SQUAD: Software Quality Understanding through the Analysis of Design

Foutse Khomh
Ptidej Team - GEODES Group, DIRO
University of Montreal, QC, Canada
Email: foutsekh@iro.umontreal.ca

***Abstract*—Object-oriented software quality models usually use metrics of classes and of relationships among classes to assess the quality of systems. However, software quality does not depend on classes solely: it also depends on the organisation of classes, *i.e.*, their design. Our thesis is that it is possible to understand how the design of systems affects their quality and to build quality models that take into account various design styles, in particular design patterns, antipatterns, and code smells. To demonstrate our thesis, we first analyse how playing roles in design patterns, antipatterns, and code smells impacts quality; specifically change-proneness, fault-proneness, and maintenance costs. Second, we build quality models and apply and validate them on open-source and industrial object-oriented systems to show that they allow a more precise evaluation of the quality than traditional models, like Bansiya *et al.*'s QMOOD.**

***Keywords*-Design styles, change-proneness, fault-proneness, quality models.**

I. Research Context: Software Quality

Maintenance costs during the past decades have reached more than 70% of the overall costs of object-oriented systems [1], because of many factors, such as changing software environments, changing users requirements, and the overall quality of systems [2]. One factor on which we have a control is the quality of systems; thus, being able to assess their quality could significantly help in the prediction of maintenance efforts and the reduction of maintenance costs.

Quality models link software artifacts with quality characteristics to predict quality. Many quality models exist in the literature. They focus on the internal attributes of classes (such as size, filiation, and cohesion) or, at best, of pairs thereof, and disregard their organisation. Briand and Wust, who recently surveyed quality models [3], remarked that the only quality models that attempt to link internal attributes and external characteristics are limited to fault-proneness and do not considered the systems designs. Thus, they can hardly distinguish between a well-structured system and a system with poor design, even though their respective designs are the first things that maintainers see.

II. Question

The global research question of our thesis can be stated as follows: *How to build quality models that take into account the design of systems, in particular design patterns, antipatterns, and code smells?*

We analyse the impact of various design styles and build quality models to measure the quality of object-oriented systems by taking into account both their internal attributes and their design, especially design patterns, antipatterns and the code smells.

III. Answer

There are many principles and techniques to design systems with good quality characteristics; among these, design patterns and their opposite, antipatterns and code smells, form an interesting bridge between the internal attributes of systems, external quality characteristics, and software designs, because they link internal attributes (concrete implementation of systems) and subjective quality characteristics (subjective perception on systems), such as reusability [4] and tangible quality characteristics, such as fault [5] and change proneness.

Since their popularisation in the software engineering community in 1994, design patterns have gained importance for system design and have been the subject of many studies on quality. Some authors like Venners claim that design patterns improve the quality of systems while others like Wendorff [6] suggested that their use do not always result in "good" designs. MacNatt *et al.* [7] showed that a tangled implementation of patterns impacts negatively quality. In the following, we will study in more details the impact of tangled implementation of design patterns on different quality characteristics of systems.

Antipatterns [8] are poor design choices that are conjectured in the literature to hinder object-oriented software evolution. They are opposite to design patterns [4]: they are "poor" solutions to recurring design problems. Antipatterns are composed of code smells, which are poor implementation choices.

Code smells are to antipatterns what roles are to design patterns: classes participating in an antipattern may possess one ore more of the smells defining it. Fowler's 22 code smells [9] are symptoms of antipatterns, such as Brown's 40 antipatterns [8]. Consequently and in practice, antipatterns are in-between design and implementation: they concern the

design of one or more classes, but they concretely manifest themselves in the source code as classes with specific code smells.

We analyze the impact of design patterns, antipatterns, and code smells on different quality characteristics of systems to build quality models that take them into account when measuring the systems.

IV. Method

To answer our global research question, we propose and follow a method, DEQUALITE (Design Enhanced QUALITy Evaluation), to build quality models to measure the quality of object-oriented systems by taking into account both their internal attributes and their designs.

Our method is built on previous work and our own experience. It consists of four main steps, from the detection of design patterns, design defects, and code smells, through the analysis of the impact of these design styles on quality, to the building of quality models and the validation of these models:

1. **Detection:** To improve previous detection techniques to handle the uncertainty on detection results, we have proposed a Bayesian approach for the detection of antipatterns and code smells [10], we choose Bayesian models because they can work with missing data and allow quality analysts to specify explicitly the decision process. When data is unavailable or must be adapted to a different context, an analyst can encode her judgement into the model. We have also proposed an approach to provide a ranking of the results of a design patterns detection [11].
2. **Empirical studies:** To take into account the impact of design styles in the assessment of the quality of systems, we have performed a series of empirical studies to understand the impact of design patterns, antipatterns, and code smells on the change- and issue-proneness of systems. Table I summarises the questions and their results. Details on these studies can be found in [11], [12]. A complementary study is actually underway to understand the impact of antipatterns and code smells on the maintenance costs.
3. **Building quality model:** In this step, we propose the use of naive Bayes to build a probabilistic model that takes into account all the results of the analyses performed in the second step of our method. We have already built a model, PQMOD, that takes into account design patterns. We are currently working on taking into account our results on antipatterns and code smells.
4. **Validation:** We are setting up experimental studies on various systems to validate our quality models by comparing them to traditional models, like QMOOD [14]. A validation of PQMOD and a comparison with QMOOD have been done in [15].

V. On-going Work and Forecast Completion

At this time of our thesis, we realised the first two steps of our method, the third step being in progress. Our on-going work consists on building a Bayesian network that includes the information on classes participating in design patterns, antipatterns, and code smell to assess the quality of systems. We are also studying the impact of antipatterns and code smells on maintenance costs on an industrial system. We plan to enrich our quality model with this information as well. The fourth step of our method, which consists of experimental studies for the validation of the resulting quality models, will be performed on many versions of one industrial system and also on some open-source systems, such as Rhino, ArgoUML.

VI. Related Work

We present some major work on quality models and show that none of the existing work attempts to build a quality model while considering design.

Briand and Wüst [3] present a detailed and extensive survey of quality models. They classify quality models in two categories: correlational studies and experiments. Correlational studies use univariate and multivariate analyses, while experiments use, for examples, analysis of variance between groups (ANOVA). To the best of our knowledge, none of the presented quality models attempts to assess the architectural quality of programs directly. They all use class-based metrics or metrics on pairs of classes.

Wydaeghe et al. [16] assess the quality characteristics of the architecture of an OMT editor through the study of 7 design patterns. They conclude on flexibility, modularity, reusability, and understandability of the architecture and the patterns. However, they do not link their assessment with any quality model.

Although some studies have assess some architectural characteristics of program none have attempted to build a predictive quality model.

Vokac [5] analysed the corrective maintenance of a large commercial system over three years and compared the fault rates of classes that participated in design patterns against those of classes that did not. He noticed that participating classes were less fault prone than others. He also concluded that the Observer and Singleton patterns are correlated with larger classes that could require special attention and that classes designed with the Factory Method pattern are more compact and less coupled than others and, consequently, have lower fault rates. Vokac could not find a clear tendency for the Template Method pattern. Vokac's work inspired this study, in particular the use of logistic regression to analyse the correlations between antipatterns and change-, issue-, and unhandled exception-proneness. Other studies deal

Table I
ANALYSIS OF THE IMPACT OF DESIGN STYLES.

Research Questions	Answers
What is the proportion of classes playing zero, one, or two roles in some motif(s)?	Their proportion varies from 4.02% to 30.72% [11].
What is the proportion of classes participating in an antipattern, a code smell?	The proportion for the Blob varies from 2% to 15%.
Do Design Patterns Impact Software Quality Positively?	Some patterns do not necessarily promote reusability, expandability, and understandability [13].
What are the internal characteristics of a class that are the most impacted by playing one or two roles *w.r.t.* zero role?	playing two roles has a major impact on classes when compared to playing zero or one role [11].
What are the external characteristics of a class that are the most impacted by playing one or two roles *w.r.t.* zero role?	Playing roles has a major impact on change- and issue-proneness [11].
What is the relation between antipatterns and change proneness?	There is a greater proportion of classes participating in antipatterns that change *w.r.t.* other classes [12].
What is the relation between particular kinds of antipatterns and change proneness?	For all antipatterns except LargeClass and SpaghettiCode, infected classes are significantly more change-prone than other classes [12].
What is the relation between the code smells composing antipatterns and change proneness?	Classes with code smells are in general significantly more change-prone than other classes [12].
What is the relation between antipatterns and issue proneness?	The proportion of classes participating to antipatterns and reported in issues is twice as large as the proportion of other classes [12].
What is the relation between particular kinds of antipatterns and issue proneness?	For all antipatterns except LargeClass and SpaghettiCode, infected classes are significantly more issue-prone than other classes [12].
What is the relation between the code smells composing antipatterns and issue proneness?	Four code smells are symptomatic of issues: AntiSingleton-NotClass-GlobalVariable, ComplexClass-LargeClassOnly, LazyClass- FewMethods, and LongMethod-LongMethodClass [12].

with the changeability and resilience to change of design patterns [17] and of classes playing a specific role in design patterns [18], or their impact on the maintainability of a large commercial system [6].

While some of these studies investigate the positive (and negative) impact of good design principles, *i.e.*, design patterns, on software systems, none have studied the impact of poor design choices, *i.e.*, antipatterns, on software evolution and more generally on the quality of systems.

VII. CONCLUSION

We proposed DEQUALITE, a method to build quality models that allows the measurement of the quality of object-oriented systems by taking into account their internal attributes and their designs. This method is being implemented as a tool for practitioners–developers, quality assurance personnel, and managers to assess the quality of their systems. This tool will return a list of classes more likely to change often and–or to be fault-prone than other classes and, therefore, more likely to be costly. Consequently, a tester could decide to focus on the provided list of classes because she knows that such classes have more risks. Similarly, a manager could use this quality model to assess the volume of classes with bad quality in a to-be-acquired system and, thus, narrow down her price offer and forecast the system cost-of-ownership.

ACKNOWLEDGMENTS

I am deeply grateful to my supervisor Yann-Gaël Guéhéneuc for his guidance and support.

REFERENCES

[1] R. S. Pressman, *Software Engineering – A Practitioner's Approach*, 5^{th} ed. McGraw-Hill Higher Education, November 2001. [Online]. Available: www.accu.org/bookreviews/public/reviews/s/s000005.htm

[2] K. H. Bennett and V. Rajlich, "Software maintenance and evolution: a roadmap," in *The Future of Software Engineering*, A. Finkelstein, Ed. ACM Press, 2000.

[3] L. C. Briand and J. Wüst, "Empirical studies of quality models in object-oriented systems," *Advances in Computers*, 2002.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, 1^{st} ed. Addison-Wesley, 1994.

[5] M. Vokac, "Defect frequency and design patterns: An empirical study of industrial code," pp. 904 – 917, December 2004.

[6] P. Wendorff, "Assessment of design patterns during software reengineering: Lessons learned from a large commercial project," in *Proceedings of* 5^{th} *Conference on Software Maintenance and Reengineering*, P. Sousa and J. Ebert, Eds. IEEE Computer Society Press, March 2001, pp. 77–84. [Online]. Available: http://www.computer.org/proceedings/csmr/1028/10280077abs.htm

[7] W. B. McNatt and J. M. Bieman, "Coupling of design patterns: Common practices and their benefits," in *Proceedings of the* 25^{th} *Computer Software and Applications Conference*, T. Tse, Ed. IEEE Computer Society

Press, October 2001, pp. 574–579. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=960670

[8] W. J. Brown, R. C. Malveau, W. H. Brown, H. W. McCormick III, and T. J. Mowbray, *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*, 1^{st} ed. John Wiley and Sons, March 1998. [Online]. Available: www.amazon.com/exec/obidos/tg/detail/-/0471197130/ref=ase_theantipatterngr/103-4749445-6141457

[9] M. Fowler, *Refactoring – Improving the Design of Existing Code*, 1^{st} ed. Addison-Wesley, June 1999.

[10] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui, "A Bayesian Approach for the Detection of Code and Design Smells," in *Proceedings of the* 9^{th} *International Conference on Quality Software*, D.-H. Bae and B. Choi, Eds. IEEE Computer Society Press, August 2009.

[11] F. Khomh, Y.-G. Guéhéneuc, and G. Antoniol, "Playing roles in design patterns: An empirical descriptive and analytic study," in *Proceedings of the* 25^{th} *International Conference on Software Maintenance (ICSM)*, K. Kontogiannis and T. Xie, Eds. IEEE Computer Society Press, September 2009. [Online]. Available: http://www-etud.iro.umontreal.ca/~ptidej/Publications/Documents/ICSM09.doc.pdf

[12] F. Khomh, M. D. Penta, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on software changeability," École Polytechnique de Montréal, Tech. Rep. EPM-RT-2009-02, 2009, 15 pages. [Online]. Available: http://www-etud.iro.umontreal.ca/~ptidej/Publications/Documents/Research+report+Antipatterns+Changeability+April09.doc.pdf

[13] Foutse Khomh and Y.-G. Guéhéneuc, "Do design patterns impact software quality positively?" in *Proceedings of the* 12^{th} *Conference on Software Maintenance and Reengineering (CSMR)*, C. Tjortjis and A. Winter, Eds. IEEE Computer Society Press, April 2008, short Paper. 5 pages. [Online]. Available: http://www-etud.iro.umontreal.ca/~ptidej/Publications/Documents/CSMR08.doc.pdf

[14] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, vol. 28, pp. 4–17, January 2002. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=979986

[15] F. Khomh, N. Moha, and Y.-G. Guéhéneuc, "DEQUALITE : méthode de construction de modèles de qualité prenant en compte la conception des systèmes," École Polytechnique de Montréal, Tech. Rep. EPM-RT-2009-04, avril 2009, 31 pages. [Online]. Available: http://www-etud.iro.umontreal.ca/~ptidej/Publications/Documents/Research+report+DEQUALITE+April09.doc.pdf

[16] B. Wydaeghe, K. Verschaeve, B. Michiels, B. V. Damme, E. Arckens, and V. Jonckers., "Building an OMT-editor using design patterns: An experience report," 1998. [Online]. Available: citeseer.ist.psu.edu/wydaeghe98building.html

[17] L. Aversano, G. Canfora, L. Cerulo, C. D. Grosso, and M. D. Penta, "An empirical study on the evolution of design patterns," in *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. New York NY USA: ACM Press, 2007, pp. 385–394.

[18] M. Di Penta, Luigi Cerulo, Y.-G. Guéhéneuc, and G. Antoniol, "An empirical study of the relationships between design pattern roles and class change proneness," in *Proceedings of the* 24^{th} *International Conference on Software Maintenance (ICSM)*, H. Mei and K. Wong, Eds. IEEE Computer Society Press, September–October 2008. [Online]. Available: http://www-etud.iro.umontreal.ca/~ptidej/Publications/Documents/ICSM08b.doc.pdf