Towards a Lean Approach to Reduce Code Smells Injection: An Empirical Study

Davide Taibi(✉), Andrea Janes, and Valentina Lenarduzzi

Free University of Bozen-Bolzano, Piazza Domenicani, 3, 39100 Bozen-Bolzano, Italy
{davide.taibi,andrea.janes,valentina.lenarduzzi}@unibz.it

Abstract. Software Quality Assurance is a complex and time-expensive task. In this study we want to observe how agile developers react to just-in-time metrics about the code smells they introduce, and how the metrics influence the quality of the output.

1 Introduction and Aim of the Research

Software Quality Assurance (SQA) is still a complex task that requires effort and expertise. The reasons for this are manifold, e.g., that quality-related information is difficult to collect [2, 4] or that investment into quality is often still put aside in favor of other activities, e.g., adding new functionalities [4]. Thanks to current SQA tools available on the market, developers are able to increase their awareness on SQA. However, those tools often require substantial effort to understand the provided results.

In particular code smells, a set of structural characteristics of software that may indicate a code or design problem that can make software hard to evolve and maintain, can be easily identified with SQA tools. Developers are often not aware of the code smells they introduce in their source code; with the result of producing products with a maintainability that constantly decreases over time, due to the growth of code smells. For this reason, the identification of code smells is gaining acceptance in industry [1] but the application to agile processes is still not clear since the effort required to apply SQA tools and techniques is usually considered too high and not compliant with agile processes.

The goal of this study is to understand if SQA tools, and in particular SonarQube[1], one of the most common SQA tools, can be effectively applied in agile processes increasing the developers' productivity and the number of generated bugs.

Therefore, we formulate our research questions as follows:

- RQ1: Is the continuous application of a SQA tool (SonarQube) applicable to agile development processes?
- RQ2: Does the continuous application of a SQA tool (SonarQube) help to improve the developers' awareness of code smells in agile processes?

[1] SonarQube: http://www.sonarqube.org.

H. Sharp and T. Hall (Eds.): XP 2016, LNBIP 251, pp. 300–304, 2016.
DOI: 10.1007/978-3-319-33515-5_30

The execution of this study at XP2016 gives us the opportunity to understand if SQA tools can be effectively applied in agile processes, considering industry practitioners that would not participate in an industrial context because of effort reasons.

This paper is structured as follows: after the introductory section containing the aim of our research, we briefly discuss the background and related work in Sect. 2. Section 3 describes the design of the proposed case study.

2 Background and Related Work

Software Engineering, as every other engineering discipline, develops ways to analyze the produced artifacts with the intention to learn how to improve the various engineering methods and to produce outputs of an increasing quality. Approaches like the Experience Factory [7] recommend to have a dedicated team that studies how to improve quality and which packages the collected data into reusable knowledge so that the development teams can reuse it later. In many agile environments, this is not feasible. Developing approaches tailored for Agile and Lean environments requires understanding the specific information needs and the period in which the needed information is valuable. Particularly in Lean, a **just-in-time** approach to feedback is required: the right information at the right moment.

Moreover, the complexity of the QA domain makes results hard to interpret within small companies, since they cannot afford a dedicated team or to pay external consultancy for QA. From this point of view, a tool like SonarQube helps companies to analyze the source code with respect to different quality aspects presenting the results in form of a web page. Unfortunately, SonarQube encourages a "one size fits all" QA model in which users can analyze their source code with a set of predefined measures. This is an additional impediment for teams to use SonarQube to apply a customized QA model within their context, as it requires time and expertise. To apply QA within agile, **a tailored set of metrics has to be used** [9, 10].

3 The Case Study

The objective of our case study is to understand if the continuous application of SonarQube, tailored to an agile development process (as suggested in [9]), helps to reduce the number of injected code smells without influence the developers' productivity and if it helps developers to learn how to avoid code smells in the future.

According to our expectation, we formulate the goal of the study as follows:

analyze the continuous application SonarQube
for the purpose of evaluating and comparing
with respect to applicability and the code smells awareness
from the point of view of the developers
in the context of agile software development.

3.1 Data Collection Methods

The case study targets developers with at least three years of development experience. We aim at collecting data from developers alongside existing programming exercises in existing workshops, during the XP2016 conference.

The data will be collected in two steps: (1) during the development process (2) at the end of the development process.

Before the beginning of a coding session, we will provide the access to our tailored SonarQube platform. Our researchers will configure the platform for the projects to be developed, to avoid adding any extra task to the participants. Moreover, for those who accept to track their development activities, we will also install a tool we developed [8] that simply logs the current application on focus. Using this tool, we track the time spent by the developer per application and the time spent reading our reports. We will ask to manually track the time needed to check the report to those who prefer to not install our tool.

During the development, we will ask participants to commit the source code related to the development of a specific user story, reporting the user-story-id number. After the commit, the platform will present a short report with the list of code smells introduced in the current commit and the list of all previously introduced code smells. For usability reasons, we will also provide a printed version of the report to the developers who prefer to not switch to SonarQube to see the reports. Developers will be free to decide if the code smells should be removed or not.

To understand what participants think about our approach, we will distribute a questionnaire at the end of the development process. To answer this question, we will collect the time overhead needed to read and understand the results provided by the tools and the opinions of the participants by means of the Technology Acceptance Model [5], collecting the metrics listed below.

All statements will be evaluated based on a 5-point ordinal Likert scale with the following options: 1 (strongly disagree), 2 (disagree), 3 (neither agree nor disagree), 4 (agree), 5 (strongly agree).

Perceived usefulness: measures the degree to which the participant considers the approach useful.

- "I learned which kind of code smells I usually introduce in the source code."
- "The report pointed out code smells I was not aware of."
- "The identified code smells do not make sense."
- "The effort required to analyze the report is too high compared to the provided benefits."

Perceived understandability: measures the effort needed by the subject to understand the approach built or whether the participants will need to exert little effort to understand the relationship with the system concepts.

- "It was easy for me to understand how the approach works."

Perceived easiness: measures the degree to which the subject believes that he or she was able to make project decisions easier than without the approach.

- "It was easy for me to decide to remove the code smell or not, based on the information provided by the tool."
- "It was easy for me to identify the code smell."
- "After using the tool I was able to remember previous code smells and how to not introduce them anymore."

Self-efficacy by applying the technique.

- "The approach helped me to increase my productivity reducing refactoring time."

3.2 Data Analysis

For our fist research question (RQ1), we will analyze separately the results for the participants who will install our window tracking tool to calculate the time spent on each window with those who will report the time by manually. Then after statistical tests to check data normality, we will analyze the code smells trend in each commit, to understand the percentage of time spent on the report and on developing.

Q2 will be analyzed first performing a descriptive analysis of the collected data and then with a One-Sample Wilcoxon Signed-Rank test for comparing the obtained medians to the hypothesized median ($\alpha = 3$). Moreover, to make sure that the statements on the given scale will measure the same underlying assumption, we will perform a reliability test by calculating the Cronbach's α reliability measure.

References

1. Fontana, F.A., Braione, P., Zanoni, M.: Automatic detection of bad smells in code: an experimental assessment. J. Object Technol. **11**(2), 5:1–38 (2012)
2. Hampp, T.: A cost-benefit model for software quality assurance activities. In: Proceedings of the 8th International Conference on Predictive Models in Software Engineering (PROMISE 2012), pp. 99–108. ACM, New York (2012)
3. Kamp, P.: Quality software costs money–heartbleed was free. Commun. ACM **57**(8), 49–51 (2014)
4. Diaz-Ley, M., Garcia, F., Piattini, M.: Implementing a software measurement program in small and medium enterprises: a suitable framework. IET Softw. **2**(5), 417–436 (2008)
5. Caldera, G., Rombach, H.D., Basili, V.: Goal question metric approach. In: Encyclopedia of Software Engineering, pp. 528–532. Wiley, New York (1994)

6. Venkatesh, V., Davis, F.: A theoretical extension of the technology acceptance model: four longitudinal field studies. Manage. Sci. **46**(2), 186–204 (2000)
7. Basili, V.R., Caldiera, G., Rombach, D.H.: The experience factory. In: Encyclopedia of Software Engineering–2, Volume Set, pp. 469–476. Wiley (1994)
8. Janes, A.: Squirrel: an architecture for the systematic collection of software development data in microenterprises to support lean software development. In: International Conference on Software and System Process (ICSSP 2015), New York, USA, pp. 171–172 (2015)
9. Davis, C.W.H.: Agile Metrics in Action: Measuring and Enhancing the Performance of Agile Teams, 1st edn. Manning Publications Co., Greenwich (2015)
10. Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: Predicting OSS trustworthiness on the basis of elementary code assessment. In: 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2010) (2010)