QualityMonitor: Visualizing and Monitoring Evolution of Software Quality

Alexandre Bergel, Marco Orellana, David Rötlisberger, Christian Palomares, Romain Charnay

Department of Computer Science (DCC),
University of Chile, Santiago, Chile

ABSTRACT

The success of a software depends very much on the ability of the software to adapt to new user requirements. However, the cost to modify a software depends very much on the quality of the software. A software of a good quality is easier to adapt and improve. A software that is poorly conceived, independently whether it is functional or not, is costly to change and adapt.

Unfortunately, quality is difficult to achieve without proper monitoring tools and methodologies. Numerous tools have been proposed to assist software development, however they are either restricted to a problem range (making it unsuitable as a global quality platform) or are considered too "academic" (identified quality problems are not always understandable by a non expert). Moreover, many of them perform well to identify problems but provide little indication on how to remove those problems.

QualityMonitor is a product used to monitor the quality of application source code. QualityMonitor innovates by delivering intuitive software visualizations to monitor quality. These "X-Ray" for software are accompanied with detailed but comprehensible indications on how to address quality deficiencies. QualityMonitor's visualizations are adjustable to the corporative programming conventions and particularities of the analyzed software, making it more flexible and agile than concurrent solutions.

In 2009, software maintenance costed 437 millions USD for the global Chilean market according to a survey sponsored by Microsoft and the Chilean government. The controlled experiments we realized in Europe and in South-America with our functional prototype suggest a significant reduction of maintenance cost.

We identified three large and prominent IT companies (Coasin[1], NIC Chile[2], Sonda[3]) that are expressing their interest in QualityMonitor. The problems these companies are facing are similar: a large software, developed over a long period, has to be maintained and enhanced with new features, however, the knowledge of its internal has evaded with changes of the development team. By drawing high level representations of software internal, QualityMonitor reengineers the knowledge of software internal, thus facilitating evolution and maintenance.

[1] `http://www.coasin.com`
[2] `http://www.nic.cl`
[3] `http://www.sonda.com/portada`

The product and services of QualityMonitor will be operated by Object Guidance, a recently created company. The international team behind Object Guidance is composed of 5 people. We are currently applying for a 90,000 USD grant, resources that will be used to shape the solutions of client requirements.

1. DIFFICULTY TO BRING QUALITY INTO SOFTWARE

Software Quality by example.

During a workshop on software quality[4] in September 2008, A. Bergel demonstrated an early prototype of Quality Monitor. A quality assurance architected working for a major car industry presented the following problem:

> *"I am in charge of the quality assurance for a large European automobile firm. The largest part of our production chain is controlled by a massive software. We have been working on this software for more than 15 years. The software has more than 2.5 millions lines of code and is developed in C (80% of the total amount of source code), C++ (19.5%) and few homemade languages (0.5%).*
>
> *Plotting the maintenance cost against the software change is an exponential curve. It is becoming impossible to precisely predict and control the cost of each new release."*

This situation is found in any software producer that has not adopted a proper solution to monitor software development quality.

Dedicated visualizations were made using an earlier version of Quality Monitor for that very problem. Visualizations pinpointed components that are relatively easy to improve and without no significant ripple effect.

IT Solution in Chile.

According to a study realized by the GECHS[5] and CETIUC[6] and sponsored by Microsoft and InnovaChile, the chilean industry of information technology comprises 1,871 companies, in which only 350 are involved in producing software and IT

[4] `http://www.systematic-paris-region.org`
[5] Sociedad Chilena de Software y Servicios, `http://www.gechs.cl`
[6] Centro de Estudios de Tecnologias e Información, `http://www.cetiuc.cl`

services. These 350 companies are developing and selling custom software and general IT solutions. The average income in 2009 for each of the 70 small and medium companies part of the GECHS is evaluated at 2.5 millions USD.

The study further shows that the necessity of IT solutions has generated an annual increase of 15% of income for the Chilean market, since 2008. The main services offered in Chile for the international market are software license (representing a share of 31% of the total income) and general project administration (17% of the total income). According to the GECHS, small and medium companies spend 52 millions USD to maintain their software, every year.

These figures emphasizes the importance of controlling the quality of what is being produced and consumed, especially in an emerging country such as Chile.

Software Quality.

Software engineering is expensive and large resources are necessary to deliver software of quality (*i.e.,* bugfree, robust, extendable with new features at a low cost). Sommerville [Som00] and Davis [Dav95] estimate that the cost of software maintenance accounts for 50% to 75% of the overall cost of a software system. Applied to the chilean industry, these figures reveals that each of the 70 surveyed companies spends between 1.25 and 1.87 millions USD. We estimate the lower bound of the cost of software maintenance in Chile to be 437 millions USD, extended to the 350 companies involved in producing software and IT services.

There is currently no global (both international and national) solutions to reduce the cost of software maintenance and controlling the quality of solution development. Symptoms of software quality problems are:

- The author of a poorly written, but functional, critical component may gain an excessive importance in the development team. We have seen numerous situations in which the departure of an engineer may endanger the whole company. By being poorly written, the component is hardly understandable by other fellows.
- In the case of a departure of a central software engineer, completely rewriting parts of the software is often the only feasible way to not meet new requirements set by customers.
- Outsourcing the development has a tendency to significantly increase the cost of software maintenance. For example, a number of critical software programs of Banco de Chile have been developed in India. As a result, the way to fix software problems is often to send experts to Chile, which incurs high cost and long delay.

These problems are well known. Solutions that are commonly employed to prevent or address them are:

- *Training.* Many Chilean Universities offer professional training programs to level up engineers and developers. These programs are usually costly and very long with a variable return on investment.
- *Metric control.* ISO-based quality models employ metrics to measure software development progress. ISO models are employed in several large European companies[7]. Similarly to the field of economy, metrics can be aggregated in a dashboard to monitor the software evolution[8]. These models are efficient to detect lack of quality in an application source code, however they are of little help on how to fix the situation since metrics output (*i.e.,* numbers) are not directly mappable to the defects.
- *Dedicated consulting.* Externalizing the control quality is a common practice[9]. This is often a punctual effort that cannot be easily used for a continuous monitoring and improvement.

[7] `http://www.qualixo.com/Squale/squale.html`

2. VISUALIZING AND MONITORING EVOLUTION OF SOFTWARE QUALITY

Objectives.

QualityMonitor is the product we envision to answer the following questions: *How to help development teams to easily maintain and continuously monitor the quality of their software programs?* and *What are the actions to be taken to improve the quality of the development?*. We address these questions using expressive visualization mechanisms. QualityMonitor produces for a given software a set of "radiographies" to immediately visualize code anomalies, sub-optimal structure and assess the test coverage. Visual pattern are associated to a given list of actions to efficiently react, leading to an improvement of the coverage.

Example of visualizations.

Figure 1, Figure 2 and Figure 3 are three visualizations obtained from an open source software. These visualizations are called *test blueprint*, and they show the test coverage of the analyzed software.

Figure 1 illustrates the principle of the visualization. Each large box is a class of the system. Class inheritance is indicated with edges. A superclass is above its subclasses. Each small inner box is a method. Invocation between methods is indicated with edges. An untested method has a red border. The height of a method is proportional to its complexity. The width says how many methods invoke the method. The gray intensity says how many times a method has been invoked by the tests (dark = tested many times).

The figure indicates that the classes `MOAbstractGraphLayout` and `C3` are relatively well tested: all but one of its methods are tested. The class contains some tall and dark methods, indicating the methods have been executed many times and in many different situations. Not all of the subclasses are well covered. For example, `C1` is not covered at all. `C2` contains 3 non-tested methods. One of them is tall, which indicates its complexity. Leaving a complex and untested software component is a risk for the general health of the application.

Figure 2 shows the evolution of the class `MOTreeMapLayout`. In the version 2.2 of the software, only three of its methods are covered. The presence of red methods indicated where to focus the testing effort. Gradually, the class went from a coverage of 27.27% to 100%, in Version 2.5.

[8] `http://www.castsoftware.com`

[9] Many companies offers service for testing and assessing the quality of architecture, *e.g.,* `http://www.americaxxi.cl` and `http://www.lattix.com`

The visualization helps assessing the inherent complexity of a software. The above part of Figure 3 represents a central class of the software. It contains many methods, with many dependencies between them. The below part represents an improvement of the class: many methods have been removed (especially obsolete, unused and dead code), leading to a global improvement of the class. The Version 2.17 of the class contains many less methods and dependencies.

To keep this proposal under the imposed space constraint, only the *test blueprint* visualization is here presented. Many more visualizations are available however[10].

Remedy action.

Visualizations identify deficiencies in the application source code. Each visualization comes with a set of actions to remedy the identified quality problems. Consider the example given in Figure 1 and Figure 3. Uncovered components are indicated in red. In that case, new tests may simply be written and the red boxes will be turned gray.

The visualization plays here an important role. Instead of simply saying whether a component is covered or not, the visualization gives the *context* in which the component has to be considered. This is important piece of information to decide whether or not the component is worth testing. Not all the components requires the same effort to test and not all of them are worth testing. Quality Monitor offers a set of interactions to drill down from the visualization to the source code, and tools to precisely identify components visually represented. Another benefit of this visualization is to monitor the reduction of complexity (Figure 3).

Related work.

Software quality has attracted the interested of numerous companies, including major software producers. Numerous tools have been produced to assist developers to control the quality of their code.

Microsoft produced Pex and Visio. Pex[11] automatically generate unit test with a high coverage. It uses a sophisticated mathematical engine to generate unit tests. Pex and the visualization given in Figure 1 address the same problem: increasing the test coverage. They however diverge on the way how to achieve this. Pex produces executable and ready to use unit tests for a given program input. Pex however does not determine which components are already tested. It is also ineffective to reduce internal software complexity.

Visio[12] intensively uses visual diagrams for data and control flow. It offers sophisticated capabilities to use a data base as input and Visio drives it along visual diagrams. Visio is advertised as a diagram tool constructor and as a way to quickly and easily manipulate data bases. Its focus is rather different than what Quality Monitor proposes. For example, Visio cannot identify complex software components therefore it cannot help reducing this complexity.

Fortify[13] is "a suite of tightly integrated solutions for identifying, prioritizing, and fixing security vulnerabilities in software. It automates key processes of developing and deploying secure applications." Fortify processes source code and apply some rules that checks proper usage of APIs, programming conventions and security. It also enables dynamic trace analysis. Fortify is about identifying and resorbing vulnerabilities. QualityMonitor has a different and complementary focus, which is the monitoring the quality. The primary focus addressed by Fortify is "how to make software less vulnerable", Quality Monitor focuses on "how to make software easier and less costly to maintain".

It is well known that monitoring the evolution of software structure is important for the maintainability of it. Lattix[14] is a tool that graphically represents dependency cycles among software components. It uses a sophisticated "dependency structural matrix" to visually represent dependencies and their anomalies. The objective of Lattix is about identifying wrong dependencies. This is indeed within the range of Quality Monitor. Quality Monitor solves this very problem by using a different visual presentation (a graph instead of a matrix). In addition, Quality Monitor enables visualization to be tailored for a particular application, according to the culture and convention adopted in the programming style.

Robust prototype.

These visualizations have been obtained using the Moose platform. Moose[15] is an open-source software analysis platform. Moose is the result of an international effort, in which the University of Chile, INRIA and the University of Bern are the principal actors. Moose is a robust prototype, principally used for research purpose. Moose has been in a constant development for more than 12 years. Based on numerous controlled experiments, we judge that Moose has now reached a level of maturity to conduct analysis on industrial and commercial software.

Moose has been employed to analyze the software of several large companies in France (Renault, Airbus, France Telecom), Argentina (Caesar Systems) and Chile (NIC Chile). Even if these analyzes where successful (*i.e.,* problems were identified and plan for addressing them were proposed), Moose remains a research prototype and not a product: documentation is missing and its implementation detached from practical considerations. Currently, conducting an analysis requires many technical steps, which represents a cost that a company cannot easily afford according to the experience we gained when realizing controlled experiments.

The purpose of this Intel Challenge proposal is to give the necessary resources to build QualityMonitor, at the top of Moose. QualityMonitor will offer a web interface to easily upload software source code and generate detailed report.

QualityMonitor is a stand-alone application, independent from any programming environment. The reason for keeping quality analyses away from production environment stems from the necessity to have different actors for producing software and assessing their quality.

QualityMonitor and quality certification.

Our primary target is the Chilean market as a short term penetration. The chilean software industry is particularly avid to quality certification in order to gain credibility. This is particularly relevant when producing software system for governmental agencies, military and exporting outside national boundaries.

[10] `http://www.moosetechnology.org/docs/visualhall`
[11] `http://research.microsoft.com/en-us/projects/pex`
[12] `http://office.microsoft.com/en-us/visio`
[13] `https://www.fortify.com/products/fortify360/index.html`
[14] `http://www.lattix.com`
[15] `http://www.moosetechnology.org`

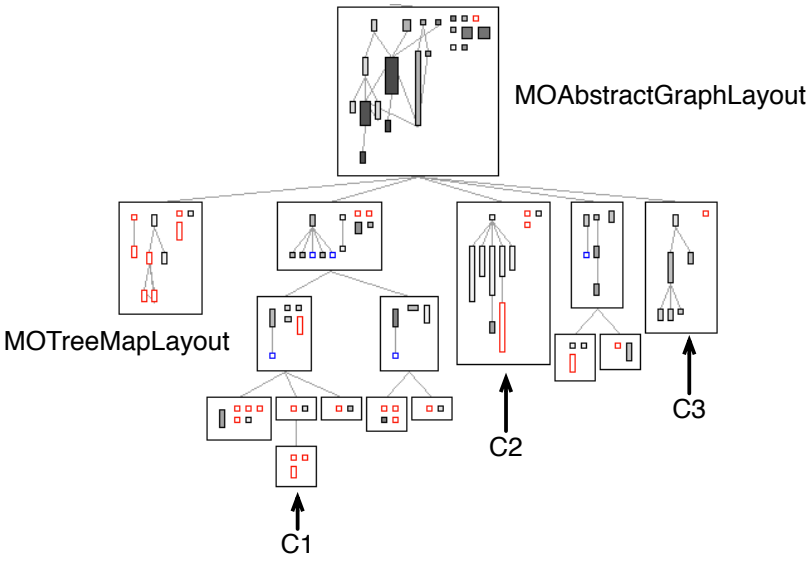

Figure 1: Example of test coverage visualization

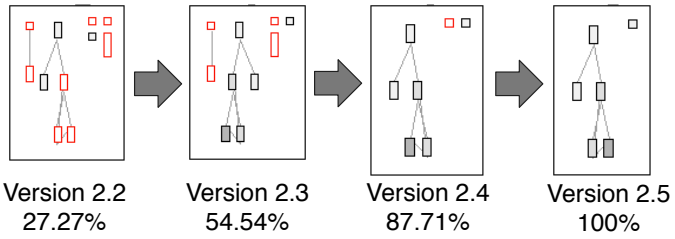

Figure 2: Test coverage evolution

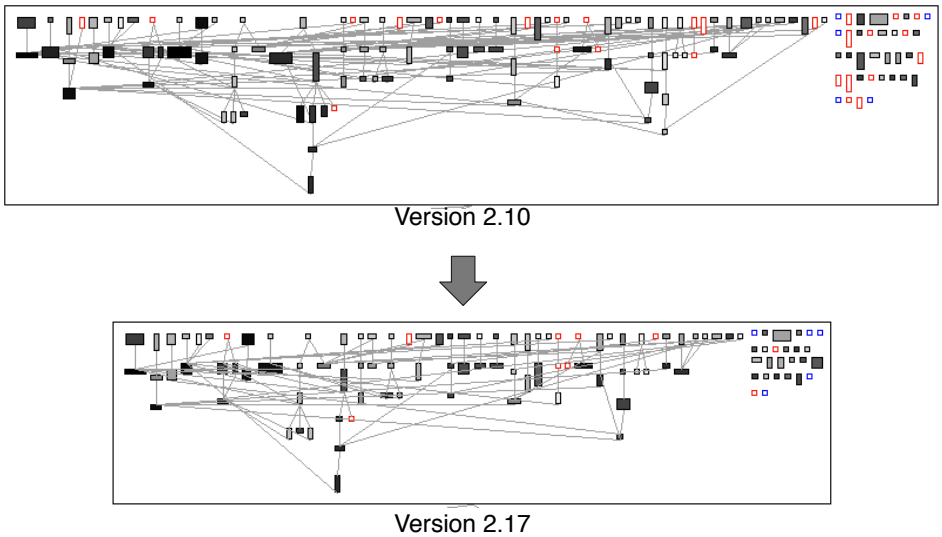

Figure 3: Complexity reduction

The *Capability Maturity Model Integration* (CMMI) is a process development approach for companies producing software systems. Obtaining a CMMI level 2 or higher is often the ambition of chilean companies who wants to reach new market segments.

Associating QualityMonitor to the CMMI certification is a perspective we started to investigate. The unique chilean company to deliver CMMI certification is America XXI[16]. The Quality Monitor team visited America XII twice, in Oct 2009 and Apr 2011. The executive direction of America XXI welcomes Quality Monitor and they expressed their interest in collaborating. We plan to visit them again in February 2012, after the propulsion of Quality Monitor. Their interest in Quality Monitor stems from their weakness to analyze application source code. Most of their evaluation effort is concentrated on the process and the formalization of the requirements. Our expertise complements well theirs.

3. BUSINESS MODEL CANVAS

We present the business model canvas[OP10] associated to QualityMonitor.

Customer segment: companies developing software, either for themselves or for tierce clients.

Value proposition: (i) controlling the quality of software; (ii) increase the performance of the software; (iii) reduce the dependency between engineers and the produced software; (iv) real-time monitoring of the software development.

Channels: (i) The QualityMonitor product will be distributed via the Web (software to be analyze is uploaded on a website) or (ii) installed onsite. (iii) Adequate capacitation will then be offered to addressing defaults found by QualityMonitor in the analyzed software applications.

Customer relationships: personal assistance and product sell.

Revenue stream: per use, license, subscription.

Key resources: (i) qualified human capital (excellent software programmers and engineers are essential for the business since they will have to report detailed analysis); (ii) non-disclosure agreement of the software analyses.

Key activities: analyzing software and proposing improvement to remove quality-associated default. Improvements will have to be quickly proposed to the clients.

Key partnerships: (i) Community around the Pharo programming language[17] (Pharo is used to build Moose); (ii) Community around the Moose software analysis platform.

Cost structure: (i) proposition for a premium-level quality; (ii) motivation for the value of the service; (iii) principal cost associated to the human capital.

4. TIMELINE AND BUDGET

We envision the following milestones:

- August 2011 - Creation of Object Guidance
- October 2011 - Government sponsored 90,000 USD grant, *Capital Semilla*. Two full time engineers will then be enrolled.
- February 2012 - Implantation of Quality Monitor within two of our customers
- June 2012 - Online services publicly offered

Engineers will be selected from the pool of students Object Guidance will interact with (e.g., bachelor and master internship).

The average salary in Chile is about 1,800 USD per month for an experienced programmers. We plan an initial investment of 6,000 USD for the necessary infrastructure for software development (two laptops and one server). Local and internet connection will be provided by the University of Chile.

5. OBJECT GUIDANCE TEAM

Object Guidance will be created in August 2011 as a stock-based company.

The current Team involved in the creation of Object Guidance is:

- Prof. Dr. Alexandre Bergel – PhD University of Bern, Switzerland. Assistant Professor University of Chile. Expert in Software Quality.
  `http://bergel.eu`
- Dr. David Rötlisberger – PhD University of Bern, Switzerland. Postdoctoral in the Pleiad laboratory at the University of Chile. Expert in Software Quality.
  `http://www.droethlisberger.ch`
- Marco Orellana Fuenzalida – Civil Engineer in industry and Computer Science. University of Chile. Currently in charge of the business plan of Quality Monitor.
- Christian Palomares – Civil Engineer in Computer Science. University of Chile. Experimented programmer.
- Romain Charnay – Executive at the Novos[18] incubador

6. REFERENCES

[Dav95] Alan Mark Davis. *201 Principles of Software Development*. McGraw-Hill, 1995.

[OP10] Alexander Osterwalder and Yves Pigneur. *Business Model Generation*. Wiley, 2010.

[Som00] Ian Sommerville. *Software Engineering*. Addison Wesley, sixth edition, 2000.

[16] `http://www.americaxxi.cl`

[17] `http://www.pharo-project.org`

[18] `http://novos.cl`