SQUANER: A Framework for Monitoring the Quality of Software Systems

Nicolas Haderer
LIRMM, Université Montpellier 2, France
Email:nicolas.haderer@etud.univ-montp2.fr

Foutse Khomh
PTIDEJ Team, DIRO, Université de Montréal, QC, Canada
Email: foutsekh@iro.umontreal.ca

Giuliano Antoniol
SOCCER Lab., DGIGL, École Polytechnique de Montréal, QC, Canada
Email: giuliano.antoniol@polymtl.ca

***Abstract*—Despite the large number of quality models and publicly available quality assessment tools like PMD, Checkstyle, or FindBugs, very few studies have investigated the use of quality models by developers in their daily activities. One reason for this lack of studies is the absence of integrated environments for monitoring the evolution of software quality. We propose SQUANER (Software QUality ANalyzER), a framework for monitoring the evolution of the quality of object-oriented systems. SQUANER connects directly to the SVN of a system, extracts the source code, and perform quality evaluations and faults predictions every time a commit is made by a developer. After quality analysis, a feedback is provided to developers with instructions on how to improve their code.**

I. Introduction

Large object-oriented software systems are now pervasive in our society. They play a vital role in our everyday life and are increasingly more and more complex. Their quality is thus of major importance. Moreover, the current trend in outsourcing development and maintenance requires means to measure quality with great details and monitor its evolution. Because the code is geographically distributed and developers generally have a limited knowledge of the entire system under development, changes on parts of the code sometimes invalidate the original design of the system, causing the design to degrade and therefore negatively impacting the overall quality of the system.

Thus, being able to monitor the quality of the system and provide quick feedback to developers working on parts of a system is essential to control and improve the quality of the system. Many pioneers of software engineering like Pfleeger recommend that more time be spent to communicate the big picture of systems under development to everyone in every position: "The people working on the pieces need to know how their one piece fits into the entire architecture" [1]. Early detection of defects in code is very important to reduce software development costs. Megen and Meyerhoff [2] observed that the costs of fixing defects in systems grow dramatically when defects are found late.

In this paper, we propose SQUANER (Software QUality ANalyzER), a framework for monitoring the quality of object-oriented systems. The contribution of SQUANER comparatively to other quality assessment tools like Squale and Sonar is it continuous evaluation of the systems under development, and its non-dependance on specific technologies, like Maven: SQUANER connects directly to the SVN of a system, extracts the source code, perform design patterns, antipatterns and code smells detections, perform quality evaluations and faults predictions every time a commit is made by a developer. After quality analysis, a feedback is provided to developers with instructions on how to improve their code. Contrary to Sonar, SQUANER is not dependant on Maven, and can be extended to analyze Maven projects as well.

With its focus on early defects detection and quality evaluation, SQUANER will help reduce a substantial amount of systems' budgets and save development time [2]. In a distributed development scenario, SQUANER provides to all developers working on parts of a system, a feedback on the overall quality of the system; thus increasing their knowledge of the system and enabling an effective quality control.

With SQUANER, the research community will benefit a new tool for evolution studies. Its continuous quality assessment, design patterns and design smells detections offer the possibility to perform interesting studies such as: the analysis of system decay, the effectiveness of quality analysis tools, or their impact on developers behavior and software development practices.

II. SQUANER

SQUANER is a framework built in layers. Figure 1 gives an overview of SQUANER's architecture and Figure 2 describes the components of SQUANER analyzer.

Once a commit is made in the SVN of a project, the source code is checked out and the parser **PADL creator** is used to build a PADL model of the system. **PADL creator** is based on the meta-model PADL [3] and provide parsers for C♯, C++, and Java. From this model of the system, metrics values are computed with the component **POM** by applying each metric on each class of the model. **POM** is a framework that offers more than 60 different metrics from the literature, including class-method import and export coupling [4]; Coupling Between Objects (CBO), and Weighted Method Count (WMC) [5]; Lack of Cohesion in Methods (LCOM5) [6]; 'C' connectivity of a class [7]; numbers of new, inherited,

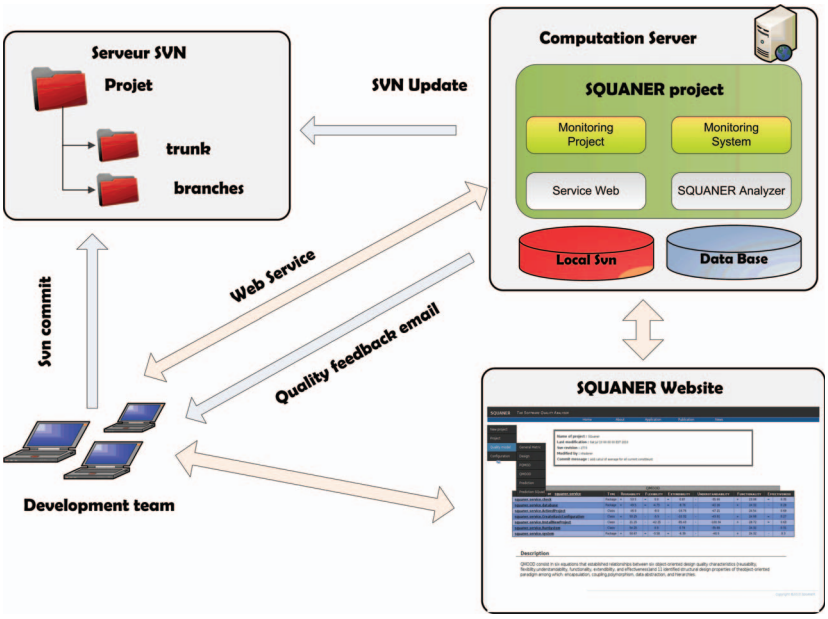


Fig. 1. Architecture of SQUANER.

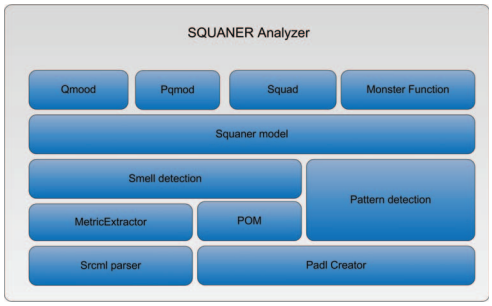


Fig. 2. SQUANER Analyzer.

and overridden methods and total number of methods [8]; Cyclomatic Complexity Metric (CC) [9]; numbers of hierarchical levels below a class and class-to-leaf depth [10].

The component **Pattern detection** is responsible for design patterns detection. This component is based on PTIDEJ [11] which implements the detection approach DeMIMA by Guéhéneuc and Antoniol [12] and the approaches by Kaczor *et al.* [13] and Ng *et al.* [14]. Combining these approaches, PTIDEJ is able to detect 13 design motifs, namely: Abstract Factory, Adapter, Command, Composite, Decorator, Factory Method, Observer, Prototype, Singleton, State, Strategy, Template Method, and Visitor.

The component **Smell detection** is responsible for antipatterns and code smells detections. This component implements the method DECOR (Defect dEtection for CORrection) by Moha *et al.* [15], to specify and identify antipatterns and code smells. DECOR is a detection approach based on a thorough domain analysis of code smells and antipatterns in the literature; analysis on which is based a domain-specific language. DECOR is able to detect the following antipatterns from [16], [17]: AntiSingleton, Blob, ClassDataShouldBePrivate (CDSBP), ComplexClass, LargeClass, LazyClass, LongMethod, LongParameterList (LPL), MessageChains, RefusedParentBequest (RPB), SpaghettiCode, SpeculativeGenerality (SG), SwissArmyKnife. These antipatterns are representative of design and implementation problems with data, complexity, size, and the features provided by a class. In addition to these antipatterns, DECOR also detects the following code smells: AbstractClass, ChildClass, ClassGlobalVariable, ClassOneMethod, ComplexClassOnly, ControllerClass, DataClass, FewMethods, FieldPrivate, FieldPublic, FunctionClass, HasChildren, LargeClass, LargeClassOnly, LongMethod, LongParameterListClass, LowCohesionOnly, ManyAttributes, MessageChainsClass, MethodNoParameter, MultipleInterface, NoInheritance, NoPolymorphism, NotAbstract, NotComplex, OneChildClass, ParentClassProvidesProtected, RareOverriding, TwoInheritance.

The component **Squaner model** implements a meta-model taking into account the history of systems as an explicit entity similarly to Hismo [18]. This meta-model builds on top of our existing meta-model PADL which describes structural information. The Squaner model can also analyse Maven projects and therefore is able to integrate and extend Sonar modules.

The component **Qmood** implements the quality model QMOOD by Bansiya and Davis [19]. QMOOD consists in six equations that established relationships between six object-oriented design quality characteristics (reusability, flexibility, understandability, functionality, extendibility, and effectiveness) and 11 identified structural design properties of the object-oriented paradigm among which: encapsulation, coupling, polymorphism, data abstraction, and hierarchies.

QMOOD is one of the most used and most referenced models in recent studies.

The component **Pqmod** implements the quality model PQ-MOD by Khomh and Guéhéneuc [20]. This quality model is composed of seven rules for the evaluation of the quality of systems by taking into account their design. These rules are defined for expandability, generality, modularity, modularity at runtime, understandability, reusability, scalability.

The component **Squad** implements two BBNs models that allow the prediction of faulty classes in systems by taking into account their metric values, their design smells and design patterns.

The component **Monitoring project** contains project monitors that are responsible for the scheduling of all the tasks involved in the analysis of systems, while the component **Monitoring system** schedules the executions of projects monitors.

III. Usage Scenario

After each developer contribution in the SVN repository, SQUANER detects the new changes automatically and downloads the new code. A series of analysis of this source code is then performed through the various components described in Section II. These analysis are performed following a configuration set beforehand by developers or quality managers. The configuration consists in choosing metrics, quality models, design patterns and design defects of interest and the frequency of their analysis.

Following the analysis, an email is sent to the development team summarizing the results, highlighting the more risky classes and providing advices on how to improve the code. With these emails, developers can have an overview of the evolution of the quality of the system since the last revision. Moreover, all the development team is aware of the impact of the new contribution on the overall quality of the system. Figure 3 presents the structure of the emails.

All the analysis results are stored in the SQUANER database to enable monitoring the evolution of the quality. SQUANER provides two means to access these information in the database: First, via a web interface at http://www.ptidej.net/research/squaner/; developers and quality analysts can browse through the packages and classes of a system and visualize detailed data on different quality characteristics. Figure 4 illustrates the visualization of results through the web site of SQUANER. Second, developers and quality analysts can use web services to schedule regular automatic quality evaluations or access the information on the quality of systems stored in the database. They can also use web services to suggest or comment on some evolution actions as a feedback to other developers; for example, when they identify problems that they cannot resolve because they do not fit their skills and knowledge.

```
Hello nhaderer,

here's a report on your recent commit change for the revision : 2773

*********************************************************************
File :squaner.service.InstallNewProject.java
*********************************************************************

Class
+--------+--------+--------+--------+--------+--------+
»           | DIT | SIX | WMC | McCabe | CBO | LCOM5 |
+------------+--------+--------+--------+--------+----|
|NEW Metrics | 1   | 0   | 56  | 25     | 22     | 0  |
+------------+--------+--------+--------+--------+----|
|OLD Metrics | 1   | 0   | 53  | 24     | 22     |0.3 |
+------------+--------+--------+--------+--------+----+

Design analysis :
This class contains 2 smells

+--------+--------+--------+--------+--------+--------+
Smell: ComplexClass
+--------+--------+--------+--------+--------+--------+
ComplexClassOnly-0 = squaner.service.InstallNewProject
McCabe-0 = 25.0
McCabe_MaxBound-0 = {McCabe_MaxBound =40.0, McCabe_UpperQuartile= 20.0}
+--------+--------+--------+--------+--------+--------+
Advice : consider restructuring this class
+--------+--------+--------+--------+--------+--------+

+--------+--------+--------+--------+--------+--------+
Smell : LongMethod
+--------+--------+--------+--------+--------+--------+
LongMethodClass-0 = squaner.service.InstallNewProject
MethodName-0 = installNewProject()
LOC-0 = 245.0
LOC_MaxBound-0 = {METHOD_LOC_MaxBound=401.5, METHOD_LOC_UpperQuartile=178.0}
+--------+--------+--------+--------+--------+--------+
Advice : consider restructiring method
+--------+--------+--------+--------+--------+--------+

Please visite the web site of Squaner project for more details :
http://www.squaner.khomh.net

Brought to you by :
SQUANER The Software QUality AnalyzER
```

Fig. 3. Example of generated quality feedback email (The e-mail was edited to fit the space)

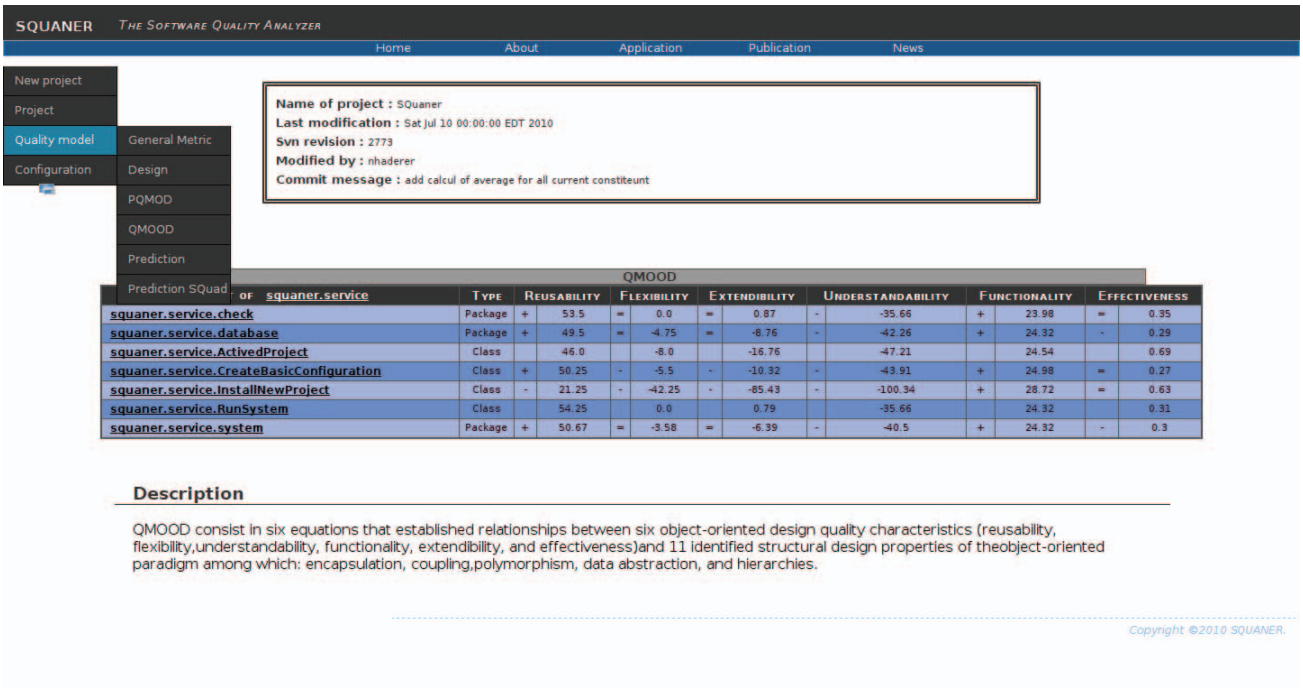
IV. Related Work

Some quality assessment tools have been proposed in the open source community, among these, Sonar[1], and Squale [2] are the most mature projects. Sonar uses various static code analysis tools such as Checkstyle, PMD, FindBugs, and Clover to extract software metrics for quality assessments and statistical visualization tools to report information on the quality of the systems. Squale implements existing standard quality models, such as ISO/IEC 9126, McCall and use a weather metaphor to report the quality of the systems. Although these tools can be apply on systems to evaluate their quality, they do not provide means to perform a continuous quality evaluation. Moreover, contrary to SQUANER which is nonintrusive as it accesses the source code directly from SVN repositories, Sonar and Squale are tools which need some training from developers and quality managers to be fully effective.

V. Conclusion

In this paper we presented SQUANER, a framework for monitoring the quality of object-oriented systems. SQUANER connects directly to the SVN of a system, extracts the source code, perform design patterns, antipatterns and code smells detections, perform quality evaluations and faults predictions

[1]http://www.sonarsource.org/
[2]http://www.squale.org/



of squaner.service	Type	Reusability		Flexibility		Extendibility		Understandability		Functionality		Effectiveness
squaner.service.check	Package	+	53.5	=	0.0	=	0.87	-	-35.66	+	23.98	=	0.35
squaner.service.database	Package	+	49.5	=	-4.75	=	-8.76	-	-42.26	+	24.32	-	0.29
squaner.service.ActivedProject	Class		46.0		-8.0		-16.76		-47.21		24.54		0.69
squaner.service.CreateBasicConfiguration	Class	+	50.25	-	-5.5	-	-10.32	-	-43.91	+	24.98	=	0.27
squaner.service.InstallNewProject	Class	-	21.25	-	-42.25	-	-85.43	-	-100.34	+	28.72	=	0.63
squaner.service.RunSystem	Class		54.25		0.0		0.79		-35.66		24.32		0.31
squaner.service.system	Package	+	50.67	=	-3.58	=	-6.39	-	-40.5	+	24.32	-	0.3

Fig. 4. Visualization of results through the web

every time a commit is made by a developer. After quality analysis, a feedback is provided to developers with instructions on how to improve their code. With SQUANER, developers and quality analysts can also use web services to schedule regular automatic quality evaluations, to suggest or comment on some evolution actions as a feedback to other developers, or to access information on the quality of systems stored in a database. SQUANER is freely available online at the following location: http://www.ptidej.net/research/squaner/.

Acknowledgements. This work has been partly funded by the NSERC Research Chairs in Software Change and Evolution and in Software Patterns and Patterns of Software.

References

[1] C. Jones, *Patterns of Software System Failure and Success*. Boston, Massachusetts: Computer Press, 1996.

[2] R. van Megen and D. B. Meyerhoff, "Costs and benefits of early defect detection: experiences from developing client server and host applications," *Software Quality Journal*, vol. 4, no. 4, pp. 247–256, 1995.

[3] Y.-G. Guéhéneuc, "PADL," since July 1999, a meta-model (and parsers) to represent and to manipulate object-oriented programs and design motifs.

[4] L. Briand, P. Devanbu, and W. Melo, "An investigation into coupling measures for C++," in *Proceedings of the* 19^{th} *International Conference on Software Engineering*, W. R. Adrion, Ed. ACM Press, May 1997, pp. 412–421.

[5] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object-oriented design," MIT Sloan School of Management, Tech. Rep. E53-315, December 1993.

[6] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*, 1^{st} ed. Prentice Hall, December 1995.

[7] M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object-oriented systems," in *Proceedings of the* 3^{rd} *International Symposium on Applied Corporate Computing*. Texas A & M University, October 1995, pp. 25–27.

[8] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: A Practical Approach*, 1^{st} ed. Prentice-Hall, July 1994.

[9] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Communications of the ACM*, vol. 32, no. 12, pp. 1415–1425, December 1989.

[10] D. P. Tegarden, S. D. Sheetz, and D. E. Monarchi, "A software complexity model of object-oriented systems," *Decision Support Systems*, vol. 13, no. 3–4, pp. 241–262, March 1995.

[11] Y.-G. Guéhéneuc, "PTIDEJ," since July 2001, a tool suite to evaluate and to enhance the quality of object-oriented programs.

[12] Y.-G. Guéhéneuc and G. Antoniol, "DeMIMA: A multi-layered framework for design pattern identification," *Transactions on Software Engineering (TSE)*, vol. 34, no. 5, pp. 667–684, September 2008, 18 pages.

[13] Olivier Kaczor, Y.-G. Guéhéneuc, and S. Hamel, "Identification of design motifs with pattern matching algorithms," *Information and Software Technology (IST)*, August 2009, 46 pages.

[14] Janice Ka-Yee Ng, Y.-G. Guéhéneuc, and G. Antoniol, "Identification of behavioral and creational design motifs through dynamic analysis," *Journal of Software Maintenance and Evolution: Research and Practice (JSME)*, November 2009, 30 pages.

[15] Naouel Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. Le Meur, "DECOR: A method for the specification and detection of code and design smells," *Transactions on Software Engineering (TSE)*, 2009, 16 pages.

[16] 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.

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

[18] T. Grba and S. Ducasse, "Modeling history to analyze software evolution," *Journal of Software Maintenance: Research and Practice (JSME)*, vol. 18, pp. 207–236, 2005.

[19] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Trans. on Software Engineering*, vol. 28, pp. 4–17, Jan. 2002.

[20] F. Khomh and Y.-G. Guéhéneuc, "Dequalite: Building design-based software quality models," in *Proceedings of the 2nd PLoP Workshop on Software Patterns and Quality (SPAQu)*, october 2008.