A Continuous Software Quality Monitoring Approach for Small and Medium Enterprises

Andrea Janes
Free University of Bozen-Bolzano
Piazza Domenicani, 3
Bozen-Bolzano
39100 Italy
andrea.janes@unibz.it

Valentina Lenarduzzi
Free University of Bozen-Bolzano
Piazza Domenicani, 3
Bozen-Bolzano
39100 Italy
valentina.lenarduzzi@unibz.it

Alexandru Cristian Stan
Free University of Bozen-Bolzano
Piazza Domenicani, 3
Bozen-Bolzano
39100 Italy
astan@unibz.it

ABSTRACT

Context: SMEs cannot always afford the effort required for software quality assurance, and therefore there is the need of easy and affordable practices to prevent issues in the software they develop.

Object: In this paper, we propose an approach to allow SMEs to access SQA practices, using an SQA approach based on a continuous issue and error monitoring and a recommendation system that will suggest quality practices, recommending a set of quality actions based on the issues that previously created errors, so as to help SMEs to maintain quality above a minimum threshold.

Method: First, we aim at identifying a set of SQA practices applicable in SMEs, based on the main constraints of SMEs and a set of tools and practices to fulfill a complete DevOps pipeline. Second, we aim at defining a recommendation system to provide software quality feedback to micro-teams, suggesting which action(s) they should take to maintain a certain quality level and allowing them to remove the most severe issues with the lowest possible effort. Our approach will be validated by a set of local SMEs. Moreover, the tools developed will be published with an Open Source license.

Keywords

Continuous Quality Assurance; Software Monitoring; Software Maintenance; Code Smells; Anti-patterns

1. INTRODUCTION

Software quality assurance (SQA) is still a complex task that requires a lot of effort and expertise. The reasons are manifold, e.g., the fact that quality-related information is difficult to collect [11] [8], or that investment into SQA is often still put aside in favor of other activities, e.g., the addition of new functionalities [12]. Moreover, developers commonly do not trust existing one-size fits all quality models [13] [14], because of their complex interpretation that often requires dedicated SQA personnel [6]. This is why automated solutions that do not distract developers in their work and provide useful feedback to create a more effective workplace are needed.

One aspect that influences software quality are code smells and anti-patterns, i.e., particular structures in code that can cause negative effects on software maintenance and should be refactored [9]. They are a cause of low maintenance of systems and several works highlighted that reducing code smells can reduce the risk of injecting bugs in the source code. A solution to reduce the number of code smells is to apply a continuous SQA monitoring approach.

In recent years, SonarQube[1], an Open Source Continuous SQA platform for the continuous analysis of the technical quality of source code, has gained more and more popularity, and today it is the de facto standard SQA tool adopted in industry. SonarQube analyzes source code with respect to different quality aspects and presents the results in the form of a web page or a log file. SonarQube and its competitors provide a set of raw measures without interpretation. Interpretation is delegated to the developers, who must define models to interpret measurements for each project, continuously monitor the software quality, and provide ad-hoc actions to the developers for quality improvements.

To reduce the burden for developers and to understand which quality aspect and which code smell is effectively impacting software maintenance, in this paper we propose a continuous SQA monitoring approach, that combines continuous learning techniques based on SMEs common quality issues and a recommendation system to suggest developers which code smell should be removed in order to reduce the probability of injecting bugs in their code.

From a bird's-eye view, our approach foresees the following steps (see Fig. 1):

(1) While the end-user uses the developed system, a continuous monitoring system observes the state of the system and collects all faults that arise.
(2) When a fault is detected, the monitoring system collects information available on runtime about the state of the monitored system (e.g., the stack trace or a screen shot of the current view) and adds an issue to the issue tracker of the development team.
(3) When the developers solve the issue, they commit the changes into the versioning system.
(4) The continuous SQA system analyzes the changed source code, identifies changes in the collected metrics (e.g., the presence of a specific code pattern) and relates these changes to the description of the commit of the changes.
(5) A recommender system uses topic detection to identify key terms in the commit and issues descriptions and builds a model that relates the author of the commit, the current location in the source code, the key terms of the change, and the change in the collected software quality metrics collected by SonarQube.
(6) As soon as a new version is committed to the repository, a continuous inspection component uses SonarQube to get the current quality metrics for the newly committed version.

ICPE'17 Companion, April 22–26, 2017, L'Aquila, Italy.

ACM 978-1-4503-4899-7/17/04…$15.00.
DOI: http://dx.doi.org/10.1145/3053600.3053618

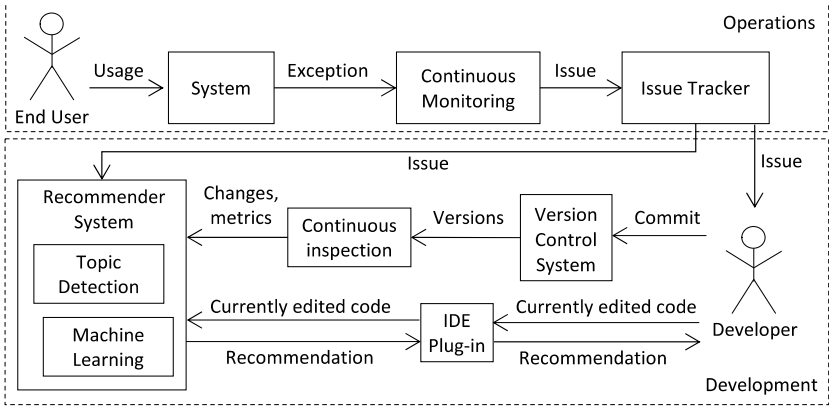

Figure 1: Data flow through the envisioned continuous monitoring system.

(7) During the development, the IDE plugin of the continuous SQA system will recommend to developers editing a particular piece of code, which quality action should be taken to reduce the probability of issues that typically arise in the system, based on historical data collected during the monitoring and the maintenance phase.

What differentiates our approach from existing IDE plug-ins for software quality is the recommendation system at the developer and development-team level that will provide customized recommendations, based on the most relevant SQA practices dynamically defined for each team. Moreover, the implementation based on an IDE Plug-in, will allow developers to reduce the complexity of the installation, since they will not require to install additional software on their local machine, but only to install the plug-in directly from the IDE and to connect the plug-in to their SonarQube instance simply providing the server url and the developer' credentials.

2. RELATED WORK AND BACKGROUND

Software quality assurance (SQA) includes methods and techniques to assure that a software has a certain (desired) quality.

Various SQA approaches have been developed in industrial contexts but, as highlighted by several studies [1], [15], [5], [4], many are not applicable to SMEs and especially to micro enterprises since they require dedicated SQA teams or large developer effort overhead for the SQA activities prescribed by the SQA team. Therefore, SMEs and in particular micro-enterprises[2] need automated ways (i.e., requiring few resources) to understand when the quality of their product is decreasing and to get continuous practical suggestions on how to maintain quality above a minimum threshold.

Existing measurement tools, such as SonarQube, support analyzing the produced source code; however, such tools often provide large amounts of data from different sources but do not provide a personalized view on the data, depending on the needs of the developer solving a specific task [3]. This is a problem that a recommender system can alleviate: recommender systems are programs that help a user to choose items (e.g., products, songs, movies) from a large offer [16]. Often the goal is to help the user to choose interesting items. In our case, the recommender system aims to point out the most relevant metrics to the developer for the specific piece of code that he or she is currently editing.

The goal of the here envisioned system is to change the developers' behavior and convince them to adopt more beneficial quality management activities, therefore, we need to design and develop persuasive solutions. This means that the recommender is not aimed at maximizing prediction accuracy (what the user will do), which is the standard performance evaluation metric in recommender systems, but at optimizing recommendation adoption. This problem will be mainly addressed by the design of persuasive interfaces and especially explanations [20][17]. Similar approaches, have been already developed [13] and [14] however, they have never been widely adopted, probably because they needed a tailored definition of the quality models, not easily manageable from SMEs.

When introducing the envisioned system to a team, the following aspects need to be considered [21]:

- Focus on essential problems: avoid measuring just for the sake of measurement but focus on problems that are relevant for the practitioner;
- Domain semantics should be understood correctly for data preparation: researchers and practitioners need to collaborate in data interpretation, data selection, and data filtering. Researchers need to understand:
  - the basic definitions of domain-specific terminologies and concepts to conduct data interpretation;
  - the connections between the data and the problem to be solved to conduct data selection; and
  - the defects and limitations of existing data to avoid incorrect inference to conduct data filtering.
- A usable system should be built early to enable a feedback loop between researchers and practitioners.
- Evaluation criteria should be tied to real tasks in practice: software analytics projects should be (at least partly) evaluated using the real tasks that they are targeted to help with.

3. ROADMAP

Overall, the research objectives and outcomes of this work are:

- Identification of a set of SQA practices applicable in SMEs, based on the main constraints of SMEs, such as personnel, budget, investments, time frame, etc. In this task, we aim at defining a set of ranked quality criteria and possible actions needed to implement them.

[2] Depending on the country, the definition for small, medium, and microenterprises vary. For example, within the European Union microenterprises have less than 10 employees, small enterprises less than 50, and medium enterprises less than 250. There are also restrictions on the turnover and the balance sheet total.

- Identification of the code smells that influence more the software maintainability based on the literature. This step is needed for the cold start of the recommendation system and will be carried out by means of a systematic literature review
- Identification of a set of tools and practices to fulfill a complete DevOps pipeline.
- Definition of a recommendation system to provide SQA feedback to micro-teams, suggesting which action(s) they should take to maintain a certain quality level and allowing them to remove the most severe issues with the lowest possible effort.
- Implementation of an IDE-plug-in to recommend the appropriate SQA action to the developers.

4. PROJECT IMPLMENTATION

To provide developers' feedback in a seamless way, without requiring developer's effort overhead, we will implement this approach in SonarQube. The extension of SonarQube will allow us to seamlessly adopt our approach in companies already using SonarQube and to learn how to apply SonarQube based on best practices reported by the aforementioned companies. To reduce the invasiveness of the approach, we will implement an extension of SonarLint as IDE plug-in. SonarLint is an OSS SonarQube plug-in available for Eclipse, IntelliJ and VisualStudio and allows developers to highlight issues in code. The result of our approach will be implemented in this plug-in so as to allow developers to see potentially risky code smells highlighted as issues without requiring them to access to other platforms or to perform any steps who would require extra effort.

The approach will be developed based on the following steps:

- *Continuous Issue Monitoring System.* This component is responsible of monitoring the system and, in case of errors or exceptions create an issue in the issue tracking system (eg. Jira3). In case of duplicated issues, the system will report that the same issue occurred again, so as to support developers in the identification of the most frequent issues.
- *Continuous Inspection tool.* This component is responsible of analyzing the quality and the related static and dynamic measures [18] that changed after the resolution of an issue. Moreover, it will automatically forward the information on the changed measures, the commit message, the issue description and the complete stack-trace to the Recommender System component. This component will be developed as an extension of SonarQube.
- *Recommender System*
  - Topic Detection analysis. This component will receive all the information forwarded by the Continuous Inspection tool, and classify the topics reported in the commit and issue description, together with the stack-trace. This classification will help developers to understand if a specific measure can impact the quality or increase the probability of issues in specific part of the systems they develop.
  - Recommender system training. This component adopts the approach defined in [7]. It will learn how to associate existing changes of metrics to issues, based on the classification provided by the topic detection component and on the metrics changed. The most suitable machine learning technique will be identified during the project. Based on the availability of data from each company we will define if the training of the machine learning algorithm will be based on data coming from one single company or on several similar companies.
- *IDE-Plugin.* Based on the recommendations provided by the recommendation system, we will implement a plug-in for SonarLint4), the IDE plug-in for SonarQube. Implementing an extension of this plug-in will allow developers who already use SonarQube to seamless adopt our approach, without need of introducing and learning how to use new tools.

5. CURRENT STATUS

Currently we analyzed the existing literature on software maintenance for SMEs and the impact of Code Smells on software maintenance, identifying a set of relevant papers reporting maintenance issues in case of code smells (e.g., [19] and [2]). The first version of the DevOps tools pipeline has been identified. We are validating a set of tools in a local ME. The initial toolset is composed by:

- *GitLab* for source code versioning;
- *Jenkins* for continuos testing and integration. Jenkins is also responsible of launching the SonarQube execution on every commit. We decided to initially adopt Jenkins instead of the GitLab continuos integration feature so as to decouple our implementation from GitLab and allow users to adopt any other versioning system supported by Jenkins;
- *Jira* issue tracker to keep track of issues;
- *SonarQube* for continuos SQA;
- *SonarLint* as IDE plug-in.

We already developed the component to allow SonarQube to correctly analyze the most common code smells, by means of Ptidej [10]. Our decision to integrate Ptidej into SonarQube was mainly driven by two reasons: on the one hand simply because, as declared in the previous section, SonarQube is not able to detect these design flaws in the source code and on the other one because we found out in the research literature that Ptidej was, in most of the cases, selected as the detection strategy among the encountered studies treating code smells, their detection and their impact on software maintainability. Figure 2 shows an example of metrics extracted from SonarQube. Please note that in the current implementation only the raw metric-numbers are presented. As already reported, a dashboard and the IDE-plug-in will be used in the final implementation. Each time our SonarQube instance started a system's analysis, in addition to the standard static code measures, also the code smells were detected and stored in the SonarQube database as any other native metrics. We validated the SonarQube components by analyzing 22 Open Source projects form the Apache Software- Foundation, analyzing more than 5000 source code commits with millions of lines of code. The raw results of the analysis are available. We currently developed a SonarQube plug-in to extract the issues in Jira and tested it among the existing issues for the analyzed projects. At this stage, the issues were the manually submitted ones, from users and developers. Once the continuous monitoring system will be implemented, we will be able to extract the related issues with the same approach. We are currently implementing a dashboard to provide the developers with the results of the code smells related metrics. At this stage, we have not yet implemented the recommendation component and neither the IDE plug-in.

6. ACKNOWLEDGMENTS

This work has been partially supported by the project SQuaSME *"recommendation techniques for Software QUAlity improvement in Small Medium Enterprise"* funded by the Free University of Bozen-Bolzano.

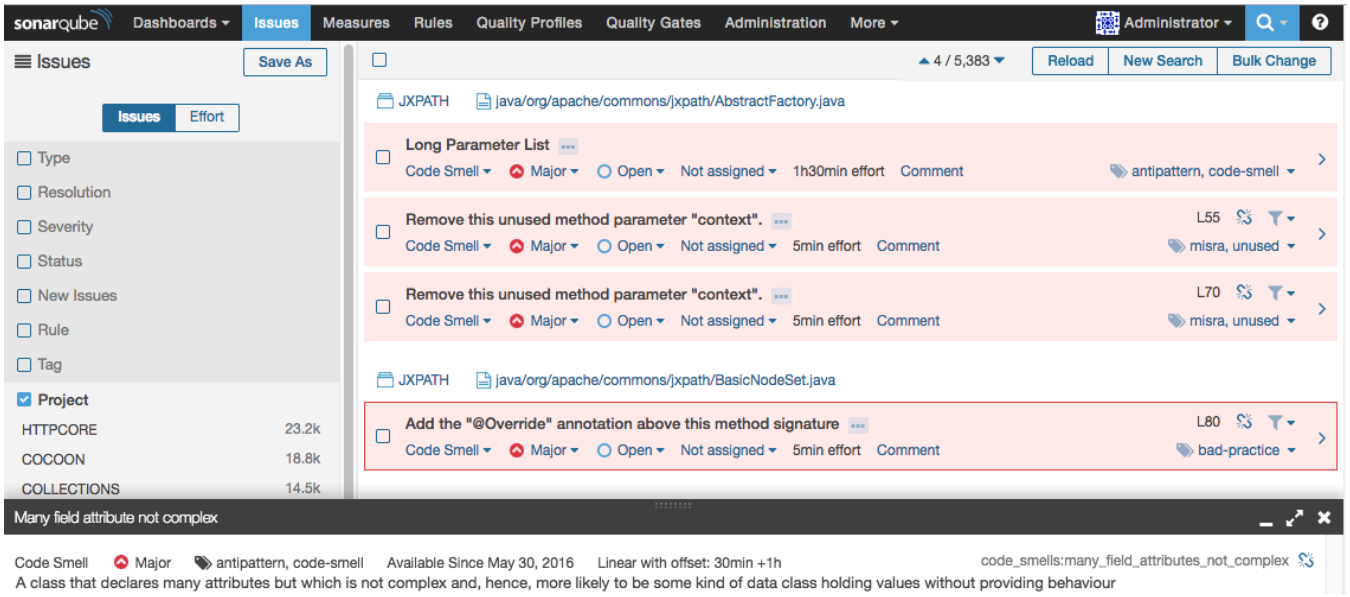

Figure 2: The current SonarQube plug-in

7. REFERENCES

[1] Almomani M.A.T., Basri S., Mahamad S., and Bajeh A.O.. 2014. Software Process Improvement Initiatives in Small and Medium Firms: A Systematic Review. In 2014 3rd International Conference on Advanced Computer Science Applications and Technologies. IEEE Computer Society, Washington, DC, USA.

[2] Bazrafshan S. and Koschke R. 2013. An Empirical Study of Clone Removals. In IEEE International Conference on Software Maintenance.

[3] Raymond P. L. Buse and Zimmermann T.. 2012. Information Needs for Software Development Analytics. In 34th International Conference on Software Engineering. IEEE Press, Piscataway, NJ, USA.Tavel, P. 2007. Modeling and Simulation Design. AK Peters Ltd., Natick, MA.

[4] Caballero E. and Calvo-Manzano J.A. 2012. A Practical Approach to Project Management in a Very Small Company.

[5] Valtierra C., Munoz M., and Mejia J. 2013. Characterization of Software Processes Improvement Needs in SMEs. In 2013 International Conference on Mechatronics, Electronics and Automotive Engineering. IEEE Computer Society, Washington, DC, USA.

[6] Del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., and Tosi, D. An Investigation of the Users' Perception of OSS Quality. 6th International Conference on Open Source Systems, OSS 2010, Notre Dame, IN, USA, May 30 – June 2, 2010.

[7] Del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., and Tosi, D. 2010. The QualiSPo Approach to OSS Product Quality Evaluation. In 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development.

[8] Diaz-Ley M., Garcia F., and Piaini M.. 2008. Implementing a software measurement program in small and medium enterprises: a suitable framework. IET Software 2, 5 (October 2008).

[9] Fowler M., and Beck K. 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley.

[10] Gueheneuc, Y. G. 2005. Ptidej: Promoting Patterns with Patterns. In 1st ECOOP workshop on Building a System using Paterns. Springer-Verlag.

[11] Hampp T.. 2012. A Cost-bene t Model for Software ality Assurance Activities. In 8th International Conference on Predictive Models in Software Engineering. ACM.

[12] Poul-Henning K. 2014. Quality Software Costs Money-heartbleed Was Free Communication ACM 57, 8 (Aug. 2014).

[13] Lavazza, L., Morasca, S., Taibi, D., and Tosi, D. 2010. Predicting OSS Trustworthiness on the Basis of Elementary Code Assessment. In 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, New York, NY, USA, Article 36.

[14] Lavazza, L., Morasca, S., Taibi, D., and Tosi, D. 2012. An Empirical Investigation of Perceived Reliability of Open Source Java Programs. In 27th Annual ACM Symposium on Applied Computing.

[15] Mishra A. and MishraD. 2013. Software Project Management Tools: A Brief Comparative View. SIGSOFT So w. Eng. Notes 38, 3 (May 2013).

[16] Ricci F., Rokach L., and Shapira B. 2015. Recommender Systems: Introduction and Challenges. In Recommender Systems Handbook. Springer.

[17] Tintarev N. and Mastho J. 2015. Explaining Recommendations: Design and Evaluation. Springer US, Boston, MA.

[18] Lavazza, L., Morasca, S., Taibi, D., and Tosi, D. 2012. On the Definition of Dynamic Software Measures. In ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, New York, NY, USA.

[19] Yamashita A. and Moonen L. 2013. Exploring the Impact of Inter-smell Relations on Software Maintainability: An Empirical Study. In 2013 International Conference on Software Engineering. IEEE Press, Piscataway, NJ, USA.

[20] Yoo K.H., Gretzel U., and Zanker M. 2012. Persuasive Recommender Systems: Conceptual Background and Implications. Springer New York.

[21] Zhang D. 2012. Software Analytics in Practice: Approaches and Experiences. In 9th IEEE Working Conference on Mining Software Repositories. IEEE Press, Piscataway, NJ, USA.