Development and evaluation of GQM method to improve adaptive systems

Shokhista Ergasheva
Innopolis University
Innopolis, Russia
s.ergasheva@innopolis.university

Artem Kruglov
Innopolis University
Innopolis, Russia
a.kruglov@innopolis.ru

Ilhar Shulhan
Innopolis University
Innopolis, Russia
i.shulhan@innopolis.ru

Abstract

The success of adaptive systems depends on many factors, which are not easy to measure objectively. Today, the quality of the system is of paramount importance while measuring it comes only after emerging issues. Measurement is a mechanism for feedback and evaluation of the system answering different kinds of questions related to software processes. It allows to determine the strengths and weaknesses of the system in terms of the quality of the product. However, determining the suitable approach to measure the quality can cause some difficulties. In this paper, we propose Goal-Question-Metric (GQM) approach to develop and evaluate the adaptive systems. The approach is based on the assumptions where it must first specify the goals of the system. Then they should be traced to the data defining the goals operationally. Afterwards, the framework is specified for transcribing the data referring to the defined goals. The paper provides examples of measurements types, the popular measurement tools for collecting quality metrics, GQM approach development based on the list of common goals, questions and metrics.

1 Introduction

Accurate measurement is a prerequisite for all engineering disciplines, and soft-ware development is not an exception. For many decades, engineers and re-searchers have sought to express the characteristics of software in numbers to facilitate evaluation of software quality. A large set of software quality metrics has been developed so far, and there are many tools for collecting metrics from program representations. This wide variety of tools allows the user to choose the most suitable tool, for example, depending on his circulation, tool support or price. However, this assumes that all metric tools compute/interpret/implement the same metrics in the same way. This is becoming particularly important in a market characterized by the advent of mobile systems [1–3], which pose particular challenges in the area of energy efficient computing [4-9].

In the context of the study, it is necessary to define a tool for collecting software metrics as a program that implements a set of definitions of software metrics [10]. This allows to evaluate the software system in accordance with the metrics, extracting the necessary objects from the software and providing the corresponding metric values. The factor-criterion-metric approach proposed by McCall [11] in relation to software leads to the concept of a software quality model. It combines the values of software metrics in a well-defined way with aggregated with numerical values to help qualitative analysis and evaluation. A suitable software quality model is provided by ISO 9126 [12].

But how to decide what needs to be measured in order to achieve the goal? And how to be sure that the goal is measurable? There are various approaches for determining measurable goals The Quality Function Deployment approach [13], The Goal Question Metric approach (GQM) [14] and Software Quality Metrics approach.

Goal Question Metric will be described in more detail in the next section. This is a top-down approach, which is particularly useful in agile environments [15, 16]. This means that first it is needed to set a goal, and then use one of the above methods to describe indicators that will allow to measure progress towards the goal.

2 Objectives

The aim of the work is to study existing open source components for calculating metrics and developing a model of the GQM approach for collecting software metrics [17–21].

2.1 Examining existing open source components

This section analyzes and classifies existing solutions and methods for collecting and calculating software metrics and discusses the advantages and disadvantages of invasive and non-invasive methods for collecting data, interpret and implement definitions of object-oriented software metrics in different ways[22-26].

Software measurement is an ongoing process of identifying, analyzing and collecting data about the software development process and its results in order to understand and control the process and its outcomes[27]. Furthermore, it provides important information to improve this process and its products. The main goals of collecting information are:

- project planning support;
- identification of strength and weaknesses of current processes and products;
- rationale for the adoption / refinement of methods;
- quality assessment of specific processes and products;
- assessment of progress;
- understanding and justification of the desired and undesirable changes.

Measuring software is one of the key aspects of improving and understanding the quality of the development process by collecting feedback and evaluation. To be effective, measurement must be:

- focused on specific goals;
- applied to all products, processes and life cycle resources;
- interpreted based on the characteristics and understanding of the organizational context, environment and goals;

There are three main objects that can be measured in the process of software development [28]:

- Processes - activities to the software production;
- Products - artifacts resulting from a process;
- Resources - input needed to complete the process.

Each measurement object has two types of attributes: internal and external. Internal attributes can be measured by examining the object itself, not regarding to its behavior. On the other hand, external measures can only be measured taking into account the behavior of the object.

3 Component Analysis

Following, there will be provided an overview, comparison and analysis of the most popular tools for collecting open source metrics. The review will cover the following tools:

-SonarQube, which is an open source web platform used to measure and analyze the quality of source code. Analysis of the quality of the code makes the code more reliable and more readable. SonarQube is written in Java, but it can analyze and manage code in more than 20 programming languages, including c/ c ++, PL / SQL, Cobol and others. There is also a set of plug-ins (more than50 plug-ins are available) that extend the functionality of SonarQube [29].

-UDPis an open source command line tool. It parses C ++ and Java file sand generates reports on various metrics, including lines of code and metrics proposed by Chidamber & Kemerer [30] and Henry & Kafura.

-Chidamber & Kemerer Java Metrics[31] is an open source command line tool. It computes object-oriented C& K metrics by processing the byte code of compiled Java files.
-Coverity is open source, works with C, C ++, C #, Objective-C, Java, JavaScript, the JS, Ruby, PHP, and Python node. Also, this tool supports 100compilers. Coverity provides a clear description of the root causes of code problems. Vulnerabilities detected include resource leaks, NULL pointers, improper use of the API, use of uninitialized data, memory corruption, buffer overflows, control flow, error handling, concurrency, insecure data, unsafe use of signed values and use of freed resources.
-Eclipse Metrics Plug-in 1.3.6from Frank Sauer - plug-in for calculating open source metrics and dependency analyzer for Eclipse IDE. It measures various metrics and detects cycles in modules and data type dependencies.
-Eclipse Metrics Plug-in 3.4from Lance Walton open source. It calculates various metrics during build cycles and warns through the Problems View tool of violations of the range of metrics.
-RIPS detects security vulnerabilities for PHP codes, provides an integrated code audit framework, and has open source code. RIPS tokenizes and analyzes the entire source code, finds sensitive vulnerabilities that can be corrupted by user input. RIPS is convenient to use for finding security issues beyond a limited time resource. Within a few minutes there is an opportunity to get meaningful results. It also integrates seamlessly with DevOps tools. It is possible to use a local version or SaaS version, depending on specific needs. The tool also allows you to track the security progress of the application, identify risks and correct them in advance.
-Code Compare resolves merge conflicts and deploys changes to the source code, is open source. It is possible to integrate with TFS, SVN, Git, Mercurial and Perforce. Code Compare supports C, C ++, Visual Basic, JavaScript, Java, and XML. Code Compare is for comparing and combining files and folders. This component can be used as a standalone tool or as a Visual Studio extension. When problems are detected, colored blocks appear for inserted, deleted, or modified text.
-Dependency Finder is open source. This is a toolkit for analyzing com-piled Java code. Its core is a dependency analysis application that extracts dependency graphs and extracts them to obtain useful information. This application comes as a command line tool, a Swing-based application, a web application, and Ant task suite.
If we look at the list of all indicators that can be calculated using any of the considered tools, the total number of different metrics (different in name) is about 200. After carefully reading the metric descriptions, it can be identified the similarities between metrics that have different names in different tools. The comparison between them is not always obvious, and in some cases, this is nothing more than a reasonable assumption. These indicators work with various program objects, for example, with a method, classes, packages, programs, and more. An example of one of the most common metrics are:

- DIT (depth of inheritance tree) - the maximum inheritance path from class to root class [30],
- NOC (Number of children) is the number of direct subclasses subordinate to the class in the class hierarchy and
- NOM (Number of methods) - these are methods in the class [32].

Software developers should be able to rely on tools that implement these metrics, help them with quality assessment and quality assurance tasks, enable them to quantify the quality of software, and provide the information they need as input to their decision-making and development processes. Currently, there is a large set of software metrics. But these are not the tools that were used to evaluate software metrics. To rely on scientific discussions and tests, it is safe to apply the results and use them in practice, it is necessary that all metric tools implement the proposed metrics in the way they were installed.

4 Develop an approach for collecting software metrics

Software development processes have a huge number of characteristics, and it is not always clear what needs to be measured to achieve a result. Therefore, the measurement is considered as a waste of time because of their huge costs and the lack of interpretation. Modern development methodologies attempt to maintain the development process in a structural manner and provide process metrics as a byproduct. But, as a rule, by-products reflect only a small part of the process and do not have a general picture of achieving the goals of the company due to the lack of consistency between the goals of the company and development methodologies. In addition, as the complexity of software systems increases, it becomes increasingly difficult to maintain awareness of the overall status of the project and understand the current bottlenecks in the process. One of the existing solutions to overcome this problem is the use of a visual panel that helps to understand the whole picture, presenting the appropriate metrics assembled taking into ac-count the goals. But the question of choosing metrics remains open. There are several methodologies for selecting metrics that allow to relate them to the goals of the company. One of the best-known standards is the Goal Question Metric approach [14].

4.1 GQM model

There is no universal combination of metrics or even metrics that could fully describe the real situation in an organization. Each company should independently choose a set of metrics that will give it valuable information about the effectiveness of the software development process. In fact, almost everything can be measured. However, this approach is almost useless. To be valuable, all measures must be defined, accurate and meaningful [33]. Measures cannot exist without context, turning the whole process into an aimless dimension of some-thing. Before choosing metrics, it needs to be carefully analyzed the reasons why the data is needed. The most commonly used technique for determining valuable metrics is the Goal Question Metric (GQM) paradigm proposed by Victor Basili. This approach allows companies to determine what data should be collected and how it can be interpreted. This creates a goal-based frame work for measuring software.
The method has three levels [34]:
- Conceptual level (goal): defines the object of research (products, processes or resources) and the reason for its study.
- Operational level (question): defines the questions that characterize various aspects of the measurement object (product, process, resource). Questions should be measurable objects because they create a relationship between the object of study and focus.
- Quantitative level (metric): defines a set of measurements that can be used to answer formulated questions.

The GQM + Strategic approach extends the GQM model, taking into ac-count the hierarchy of company's goals it explicitly generates the relationship between specific actions and measurement goals. As a result, the metrics defined by this method are closely related to the company's strategy and can be easily interpreted and used to improve the quality of the software product.

4.2 Development and design of GQM model

4.2.1 Defining Common Goals

The basis for this study is the predefined GQM model obtained by V.N. Zorin through a personal interview with 67 people from industry. Based on an analysis of the responses of the survey participants, the following most common goals can be determined:
- Improve the effectiveness of the effort assessment;
- Using resources in a more efficient way;
- Performing testing in a more efficient and systematic way;
- Improve the quality of the development process;
- Successful completion of projects;
- Successful completion of the project phase;

Using the described goals, you can identify the most common questions.

4.2.2 Defining Common Questions

To assess progress in improving the accuracy of the assessment, first of all, the team must understand how well it evaluates the tasks. Therefore, the main question is the actual accuracy of the assessment. The following questions may help to achieve a more accurate assessment of efforts:
- How well does the team rate the tasks?
- How often are delays caused by critical defects?
- What efforts are often spent on unnecessary tasks?
- How much time do employees spend on correcting and re-performing tasks due to insufficient quality?
- How much time do employees spend on correcting and reperforming tasks due to incorrect description of tasks?

It is important to note that there are many other resources that should be used more efficiently. However, participants did not provide any information about other activities, and individual responses cannot be used to summarize. The following questions may support the goal of better managing human resources:

- How often do employees switch between tasks?
- How well do developers focus on tasks?
- What is the speed of the team?
- How well are tasks distributed among team members?
- How often does the customer ask to change the functionality of the system?
- How well are the tasks described?

The process of improving the testing process is extremely important and includes several important aspects. First of all, the team must understand how well it finds defects in the code using various methods (unit testing, pair programming and code analysis). Understanding where the defects were detected can be helpful in streamlining the process of improving testing and finding out if it works well. Based on this information, the following questions can be identified:

- What is the overall quality of the testing process?
- At what stages are defects detected?
- Are the tests well-chosen?
- Is code review effective?
- Is pair programming effective?

A proper and defined software development process is extremely important for the development team. Before improving the process, it is necessary to describe its current state. Some respondents said they use checklists to control how well they monitor the process. In addition, one of the key characteristics of a particular program process is the ability of the team to correctly assess the effort required to complete the project phase (sprint) or the entire project. Another aspect is the quality of the final product: changes in the process increase or decrease the quality of the product and code. The overall quality of the software process can be characterized by the following questions:

- How well do we follow a certain software process?
- How well does the team predict the necessary effort?
- What product quality does the team deliver?
- What is the overall quality of the code we produce?

4.2.3 Defining Common Metrics

To monitor progress in achieving the goals, the questions need to be answered using special metrics. The most “typical” indicators were selected on the basis of whole questions from the questionnaire. The following metrics can be used to answer questions characterizing progress in improving the effectiveness of effort assessment:

- Effectiveness of effort assessment: (actual efforts / planned efforts) * 100;
- Delays caused by critical defects;
- Percentage of time to eliminate defects: (Time spent on troubleshooting /Duration of sprint) * 100;
- Percentage of time to eliminate defects (specific task): n (Time required to correct the generated errors / Time required to complete the task) * 100.

-Effort spent for unnecessary tasks;

- Percentage of completed tasks that are not needed: (Number of completed tasks that are not needed / Total number of tasks) * 100;
- Time spent on correcting and reperforming tasks due to insufficient quality;
- Ping-pong speed: the number of jobs returns to the development stage;
- Time to improve the task: Time of the first press - Time of the received press;
- Time spent on correcting and reperforming tasks due to incorrect description of tasks;
- Ping-pong speed (incorrect description of the task): the number of tasks returns to the development phase;
- Time to improve the task (incorrect description of the task): Time of the first press - Time of the received press.

To answer questions related to more efficient use of resources the following set of metrics can be used:
Number of switching between tasks;

- The number of parallel projects per employee;
- The number of tasks performed per employee;
- The average time of focus on the task;

- The degree of developers' focus on tasks;
- The average number of switching between applications;
- The average time spent in one application without switching;
- Time spent in disturbing applications;
- Team speed: the number of story points processed by the team for a certain period of time;
- Employee Speed: The number of points processed by the employee for a given period of time;
- Quality of task distribution among team members;
- Time to improve the task: Time of the first press - Time of the received press;
- The periodicity of the customer's requests to change the functionality of the system;
- The number of change requests;
- Speed of repeated execution (client request): (Number of tasks sent for revision / Total number of tasks) * 100;
- Degree of the tasks description in terms of clarity;
- Task understanding time: coding start time - task start time.

To answer questions related to more efficient use of resources the following set of metrics can be used:
- Number of switching between tasks;
- The number of parallel projects per employee;
- The number of tasks performed per employee;
- The average time of focus on the task;
- The degree of developers' focus on tasks;
- The average number of switching between applications;
- The average time spent in one application without switching;
- Time spent in disturbing applications;
- Team speed: the number of story points processed by the team for a certain period of time;
- Employee Speed: The number of points processed by the employee for a given period of time;
- Quality of task distribution among team members;
- Time to improve the task: Time of the first press - Time of the received press;
- The periodicity of the customer's request to change the functionality of the system;
- The number of change requests;
- Speed of repeated execution (client request): (Number of tasks sent for revision / Total number of tasks) * 100.
- Degree of the tasks description in terms of clarity;
- Task understanding time: coding start time - task start time.

In order to answer questions related to the deeper testing process, we need to take into account the following metrics:
- The overall quality of the testing process;
- Efficiency of elimination of defects: the number of defects detected at the stage / (the number of defects detected after the stage + the number of defects detected at the stage);
- Code coverage: (CT + CF + LC) / (2 * B + EL), CT - verification of true conditions, CF - verification of false conditions, B - number of conditions, LC -covered lines, EL - all lines;
- Scope of scenarios: scenarios tested by tests / all scenarios;
- Stages of defects detection;
- Distribution of defects: the number of defects inserted in a particular module;
- Coverage of the code (specific module);
- Degree of test quality;
- Assessment of mutations: (killed mutants / total number of mutants) *100;
- Code review efficiency;
- Efficiency of elimination of defects: the number of defects detected at the stage / (the number of defects detected after the stage + the number of defects detected at the stage);
- Percentage of time spent on this activity: (Time spent on this activity /All time spent on tasks) * 100;
- Pair programming efficiency;

- Efficiency of elimination of defects: the number of defects detected at the stage / (the number of defects detected after the stage + the number of defects detected at the stage.

Only a part of the above metrics, such as, for example, code coverage, can be calculated by analyzing the software. Therefore, such systems, which were discussed in the first part of this article can only partially be implemented in GQM model.

5 Conclusion

To sum up, the article describes that existing software metric tools interpret and implement definitions of object-oriented software metrics in different ways. This provides metric results depending on the tool and even affects analytic results based on these metric results. To summarize, evaluating a software system based on metrics and the measures taken to improve its design vary significantly from tool to tool. The paper also describes the process of creating a predefined Goal Question Metric (GQM) model that can be used to develop a dashboard for software organizations. A predefined GQM model was created for the most important purposes and recommendations were formulated for the effective visualization of valuable metrics in the toolbar.

5.1.1 Acknowledgements

This research project is carried out under the support of the Russian Science Foundation Grant No19-19-00623.

References

1. Luis Corral, Alberto Sillitti, Giancarlo Succi, Alessandro Garibbo, and Paolo Ramella. Evolution of Mobile Software Development from Platform-Specific to Web-Based Multiplatform Paradigm. In Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2011, pages 181–183, New York, NY,USA, 2011. ACM. ISBN 978-1-4503-0941-7. doi: 10.1145/2048237.2157457. URLhttp://doi.acm.org/10.1145/2048237.2157457
2. Luis Corral, Alberto Sillitti, and Giancarlo Succi. Software development processes for mobile systems: Is agile really taking over the business? InEngineering of Mobile-Enabled Systems (MOBS), 20131st International Workshop on the, pages 19–24, May 2013. doi:10.1109/MOBS.2013.6614218
3. Alberto Sillitti, Andrea Janes, Giancarlo Succi, and Tullio Vernazza. Measures for mobile users: an architecture. Journal of Systems Architecture, 50(7):393–405, 2004. doi: 10.1016/j.sysarc.2003.09.005. URLhttp://dx.doi.org/10.1016/j.sysarc.2003.09.005
4. Luis Corral, Anton B Georgiev, Alberto Sillitti, and Giancarlo Succi. A method for characterizing energy consumption in Android smartphones. In Green and Sustainable Software(GREENS 2013), 2nd International Workshop on, pages 38–45.IEEE, May 2013.doi: 10.1109/GREENS.2013.6606420.URLhttp://dx.doi.org/10.1109/GREENS.2013.6606420.
5. Luis Corral, Anton B. Georgiev, Alberto Sillitti, and Giancarlo Succi.Method Reallocation to Reduce Energy Consumption: An Implementa-tion in Android OS. InProceedings of the 29th Annual ACM Symposiumon Applied Computing, SAC '14, pages 1213–1218, Gyeongju, Republic of Korea, 2014. ACM. ISBN 978-1-4503-2469-4.
6. Anton B Georgiev, Alberto Sillitti, and Giancarlo Succi. Open Source Mo-bile Virtual Machines: An Energy Assessment of Dalvik vs. ART. InIFIPInternational Conference on Open Source Systems, pages 93–102. SpringerBerlin Heidelberg, 2014.
7. Luis Corral, Anton B. Georgiev, Alberto Sillitti, and Giancarlo Succi. Canexecution time describe accurately the energy consumption of mobile apps? An experiment in Android. InProceedings of the 3rd International Work-shop on Green and Sustainable Software, pages 31–37. ACM, 2014.
8. Luis Corral, Anton B Georgiev, Alberto Sillitti, Giancarlo Succi, and Ti-homir Vachkov. Analysis of offloading as an approach for energy-awareapplications on Android OS: A case study on image processing. InInterna-tional Conference on Mobile Web and Information Systems, pages 29–40.Springer International Publishing, 2014.

9. Luis Corral, Anton B Georgiev, Alberto Sillitti, and Giancarlo Succi.A Study of Energy-Aware Implementation Techniques: Redistribution ofComputational Jobs in Mobile Apps.Sustainable Computing: Informaticsand Systems, 7:11–23, 2015. ISSN 2210-5379
10. Tullio Vernazza, Giampiero Granatella, Giancarlo Succi, Luigi Benedicenti,and Martin Mintchev. Defining Metrics for Software Components. In Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, volume XI, pages 16–23, July 2000.
11. Kothuri Parashu Ramulu and Dr. B.V. Ramana Murhtyr. Importance of software quality models in software engineering. 2018. doi: 10.5281/ZEN-ODO.1218182. URLhttps://zenodo.org/record/1218182.
12. Brijendra Singh and Suresh Prasad Kannojia. A model for software product quality prediction. Journal of Software Engineering and Applications, 05(06):395–401, 2012. doi: 10.4236/jsea.2012.56046. URLhttps://doi.org/10.4236/jsea.2012.56046.
13. S. Sivaloganathan and N.F.O. Evbuomwan. Quality function deployment the technique: State of the art and future directions. Concurrent Engineering, 5(2):171–181, June 1997. doi: 10.1177/1063293x9700500209.URL https://doi.org/10.1177/1063293x9700500209.
14. Patrik Berander and Per Jönsson. A goal question metric based approach for efficient measurement framework definition. In Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering - ISESE06. ACM Press, 2006.doi: 10.1145/1159733.1159781.URLhttps://doi.org/10.1145/1159733.1159781.
15. Jeremy Kivi, Darlene Haydon, Jason Hayes, Ryan Schneider, and Gian-carlo Succi. Extreme programming: a university team design experience. In 2000 Canadian Conference on Electrical and Computer Engineering. Conference Proceedings. Navigating to a New Era (Cat. No.00TH8492), volume 2, pages 816–820 vol.2, May 2000. doi: 10.1109/CCECE.2000.849579.
16. Ilenia Fronza, Alberto Sillitti, and Giancarlo Succi. An Interpretation of the Results of the Analysis of Pair Programming During Novices Integration in a Team. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM '09, pages 225–235. IEEE Computer Society, 2009.ISBN 978-1-4244-4842-5. doi: 10.1109/ESEM.2009.5315998. URLhttp://dx.doi.org/10.1109/ESEM.2009.5315998.
17. Giancarlo Succi, James Paulson, and Armin Eberlein. Preliminary results from an empirical study on the growth of open source and commercial software products. InEDSER-3 Workshop, pages 14–15, 2001.
18. György L Kovács, Sylvester Drozdik, Paolo Zuliani, and Giancarlo Succi.Open Source Software for the Public Administration. InProceedings of the6th International Workshop on Computer Science and Information Tech-nologies, October 2004.
19. Etiel Petrinja, Alberto Sillitti, and Giancarlo Succi. ComparingOpenBRR, QSOS, and OMM assessment models. InOpen SourceSoftware: New Horizons - Proceedings of the 6th InternationalIFIP WG 2.13 Conference on Open Source Systems, OSS 2010, pages 224–238, Notre Dame, IN, USA, May 2010. Springer, Heidelberg. ISBN 978-3-642-13243-8. doi: 10.1007/978-3-642-13244-518.U RLhttp://dx.doi.org/10.1007/978-3-642-13244-518.Bruno Rossi, Barbara Russo, and Giancarlo Succi. Adoption of free/libre open sources of tw are in public organizations:factors of impact. Information Technology & People, 25(2) : 156—187, 2012. doi: 10.1108/09593841211232677. U RLhttp://dx.doi.org/10.1108/09593841211232677.
20. Bruno Rossi, Barbara Russo, and Giancarlo Succi. Adoption of free/libreopen source software in public organizations: factors of impact.InformationTechnology & People, 25(2):156–187, 2012.
21. Enrico Di Bella, Alberto Sillitti, and Giancarlo Succi. A multivariate classification of open source developers. Information Sciences, 221:72–83, 2013.
22. Andrea Janes, Barbara Russo, and Giancarlo Succi. Using non-invasivemeasurement techniques in agile software development: a swot analysis.InXLII Congresso Annuale AICA (AICA 2004), Benevento, Italy. Associ-azione Italiana per l'Informatica ed il Calcolo Automatico (AICA), Septem-ber 2004.
23. Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. Non-invasive Product Metrics Collection: An Architecture. InProceedings ofthe 2004 Workshop on Quantitative Techniques for Software Agile Process,QUTE-SWAP '04, pages 76–78. ACM, 2004.

24. Martina Ceschi, Barbara Russo, Alberto Sillitti, and Giancarlo Succi. Non-invasive Investigation of the Software Process in Open Source Projects.InSharing Experiences on Agile Methodologies in Open Source SoftwareDevelopment, SESAMOSS 2005. (English Edition), page 25. Polimetricasas, July 2005.
25. Raimund Moser, Andrea Janes, Barbara Russo, Alberto Sillitti, and Gi-ancarlo Succi. PROM: taking an echography of your software process. InXLIII Congresso Annuale AICA (AICA 2005), Udine, Italy. AssociazioneItaliana per l'Informatica ed il Calcolo Automatico (AICA), October 2005.
26. Alberto Sillitti, Giancarlo Succi, and Stefano De Panfilis. Managing non-invasive measurement tools.Journal of Systems Architecture, 52(11):676–683, 2006
27. Andrea Valerio, Giancarlo Succi, and Massimo Fenaroli. Domain analysisand framework-based software development.SIGAPP Appl. Comput. Rev.,5(2):4–15, September 1997. ISSN 1559-6915.
28. Chris A. Ortiz and Murry R. Park.Visual Controls. Pro-ductivity Press, June 2018. doi: 10.4324/9781466503267.URLhttps://doi.org/10.4324/9781466503267.
29. Pablo Quezada Sarmiento, Daniel Guaman, Luis Rodrigo Barba Guamán, Liliana Enciso, and Paola Cabrera. Sonarqube as a tool to identify software metricsand technical debt in the source code through static analysis. 07 2017.
30. Shyam R. Chidamber and Chris F. Kemerer. Towards a metrics suite for objectoriented design. ACM SIGPLAN Notices, 26(11):197–211, November 1991. doi:10.1145/118014.117970. URLhttps://doi.org/10.1145/118014.117970.
31. Giancarlo Succi, Witold Pedrycz, Snezana Djokic, Paolo Zuliani, and Bar-bara Russo. An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite. Empirical Software Engineering, 10(1):81–104, January 2005. doi: 10.1023/b:emse.0000048324.12188.a2. URL https://doi.org/10.1023/b:emse.0000048324.12188.a2.
32. Martin West. Book review: Object-oriented metrics: Measures of complexity. brian henderson-sellers. published by prentice hall, hemel hempstead, u.k.,1996. ISBN: 0 13 239872 9, 234 pages. price: £26.95, hard cover.SoftwareTesting, Verification and Reliability, 6(34):255–256, September 1996. doi:10.1002/(sici)1099-1689(199609/12)6:3/4<255::aid-stvr110>3.0.co;2-r. URL https://doi.org/10.1002/(sici)1099-1689(199609/12)6:3/4<255::aid-stvr110>3.0.co;2-r.
33. Yasser Khazaal, Mathias van Singer, Anne Chatton, Sophia Achab, Daniele Zullino, Stephane Rothen, Riaz Khan, Joel Billieux, and Gabriel Thorens. Does self-selection affect samples' representativeness in online surveys? aninvestigation in online video game research. Journal of Medical Inter-net Research, 16(7):e164, July 2014. doi: 10.2196/jmir.2759. URL https://doi.org/10.2196/jmir.2759.
34. Andrea Janes and Giancarlo Succi. Lean Software Development in Action. Springer Berlin Heidelberg, 2014. doi: 10.1007/978-3-642-00503-9. URL https://doi.org/10.1007/978-3-642-00503-9