be performed. This will give an indication into the tendencies of a pattern to collect grime as it ages.

[3] http://www.uml.org/

6. THREATS TO VALIDITY

There exist several threats to the validity of this study. Internal validity refers to the ability to recognize a causative relationship in the study, and not as a result of confounding variables. Internal validity is threatened because other design defects may exist alongside grime in a pattern; thus design defects are a confounding variable in this study. To attempt to remove the effect of design defects, we utilize a large number of pattern instances in the analysis and block across pattern type. This mitigates the chance that a design defect will affect the results of the study.

External validity refers to the ability to generalize from the results of the study. External validity is threatened because of the limited datasets of design pattern instances. To combat this threat, we have utilized the Perceron's dataset, which is the only publically available dataset of patterns that features a large number of instances (over 4500), and pattern instances from a local commercial software firm. Patterns from both these datasets are implemented in Java, and the Perceron's dataset features only open source patterns. Therefore, the ability to generalize the results is limited to the population of patterns in this study.

7. CONCLUSIONS

We have outlined the work that will result in a doctoral dissertation in hopes that we can receive feedback on the merit of this research. Research gaps are presented and studies are designed that fill them. We intend to contribute novel research that strengthens the current state of empirical software engineering.

This research is in its early stages. Currently, preliminary research has been performed, for the purpose of illustrating the research gaps. This research includes generating pattern instances and manually injecting grime into them, as described in section 3.2. Additionally, two potential forms of behavioral grime have been identified. Next steps call for the analysis of a larger number of pattern instances that expand the taxonomy of behavioral grime.

8. REFERENCES

[1] Ampatzoglou, A., Michou, O., and Stamelos, I. Building and mining a repository of design pattern instances: Practical and research benefits, *Entertainment Computing*, Volume 4, Issue 2, April 2013, Pages 131-142, ISSN 1875-9521, DOI= http://dx.doi.org/10.1016/j.entcom.2012.10.002.

[2] Bansiya, J.; Davis, C.G., A hierarchical model for object-oriented design quality assessment, *Software Engineering, IEEE Transactions on* , vol.28, no.1, pp.4,17, Jan 2002

[3] Basili, V., Caldiera, G., and Rombach, H. D. 1994. The goal question metric approach. Encyclopedia of Software Engineering. 2, 528-532. DOI=http://dx.doi.org/10.1002/0471028959.sof142

[4] Bieman, J.M., and Wang, H. 2006. *Design pattern coupling, change proneness, and change coupling: A pilot study*. Technical Report. Colorado State University.

[5] Brown, W. H., Malveau, R. C., McCornnick III, H. W., and Mowbray, T. J. 1998. *Antipatterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley & Sons, NY.

[6] Curtis, B., Sappidi, J., and Szynkarski, A. Estimating the Principal of an Application's Technical Debt, *IEEE Software*, vol. 29, no. 6, pp. 34-42, Nov.-Dec., 2012 DOI= http://doi.ieeecomputersociety.org/10.1109/MS.2012.156

[7] Cunningham, W. 1992. The WyCash portfolio management system. *SIGPLAN OOPS* Mess. 4, 2 (December 1992), 29-30. DOI=http://doi.acm.org/10.1145/157710.157715

[8] Dale, M.R., and Izurieta, C. 2014. Impacts of design pattern decay on system quality. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (ESEM '14). ACM, New York, NY, USA, Article 37, 4 pages. DOI=http://doi.acm.org/10.1145/2652524.2652560

[9] Eick, S.G., Graves, T.L., Karr, A.F., Marron, J.S., and Mockus, A. Does code decay? Assessing the evidence from change management data, *Software Engineering, IEEE Transactions on*, vol.27, no.1, pp.1-12, Jan 2001.

[10] Fowler, M., Beck, K., Brant, J., and Opdyke, W. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman, Inc., Reading, MA.

[11] France, R.B., Kim, Dae-Kyoo, Ghosh, S., and Song, E. 2004. A UML-based pattern specification technique, *Software Engineering, IEEE Transactions on*, vol.30, no.3, pp.193, 206. DOI=http://dx.doi.org/10.1109/TSE.2004.1271174

[12] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[13] Gaudin, O. Evaluate your technical debt with Sonar, *Sonar, Jun, 2009.*

[14] Griffith, I., and Izurieta, C. 2013. Design Pattern Decay: An Extended Taxonomy and Empirical Study of Grime and its Impact on Design Pattern Evolution. In *Proceedings of the 11th ACM/IEEE International Doctoral Symposium on Empirical Software Engineering and Measurements*, USA

[15] Griffith, I., and Izurieta, C. 2014. Design pattern decay: the case for class grime. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (ESEM '14). ACM, New York, NY, USA, Article 39, 4 pages. DOI=http://doi.acm.org/10.1145/2652524.2652570

[16] ISO/IEC 25010: Systems and software engineering. Systems and Software Quality Requirements and Evaluation (SQuaRE). System and software quality models, 2011.

[17] Izurieta, C., and Bieman, J. 2007. How software designs decay: A pilot study of pattern evolution. In *Proceedings of the First Symposium on Empirical Software Engineering and Measurement* (Madrid, Spain, 2007). ESEM 2007. 449-451. DOI= http://dx.doi.org/10.1109/ESEM.2007.55.

[18] Izurieta, C. 2009. *Decay and Grime Buildup in Evolving Object Oriented Design Patterns*. Ph.D. Dissertation. Colorado State University, Fort Collins, CO, USA. Advisor(s) James Bieman. AAI3385139.

[19] Izurieta, C., Bieman, J. 2013. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *J. Software Quality*. 21, 2 (Jun. 2013), 289-323. DOI= http://dx.doi.org/10.1007/s11219-012-9175-x.

[20] Juristo, N., Moreno, A.M. 2013. *Basics of Software Engineering Experimentation*. Springer, US.

[21] Kim, D. 2004. *A Meta-Modeling Approach to Specifying Patterns*, Ph.D. Dissertation. Colorado State University, Fort Collins, CO, USA. Advisor(s) Robert France.

[22] Kim, D. The Role-Based Metamodeling Language for Specifying Design Patterns. In Toufik Taibi, editor, *Design Pattern Formalization Techniques*. Idea Group Inc., 2006.

[23] Letouzey, J., Ilkiewicz, M., Managing Technical Debt with the SQALE Method, *IEEE Software*, vol. 29, no. 6, pp. 44-51, Nov.-Dec., 2012.

[24] Marinescu, R., Assessing technical debt by identifying design flaws in software systems, *IBM Journal of Research and Development* , vol.56, no.5, pp.9:1,9:13, Sept.-Oct. 2012 DOI=http://dx.doi.org/10.1147/JRD.2012.2204512

[25] Nugroho, A., Visser, J., and Kuipers, K. 2011. An empirical model of technical debt and interest. In *Proceedings of the 2nd Workshop on Managing Technical Debt* (MTD '11). ACM, New York, NY, USA, 1-8. DOI=http://doi.acm.org/10.1145/1985362.1985364

[26] Sangal, N., Jordan, E., Sinha, V., and Jackson, D. 2005. Using dependency models to manage complex software architecture. In *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (OOPSLA '05). ACM, New York, NY, USA, 167-176. DOI=http://doi.acm.org/10.1145/1094811.1094824

[27] Schanz, T., and Izurieta, C. 2010. Object oriented design pattern decay: a taxonomy. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (ESEM '10). ACM, New York, NY, USA, Article 7, 8 pages. DOI=http://doi.acm.org/10.1145/1852786.1852796

[28] Strasser, S., Frederickson, C., Fenger, K., and Izurieta, C. 2011. An automated software tool for validating design patterns. In *Proceedings of the of ISCA 24th International Conference on Computer Applications in Industry and Engineering* (HI, USA, November 16-18). CAINE'11.

[29] Tom, E., Aurum, A., and Vidgen, R. 2013. An exploration of technical debt. *J. Syst. and Softw*. 86, 6 (Jun. 2013), 1498-1516. DOI=http://dx.doi.org/10.1016/j.jss.2012.12.052.

[30] Wong, S., Cai, Y., Kim, M., and Dalton, M. 2011. Detecting software modularity violations. In *Proceedings of the 33rd International Conference on Software Engineering* (Honolulu, HI, USA, May 21-28). ICSE'11, 411-420. DOI= http://doi.acm.org/10.1145/1985793.1985850

[31] Wagner, S., Lochmann, K., Heinemann, L., Kläs, M., Trendowicz, A., Plösch, R., Seidl, A., Goeb, A., and Streit, J. 2012. The quamoco product quality modelling and assessment approach. In *Proceedings of the 34th International Conference on Software Engineering* (ICSE '12). IEEE Press, Piscataway, NJ, USA, 1133-1142.