Quality of Code Can Be Planned and Automatically Controlled

Yegor Bugayenko
TechnoPark Corp.
568 Ninth Street South, 202
Naples, Florida, 34102, USA
yegor@tpc2.com

Abstract

Quality of Code is an important and critical health indicator of any software development project. However, due to the complexity and ambiguousness of calculating this indicator it is rarely used in commercial contracts. As programmers are much more motivated with respect to the delivery of functionality than quality of code beneath it, they often produce low-quality code, which leads to post-delivery and maintenance problems. The proposed mechanism eliminates this lack of attention to Quality of Code. The results achieved after the implementation of the mechanism are more motivated programmers, higher project sponsor confidence and a predicted Quality of Code.

Keywords. *quality metrics, code quality, code coverage, cyclomatic complexity, continuous integration.*

1. Introduction

Quality of Code (QoC) is a measure of compliance of a set of source code constructs to pre-defined design and coding rules [4], [15]. Quality of Product is a measure of compliance of a product functionality to a specification [5], [6]. Both qualities are calculated as a relation between the number of defective elements and the total number of elements.

Quality of Product is a visible characteristic of any software system, while QoC is hidden and may become visible much later after the system is deployed and the project is finished [8]. However, the negative consequences of low QoC are much higher [17].

There are numerous standards [5], [6] and informal approaches to quality-of-product calculation, and all of them lead to these two clear and effective metrics: "bugs fixed to bugs found" and "functional requirements accepted to functional requirements specified" [6]. The majority of software development contracts, either fixed-price or cost-reimbursement, oblige software developers to deliver the product with a highly measured quality-of-product.

Quality-of-code still remains a "nice toolkit" in most software crews, used for programmers' self-analysis and self-improvement. Many projects keep this criticaly important metric out of the project sponsor's control and unbound by a contract. As a result, the project sponsor cannot impact the QoC in their product and can merely rely on programmers' professional skills and their willingness to produce a quality code.

There is a strong necessity to have a simple and transparent mechanism that will interconnect *a*) programmers, *b*) code, *c*) QoC, and *d*) project sponsors. The mechanism proposed in the article motivates programmers to produce a quality code without any special reminders from the project sponsor.

2. Automated Control Method Overview

The mechanism ties programmers' motivation to a set of objective QoC metrics, at the same time giving the same metrics to the project sponsor. Figure 2 shows that programmers commit their source code changes to the SVN [23] repository and get back regular rewards for the quality of their code. Rewards are based on the QoC metrics, calculated by the project management software (thePanel[1]). The project sponsor receives the same metrics.

The components behind the dotted line are hidden from both programmers and the project sponsor, and work automatically at each cycle of continuous integration [3].

Continuum [1] starts a new cycle regularly, triggered by new changes in the SVN repository. Maven2 [13] compiles the code, executes unit tests, and starts a static code analyzer. The JUnit-like unit testing framework [9] and the static code analyzer produce their reports in XML files. A server-side parser and transmitter (Gist[1]) collects such XML reports, solicits QoC metrics, and sends them to thePanel.

Before the implementation of the mechanism, the project sponsor, in order to get information about the QoC, had to login directly into the static code analyzer web site, or even worse, into XML reports of such an analyzer. Moreover, in order to provide a feedback to programmers about low QoC the project sponsor had to give direct orders about certain metrics or certain modules. Such a process was very time-consuming and ineffective.

1. thePanel and Gist are proprietary software products developed in TechnoPark Corp. for automation of the mechanism described in the article.

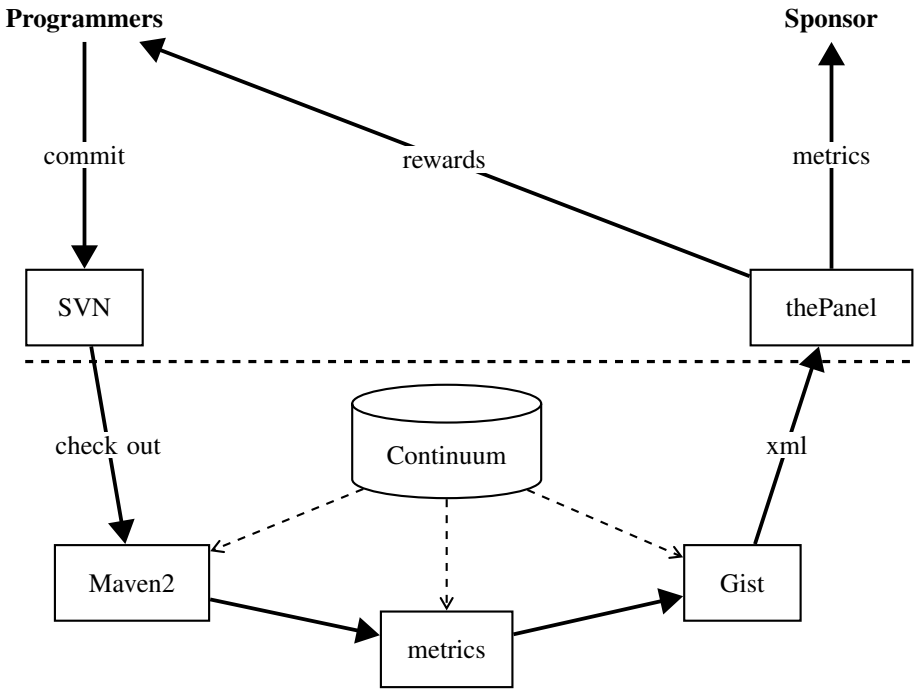

Figure 1. Endless cycle between programmers, QoC metrics, project sponsor and regular rewards given to programmers automatically by the project management software.

After the implementation of the mechanism, with automatic QoC metrics collection and bonus generation, the project sponsor no longer touches the code or reviews any reports from programmers. All that the sponsor receives is a small set of metrics. The sponsor rests assured that the metrics received influence the motivation of programmers.

3. Project Management Perspective

The proposed QoC metrics and the mechanism of their calculation from a PMBOK [18] perspective is shown in Figure 3 The mechanism could also be mapped to other project management, quality assurance, and software development standards such as CMMI [2], ISO-9001 [7], PRINCE2 [19], Rational Unified Process [20], etc.

Process "4.1 Develop Project Charter" The Project Charter and the Contract shall formally specify the list of QoC metrics and their required values. Code coverage and cyclomatic complexity [14] are the two most valuable metrics. Code coverage is measured in percentage of code constructs covered by unit tests, and it should be high. Cyclomatic complexity is an average complexity of source code, according to McCabe's formula [14], and it should be low. Other metrics could be used as well [10], [11], [16], [24].

Process "8.1 Plan Quality" During quality planning, the project manager and the project team shall identify the tools which are to be used for metrics collection [22]. In the proposed method, they are: SVN, Continuum, Maven2, JUnit, static code analyzer, Gist, thePanel. "Plan Quality" process also must identify accepted values of QoC metrics and baseline them in the Quality Management Plan.

- Version Control System — Subversion, ClearCase, AccuRev, CVS, StarTeam, Team Foundation Server, etc.
- Continuous Integrator — the software that regularly checks the status of source code in the repository and starts integration cycle when it's necessary (sometimes triggered by a user request): Apache Continuum, CruiseControl, TeamCity, Team Foundation Server, Xinc, etc.
- Builder — the software that performs routine syntax check, compilation, static library binding, unit testing and deployment in fully automated mode, without programmers attention: Maven2, Phing, MSBuild, Apache Ant, GNU make, etc.
- Testing Framework — the collection of software components that empowers the unit-testing process and generates code coverage XML reports after the testing is complete (of failed). There are many JUnit-like frameworks, while there are many other families of such instruments [25].
- Static Code Analyzer — the configurable utility that performs the analysis of the source code and produces different metrics about the results obtained. For differ-

Knowledge Area	Project Management Process Groups				
	Intiating Process Group	Planning Process Group	Executing Process Group	Monitoring & Controlling Process Group	Closing Process Group
4. Project Integration Management	**4.1 Develop Project Charter**				
5. Project Scope Management				**5.4 Verify Scope**	
6. Project Time Management					
7. Project Cost Management					
8. Project Quality Management		**8.1 Plan Quality**	**8.2 Perform Quality Assurance**	**8.3 Perform Quality Control**	
9. Project Human Resource Management			**9.3 Develop Project Team**		
10. Project Communications Management				**10.5 Report Performance**	
11. Project Risk Management					
12. Project Procurement Management		**12.1 Plan Procurements**			

Figure 2. PMBOK perspective of the described mechanism

ent languages and technologies the project team shall select appropriate tools, and configure them properly.

- Metrics Collector — the software module that is embedded into the build scenario. It should parse the XML results of the Testing Framework and the Static Code Analyzer and generate a summary report, which is suitable for the Project Management Software. We didn't find any tools available on the market for such purpose, and developed one (Gist).
- Project Management Software — the system that automates project management processes and simplifies the access to project management artifacts for all members of the project team. It is important to enable the processing of XML reports from the Metrics Collector by such a Project Management Software. The former shall be responsible for the translation of metrics into bonuses for programmers and reports for the project sponsor.

Process "4.1 Plan Procurements" As QoC requirements will be in the Contract, the project manager shall translate them to future sellers. The requirements shall be embedded into the procurement document packages.

The procurement documentation shall be written in a way that oblige sellers to deliver results with the required QoC. For some sellers (or most of them) such a requirement will sound strange since they do not have an institualized process of QoC control and prediction. However the project manager shall play a role of a trainer, if necessary, for such sellers.

Process "8.2 Perform Quality Assurance" An established mechanism of QoC metrics collection doesn't guarantee that problems won't come up during the whole project. Static code analyzers and testing frameworks become the measurement instruments for the project and shall be validated for correctness regularly.

Such quality audits should be performed by technical and management engineers together. Technical auditors shall validate the stability of used tools, review log files, update versions, etc. Management auditors shall validate whether the whole mechanism works as planned, whether the rewards really affect programmers motivation, and whether the project sponsor understands the value of the metrics regularly received.

Quality audit shall also check whether the QoC metrics are calculated on the whole set of source code files. It is a very common defect in such a process, when the QoC is analyzed using only a very limited set of source code files, for example in just one programming language. This defect shall be found by quality audit and removed immediately.

Process "9.3 Develop Project Team" The rewards produced by the project management system shall become a valuable part of project ground rules. All programmers shall understand and accept the rules. The best approach is a positive motivation delivered through regular trainings and monetary awards.

Process "5.4 Verify Scope" The project manager together with the programmers shall agree that the deliverables are accepted only when the QoC is acceptable according to the values baselined during the "Plan Quality" process.

Having such an agreement the project manager simplifies the task of keeping the QoC high enough. Moreover, only with such iterative approach to QoC control it is possible to receive source code with predicted quality.

Process "8.3 Perform Quality Control" Using different tools for QoC analysis and control (e.g., control charts, Pareto charts, or run charts) the project manager shall regularly monitor the metrics and initiate corrective actions: (in order of severity and simlicity):

- Review Motivation Policy — the easiest method that the project manager can perform, and the most effective. The vast majority of problems with performers come from defects in their motivation. The project manager shall review the ground rules in the project, discuss them with programmers and come to a situation when everybody are highly motivated to get the rewards, which the motivation policy promised to give.
- Quality Audit of Measurement Tools — maybe the instruments that validate QoC are not as effective as they should be, due to the defects in their installation configuration.
- Education Trainings — they usually help when the programmers are motivated, but don't have enough knowledge to improve QoC.
- Change Programmers — the least effective for the project and the most severe for the project team corrective action. Should be implemented only when all other corrections were implemented and didn't give a result.

Process "10.5 Report Performance" The project sponsor shall receive simple analysis of the QoC inside their regular project performance reports. It is very important to keep such analysis detail-free and non-technical. The report should not tell the sponsor about why the quality is low or where the code is not good. Such details should be omited. The report shall clearly indicate whether the QoC is within acceptable boundaries or not.

4. Experimental Application

The method was applied to five commercial projects completed by TechnoPark Corp. during 2008-2009. All of them were implemented by distributed teams, see Figure 4. Every team included 3–5 programmers working in a full-time engagement model, and over 10 part-time project participants such as like testers, designers, architects, etc.

Every team was using this method of quality assessment for the first time, and none of them were prepared or trained for it. Project sponsors (customers) were informed about the application of this model, and contracts were relatively altered to indicate the responsibility of project teams for the QoC.

We discovered that the motivation of programmers for higher QoC metrics in every project increased from the beginning. The rewards planned to be given in every project were not delivered in full to programmers; however, more than a half of them reached programmers' pockets.

Important observations showed that the post-delivery defects rate, which is a relation between the number of defects discovered during the 12 months after project completion to the number of defects discovered by the project, was much lower in projects with higher QoC (projects **C** and **D**). At the same time, lower QoC became a clear indicator of higher post-delivery defects rate (project **B**).

There was a cost of infrastructure involved, which included the amount of time in staff-hours spent for tools installation, support, and initial education of programmers. This cost was budgeted as part of project cost, and was different from one project to another. There is a strong dependency between the resources spent for the infrastructure organization and the effectiveness of the whole concept. Project **B** spent the smallest amount of time for the infrasructure, mostly because it involved more professional programmers. They claimed that they did not need to learn such a system and that they could control quality of code themselves. The numbers show that this assumption was incorrect.

After project completion, project sponsors were interviewed by the quality director of TechnoPark Corp. The feedback collected was analyzed in order to produce the overall rating. There were three questions asked in the interview (in order of importance for the overall rating):

- "Did you receive the expected quality of product?"
- "Are you going to come back with your next project?"
- "Was the project completed on time and in budget?"

It was found that QoC has a strong impact on quality of product and overall customer satisfaction, even in situations when the customer does not understand the important of QoC directly.

5. Conclusion and Future Work

The mechanism was successfully implemented in TechnoPark Corp. in a number of industry projects. The benefits received so far are:

- Programmers' motivation for better code was significantly increased;

	A	B	C	D	E
Project cost, staff-hours	1400	2150	2900	1850	2300
Project duration, weeks	15	23	29	22	28
Programmers	4	5	5	3	4
Environment, platform	PHP, MySQL, Phing, Xinc		J2EE, Oracle, Maven2, Continuum		
Metrics used	$Q_1 \in [0; 100\%]$: Code coverage, by lines of code; $Q_2 \in [1; \infty)$: Average code complexity of a class method, McCabe				
Rewarding policy	5hrs every week			7hrs bi-weekly	
Rewards calculation formula	$Q_1 > 80\% \bigwedge Q_2 < 7$				
Rewards given (portion of total capacity)	63hrs (84%)	39hrs (34%)	127hrs (88%)	70hrs (91%)	74hrs (76%)
Post-delivery defects rate (cost of defect removal)	7% 55hrs	30% 440hrs	4% 80hrs	6.5% 35hrs	unknown 40hrs+
Cost of infrastructure	40hrs	10hrs	90hrs	35hrs	50hrs+
Customer feedback	good+	average-	excellent	good	unknown

Figure 3. Experimental results obtained after the QoC method application for five industry software projects

- Project sponsors' involvement into project affairs became much higher;
- Level of post-delivery errors was twice as low;
- Customer confidence of quality was significantly improved.

Future research will answer the following question: Is it possible to produce QoC predictions for the current project using the information from other projects? How can the QoC automated analysis be integrated with risk management process? Can the risk identification and analysis be automated using the automated quality metrics?

References

[1] The Apache Software Foundation: http://continuum.apache.org.

[2] Software Engineering Institute: CMMI for Development, Version 1.2, CMU/SEI-2006-TR-008 (2006)

[3] Duvall, P., Matyas, S., Glover, A.: Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley Signature Series, USA (2007)

[4] Fenton, N., Pfleeger, S.L.: Software Metrics (2nd ed.), a rigorous and practical approach, PWS Publishing Co., Boston, MA, USA (1997)

[5] IEEE: Standard for Software Test Documentation, IEEE Std 829, USA (1998)

[6] IEEE: Recommended Practice for Software Requirements Specifications, IEEE Std 830-1998, USA (1998)

[7] ISO: Quality Management Systems, Requiremnts, Third Edition, ISO/IEC 9001:2000(E), USA (2000)

[8] Jiang, Y., Cuki, B., Menzies, T., Bartlow, N.: Comparing Design and Code Metrics for Software Quality Prediction, PROMISE'08: Proceedings of the 4th international workshop on Predictor models in software engineering, pp. 11–18, Leipzig, Germany (2008)

[9] JUnit: http://www.junit.org/.

[10] Kharb, L., Singh, R.: Complexity Metrics for Component-oriented Software Systems, SIGSOFT Softw. Eng. Notes, vol. 33, no. 2, pp. 1–3, USA (2008)

[11] Lincke, R., Lundberg, J., Löwe, W.: Comparing Software Metrics Tools, ISSTA'08: Proceedings of the 2008 international symposium on Software testing and analysis, pp. 131–142, Seattle, WA, USA (2008)

[12] Martin, R.: Agile Software Development: Principles, Patterns, and Practices (2002)

[13] The Apache Software Foundation: http://maven.apache.org.

[14] McCabe, T.J.: A Complexity Measure, ICSE'76: Proceedings of the 2nd international conference on Software engineering, p. 407, San Francisco, California, United States, IEEE Computer Society Press (1976)

[15] McConnell, S.: Code Complete, 2nd Edition, Microsoft Press, Redmond, WA (2004)

[16] Meyers, T.M., Binkley, D.: An Empirical Study of Slice-based Cohesion and Coupling Metrics, ACM Trans. Softw. Eng. Methodol., vol. 17, no. 1, pp. 1–27, ACM, New York, NY, USA (2007)

[17] Misra, S.C.: Modeling Design/Coding Factors That Drive Maintainability of Software Systems, Software Quality Control, vol. 13, no. 3, pp. 297–320, Kluwer Academic Publishers, Hingham, MA, USA (2005)

[18] Project Management Institute: A Guide to the Project Management Body of Knowledge (PMBOK Guide), Third Edition, PMI Press, 3rd edition (2004)

[19] Managing Successfull Projects with PRINCE2, Office of Government Commerce, London, UK (2005)

[20] IBM, Rational, Rational Unified Process in Rational Method Composer (2007)

[21] Scotto, M., Sillitti, A., Succi, G., Vernazza, T.: A Relational Approach to Software Metrics, SAC'04: Proceedings of the 2004 ACM symposium on Applied computing, pp. 1536–1540, Nicosia, Cyprus (2004)

[22] Sillitti, A., Russo, B., Zuliani, P., Succi, G.: Deploying, Updating, and Managing Tools for Collecting Software Metrics, QUTE-SWAP'04: Proceedings of the 2004 workshop on Quantitative techniques for software agile process, pp. 1–4, Newport Beach, California (2004)

[23] CollabNet: http://subversion.tigris.org/.

[24] Ware, M.S., Fox, C.J.: Securing Java code: Heuristics and an Evaluation of Static Analysis Tools, SAW'08: Proceedings of the 2008 workshop on Static analysis, pp. 12–21, Tucson, Arizona, ACM (2008)

[25] Yang, Q., Li, J.J., Weiss, D.: A Survey of Coverage Based Testing Tools, AST'06: Proceedings of the 2006 international workshop on Automation of software test, pp. 99–103, Shanghai, China (2006)