Experiences in Adapting a Source Code-Based Quality Assessment Technology

János Pántos*, Árpád Beszédes*, Pál Gyenizse† and Tibor Gyimóthy*

* University of Szeged, Department of Software Engineering
Árpád tér 2., H-6720 Szeged, Hungary, +36 62 544145
{pantos,beszedes,gyimi}@inf.u-szeged.hu

† GriffSoft Informatics Plc.
Thököly út 101., H-6726 Szeged, Hungary, +36 62 549100
pal.gyenizse@griffsoft.hu

Abstract

Testing-based software quality assurance often does not provide an appropriate level of efficiency and reliability. To aid this problem, different kinds of static verification techniques can be applied, like code metrics and code inspection. Many quality assessment methods that are based on static source code analysis has already been proposed, yet these can be used is particular industrial environment – in which often proprietary programming languages are used – only after appropriate adaptation. This paper presents experiences in adapting an existing technology and tools suitable for quality assessment based on source code analysis. The technology has demonstrated its success and usability in industrial environment; being capable of comprehensive and continuous quality monitoring of large and complex software systems involving proprietary technologies.

1. Introduction

The quality assessment of industry-applied software is based in most cases merely on continuous testing, which is quite expensive and does not provide adequate information on quality in many situations. Through quality indicators obtainable from the source code (using different code metrics and checking compliance with design and coding rules for instance), an appropriate approximation of the quality and maintainability of the software can be drawn [2, 8]. These quality assessments can in many cases valuably complement the testing-based verification.

The tools and results presented were obtained through the cooperation of the Department of Software Engineering of the University of Szeged, FrontEndART Software Ltd. [5], the GriffSoft Informatics Plc. [7], developer of the Forrás-SQL technology, and a financial service provider company using the technology. The subject of the analysis was a large and complex mission-critical subsystem of the IT infrastructure of the company. The financial company has recognized that their testing methodology is too expensive and relatively low in efficiency; however, they found no complete quality assessment solution designed for Forrás-SQL available on the market.

For this reason we applied methods for assessing quality features already available in the literature, but appropriate modifications and additions were necessary. During the cooperation among the partners, first a comprehensive quality assessment report has been prepared, and then continuous monitoring of the quality indicators has been established.

Section 2 elaborates on the main steps of our methodology and the adaptations performed. Section 3 presents the results and the recommendations based on them to improve software quality. Finally, Section 4 introduces our conclusions and plans for the future.

2. Quality assessment

We utilize the results of source code-based analysis on two levels. First, we give a comprehensive list of the actual problems that are based on certain *basic source code measurements*. Second, with the help of the basic measurements we make estimations on the *higher-level quality features* of the system, such as maintainability, modifiability, reusability, reliability and testability.

In addition, we provide the comparison of the measurements with so-called *baseline values*, through which the values measured gain a relative meaning. The baseline values are determined through comprehensive analysis of a large number of other systems.

The basic measurements are made up of various directly calculable code-metrics describing, *e.g.*, the size, complexity, or coupling aspects of the system. The other type of basic measurements include design or coding problems recognizable from the source code, some of which might cause serious (runtime) errors, while others disturb availability, cause slower system operation or hinder the understandability of the system. A special type of coding problems is code duplication, that is, when a new code is made by copying from existing source code (also called *code clones*).

The Columbus technology [4] and related quality assessment methods have been used in the project, which is part of our solution successfully applied in previous projects as well (see [5] for an overview of the technology).

2.1 Adaptation

The Forrás-SQL system is an integrated administrative and management system made up of modules (subsystems) using Windows-based user interfaces and MS-SQL database management software. The modules contain programs, and the programs are an aggregate of procedures. The language is procedural, and its syntax is quite similar to that of the C programming language. SQL statements are embedded as strings sent to specific library functions.

For analyzing of the source codes and using the various tools a common model had to be established, so based on the Columbus Schema [3] we implemented the Forrás-SQL Schema. It can be used to represent various elements of the language, such as statements, expressions and variables.

Bugs and coding problems Searching for coding problems of the source is done by a tool comprising more than 15 rules. Some of the problems hinder only the readability of the code, while others may predict serious program failures. The rules are quite similar to those used by other rule checkers and extensive collections of software design rules applied to different languages [6, 9, 10].

Clone detection The location of code duplicates is carried out by the enhanced, Forrás-SQL-enabled version of the clone detector module of Columbus [1] that can be used in various programming languages (C++, C#, Java). With this tool not only the fully identical code-segments can be found but also those with almost identical structures, such as those that differ only in variable names used.

Metrics The metrics-calculating tool computes traditional metrics with necessary modifications. In addition to well-known size based metrics (LOC, NOS and NPAR) we defined some language-specific ones, such as the number of variables, parameters and objects defined. Our complexity-metrics are obtained by the adaptation of McCC and WMC metrics. CBO and COF coupling metrics are used after adaptation; in case of CBO we determine the number of couplings between the programs, and the value of COF is a measure relating to the system as a whole, derived from the CBO-values of the programs. In addition to locating code duplicates we also define clone metrics. These metrics are: CC, CCL, and CI [1].

Monitoring The results of the measurements are loaded into a database for future evaluation. The results are visualized by the Monitor subsystem [5] that is capable of, among others, making different diagrams from the results, and also of indicating the location of rule violations in the code. A timeline diagram can also be produced based on the changes in the system monitored, on which the evolution of the software can be tracked. An automatic e-mail notification can also be set on negative changes, *i.e.* relative increase and increase over baseline.

3. Experiences

Our quality assessment report included the full list of programs and procedures in which some kinds of problems were identified based on our code-quality measures. During the evaluation we also used Rigi [11] for visualization of the results. In the following we give a short summary of the results obtained.

Code checks By checking the rules we found about a thousand rule-violations in the system, the majority of which belong to the milder category mentioned in the previous section. We highlight the rules indicating value-assignments of different types to variables, and procedures returning different types of values. We found 10 code-segments in the former, and 5 in the latter category which, although do not cause runtime errors in the present system, might do so due to inconsiderate future modification.

Metrics results From among the metric values we present the more important results related to complexity. The baseline values used for comparison of these values were determined through analyzing a number of other unrelated Forrás-SQL programs comprising 1.48 million real program lines (3797 programs and 28815 procedures). Table 1 shows these baselines associated to different entities.

54.02% of the programs and 32.32% of the procedures had a higher complexity than the baseline value. 57 procedures exist in this system whose McCabe value is greater than 50, and 16 procedures have a metric value higher than 100. By reviewing these complex procedures different reasons can be identified. For instance, there are a lot of nested IF statements with the depth of 4-5, which often contain

	Programs	Procedures
LOC	507.135	66.8
lLOC	388.827	51.216
NPAR	—	0.683
NOS	—	28.286
McCC	—	7.057
WMC	53.574	—
NII	4.605	1.099
NOI	7.163	1.496
CBO	2.68	—

Table 1. Metric baselines

large and complex SQL queries. There are also big DO CASE statements with large number of branches, which can be also complicated because of the number of IF and DO WHILE statements. It is also common that there is a big DO CASE statement but the branches are not so complex (they contain sometimes just one procedure call).

The above is an illustration of the necessity to interpret the metric values with caution. By simply comparing to baselines can often be misleading, since a high cyclomatic complexity can be due to different reasons, from which only some mean a real problem, which needs to be refactored.

Clones in the system During the measurement phase the clone coverage of the system decreased from 16% to 15%. Apart from minor variances, we observed a slow but clearly decreasing tendency. The number of clone classes and clone instances were increasing in a similar rate; the former grew from 527 to 548, the latter from 1805 to 1884. By reviewing the clones we found some typical and frequently occurring cloning solutions. Some of the strings for SQL queries are constructed in the same or very similar way by using variable contents and string constants. Another typical clone is when the SQL command and its management logic is identical (*e.g.*, same condition verification). Next, some blocks in the branches of the DO CASE statement are almost identical. Finally, there are procedures which, apart from minor differences, are the same.

Naturally, eliminating all clone copies cannot be a realistic expectation and it is neither practical, nor recommended in case of smaller analogies. It is advisable, though, to restructure matching segments of long statements and procedures as new procedures. The reason for the occurrence of new clones and the disappearance of old ones has to be investigated as well, since these increase the maintenance costs of the software.

4. Conclusions and future work

Based on experiences working on this project, we are convinced that the different kinds of metrics and code checking rules available today may not have an universal interpretation in arbitrary industrial environment. These techniques need to be appropriately adapted to be able to properly assess the quality of a software in a specific environment, which may involve proprietary languages, special collaborating technologies, domain- and company-specific peculiarities and so forth.

While this paper reports on our experiences in the adaptation process, it would be as interesting to see how the end user of the technology is satisfied with the quality monitoring established. By further improving the techniques, and documenting in more detail the required quality assurance activities using the technology, a comprehensive methodology will be obtained. In particular, in the future we intend to extend the techniques to involve more testing-related measurements and to improve the accuracy of measurements using dynamic analysis information.

References

[1] T. Bakota, R. Ferenc, and T. Gyimóthy. Clone smells in software evolution. In *Proceedings of the 23nd International Conference on Software Maintenance (ICSM 2007)*, pages 24–33. IEEE Computer Society, Oct. 2–5, 2007.

[2] V. R. Basili, L. C. Briand, and W. L. Melo. A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, Oct. 1996.

[3] R. Ferenc and Á. Beszédes. Data exchange with the Columbus schema for C++. In *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering (CSMR 2002)*, pages 59–66. IEEE Computer Society, Mar. 2002.

[4] R. Ferenc, Á. Beszédes, M. Tarkiainen, and T. Gyimóthy. Columbus – reverse engineering tool and schema for C++. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2002)*, pages 172–181, Oct. 2002.

[5] Homepage of FrontEndART Ltd. `http://www.frontendart.com`.

[6] Homepage of Gimpel Software. `http://www.gimpel.com/`.

[7] Homepage of GriffSoft Informatics Plc. `http://www.griffsoft.hu/en/`.

[8] T. Gyimóthy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, 2005.

[9] S. Meyers. *Effective C++: 55 Specific Ways to Improve Your Programs and Designs, 3rd Edition*. Addison-Wesley Pub Co, May 2005.

[10] Homepage of PMD. `http://pmd.sourceforge.net/`.

[11] The Rigi web site. `http://www.rigi.csc.uvic.ca`.