ScrumLint: Identifying Violations of Agile Practices Using Development Artifacts

Christoph Matthies, Thomas Kowark, Keven Richly,
Matthias Uflacker, and Hasso Plattner
Hasso Plattner Institute, University of Potsdam
August-Bebel-Str. 88
Potsdam, Germany
{firstname.lastname}@hpi.de

ABSTRACT

Linting tools automatically identify source code fragments that do not follow a set of predefined standards. Such feedback tools are equally desirable for "linting" agile development processes. However, providing concrete feedback on process conformance is a challenging task, due to the intentional lack of formal agile process models. In this paper, we present ScrumLint, a tool that tackles this issue by analyzing development artifacts. On the basis of experiences with an undergraduate agile software engineering course, we defined a collection of process metrics. These contain the core ideas of agile methods and report deviations. Using this approach, development teams receive immediate feedback on their executed development practices. They can use this knowledge to improve their workflows, or can adapt the metrics to better reflect their project reality.

CCS Concepts

•Information systems → Data mining; •Software and its engineering → Agile software development;

Keywords

Scrum, Software engineering, Process metrics, Process conformance

1. INTRODUCTION

The Unix utility *lint* [6], a static code analysis tool, flags suspicious programming constructs in C source code. It allows insights into problematic constructs in a fast and automated fashion. *ScrumLint* applies this approach to Scrum and agile processes. It analyzes development artifacts, such as commits, testing statistics or user stories, and identifies patterns that constitute problems in the implementation of agile practices. The tool was developed in the context of a university software engineering course, introducing undergraduate students to Scrum. In line with teaching recommendations [9], the course features a hands-on software development project, which all students work on collaboratively. This setup requires frequent feedback by the teaching staff to allow students to learn and adapt their processes quickly. Yet, contrary to theoretical foundations, whose understanding can be assessed through exams, the quality of practical application is difficult to monitor [5]. One solution is to employ tutors, who are present during all Scrum meetings of teams [7]. They are able to gauge collaboration and can give immediate feedback. This approach falls short, however, during the crucial teamwork phases, when actual programming takes place and urgent communication and organisational challenges arise. Here it is still difficult to obtain information to base feedback on. Instead of introducing a more controlled setting, which takes away from the core agile experience [4], ScrumLint analyzes the development artifacts that are produced during regular development activities. Already in medium size projects, such as our software engineering course with 40 participants, large amounts of development data is created. The latest installment produced 379 user stories with 4707 revisions and 1802 commits featuring 26503 file changes. As such, manually finding areas of improvement, where team members deviated from agile practices, is cumbersome and scales poorly with the amount of active participants. ScrumLint automates this process, allowing insights into the state of implementation of agile practices in a team and provides a constantly available resource of feedback for team members. It is publicly available under the MIT license[1].

CHASE'16, May 16 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897586.2897602

2. SCRUMLINT OVERVIEW

ScrumLint aims at supporting a development team in adopting or adhering to agile practices. It identifies and quantifies *violations*, instances where the executed process deviates from the defined one, as mandated by Scrum or agile best practices. While the absence of detected violations does not imply a perfectly executed process, similar to linters or test coverage tools, identified violations can reveal problem areas. These represent starting points for further analysis and discussion, activities that involve collaboration and communication between team members.

ScrumLint operates on aggregated development data collected from multiple sources, e.g. code repositories like Github or build logs from continuous integration services like Travis CI. It applies a set of rules, referred to as *conformance met-*

[1]https://github.com/chrisma/ScrumLint

rics, to this data. These metrics include information about the agile practices that are measured, as well as the specifics of how to measure and evaluate deviations. We derived metrics from best practices and experiences based on running our software engineering course over the last five years, as well as literature [8]. The main challenge lies in defining and formalizing the conformance metrics, the associated agile practices and the patterns that point to a violation.

2.1 Conformance Metric Lifecycle

We adapted Zazworka et al's. [10] model of process non-conformance as a basis for detecting process violations (see Figure 1).

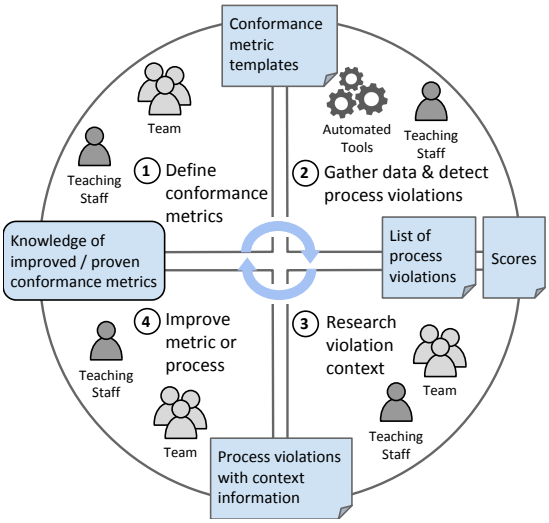

Figure 1: The conformance metric lifecycle.

It is an iterative approach that demands that metrics continually undergo "improvement" steps to make sure they stay relevant to the project's changing context. After a set of metrics and associated agile practices is defined, they are executed on the collected data, producing a list of violations. The context of these violations needs to be assessed in order to determine the best cause of action. For example, the changeset of a commit can be inspected to determine whether the connected violation is a false positive (which requires a change in the metric), or a true positive (which requires a change in process execution). The desired changes to the system and the executed process are applied and the cycle begins anew.

2.2 Result Presentation

As ScrumLint is web-based, the output presented to users is a web page containing the identified violations and their details. These are organized into categories, giving an overview of what process areas require attention. Identified violations are visualised using line and radar charts, and a score, reflecting the severity of violations, is assigned to each metric. These individual scores are aggregated into an overall *ScrumLint score*, which represents the severity of all violations of agile processes in a team for a sprint. It allows comparing teams' process conformance against each other and over iterations (see Figure 3a). A perfect score indicates that no violations were found while a low one indicates that the defined practices were rarely followed. A screenshot of ScrumLint showing a team overview page is given in Figure 2.

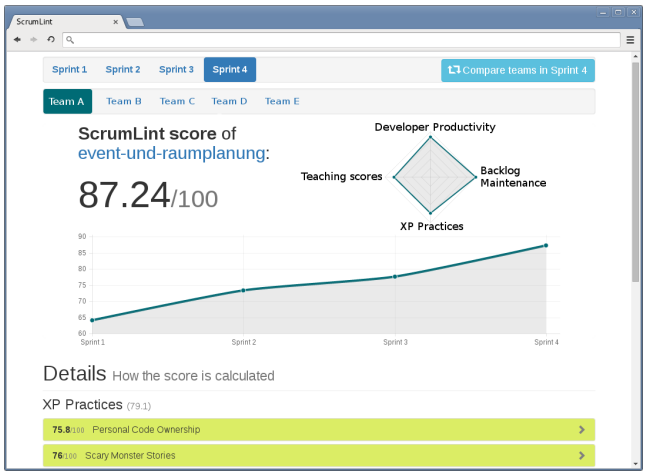

Figure 2: Screenshot showing the development of the ScrumLint score for a team.

2.3 Related Approaches

With the presented approach, ScrumLint is in line with recent similar tools, such as SQA-Mashup [2] or Microsoft CodeFlow Analytics [1]. These also aim at providing easy-to-grasp overviews of potential problems in software engineering processes and allow their users to zoom in on concrete artefacts. ScrumLint's main contribution, however, is that it is the first tool that aims to capture the core aspects of agile processes, in particular Scrum, and support teams beyond the standard agile metrics, such as Burndown charts and velocity calculation.

3. USE CASE

In an agile development team, all team members should strive to adhere to agile practices. A role in every team that is especially concerned with this is the Scrum Master (SM). The SM is tasked with supporting her team, removing blockers, and suggesting improvements to the process. She participates in development activities and has insights into how well her team is doing. However, she does not have knowledge of how her team's implementation of Scrum compares to the other teams in the project. This is of interest in order to find those areas of the process that other teams fared better and where there is learning potential.

This is a prime example where ScrumLint can be employed. It can support the SM in the following tasks:

Identify category. The SM starts research by visiting the team comparison view. Its radar chart compares all teams by category scores (see Figure 3a). The SM is able to identify categories where her team scored significantly higher or lower compared to other teams. Scoring lower might indicate a problem, while scoring higher could mean the team tried something new that is useful to other teams as well. The SM notices that her Team scored lower than other teams in the *Backlog Maintenance* category.

Identify metric. Next, the SM heads to the detail section of the team-centric view for the last sprint (see Figure 2). Here, all categories and the metrics within, are listed,

sorted by metric scores (see Figure 3b,c). She selects the metric at the top of the Backlog Maintenance section, *The Neverending Story*, which received the lowest score in this category.

Identify artifacts, research context. The details of user stories that were in the last sprint backlog as well as in the two previous ones, are presented. By following the links, the SM is led to the concrete story on Github and reads its details.

Enact improvements. Judging from the posted comments and the size of the user story, she concludes that the story is too large to be completed by the team in one iteration. She attends the next Scrum meeting, pointing out the identified stories to the assembled team and consulting with them on improving the executed process on the basis of the concrete data. Furthermore, teams that did well on this metric can be involved to find out what has worked well for them, e.g. splitting user stories by the create, read, update and delete aspects. With the knowledge of what concrete issues should be tackled and the ability to track metrics during the sprint, the team can improve their process in the next iteration.

4. ARCHITECTURE

ScrumLint is written in Python using the Django framework (see Figure 4). It implements models for conformance metrics and calculates them based on development artifacts, which are stored in a Neo4j graph database. Results of metrics are cached within the application and are updated in configurable intervals.

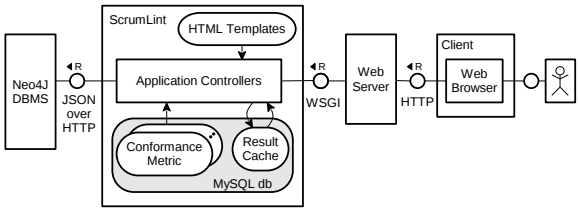

Figure 4: FMC block diagram of the architecture of ScrumLint.

4.1 Data Collection

Collection and storage of development artifacts are separated from ScrumLint, in order to simplify its reuse in different collaboration infrastructures. Currently, development artifacts from Github (commits, milestones, issues) as well as test run statistics and complexity measures for each commit are collected and written to the graph database. Furthermore, information on sprints and the composition development teams are extracted from Github. We employ a custom solution for this task, but standard solutions such as *SonarQube* [3] could easily be adapted. Adding additional data sources involves creating a new importer that has knowledge of how to the source data is connected with the existing data and that is able to write it to the database. Conformance metrics can then take advantage of the newly available data.

4.2 Conformance Metrics

Currently, the system includes ten different conformance metrics, in the categories "XP Practices", "Backlog Maintenance" and "Developer Productivity". These measure details of the Scrum process that students had problems adopting in the last iterations of the course. Table 1 gives an overview of the metrics of the Backlog Maintenance category. In order to execute a conformance metric, it must contain two main parts: a query that extracts the violation instances, and a rating function which calculates the corresponding score. Queries are defined using Cypher[2], the query language used by Neo4j, and need to include placeholders for identification of sprints and, if necessary, teams. Thus, the system is able to run each query for all teams and sprints separately to calculate score changes over time. Table 2 shows an example of the most important features of such a metric.

Table 1: Conformance metrics of the Backlog Maintenance category.

Name	Summary
The Neverending Story	User Stories in multiple backlogs.
Monster Stories	Unusually large User Stories.
Lottie and Lisa	Suspected duplicate User Stories.

Table 2: Excerpt of "The Neverending Story" conformance metric.

Name: The Neverending Story
Category: Backlog Maintenance
Severity: High
Data source: User story tracker
Description: Ideally, a sprint backlog contains exactly as many user stories as the team can complete in the iteration [Schwaber, 2013] ...
Query: `MATCH` (e:Event)-[:issue]-(i:Issue)-[:labels]-(l:Label) `WHERE` e.event="milestoned" `AND` e.title `IN` [{*sprint_list*}] `AND` l.name = "{*team*}" `WITH` i, collect(`DISTINCT` e.milestone_title) as Sprints `WITH` i, Sprints, length(Sprints) as InSprints `WHERE` InSprints > {*threshold*} `RETURN` i as Issues, InSprints, Sprints
Rating function: $max(0, 100 - (\frac{\#violations}{\#totalUS} * 100 * AvgInSprints))$, where $\#violations$ = amount of query results, $\#totalUS$ = length of Sprint Backlog, $AvgInSprints$ = average amount of sprint backlogs the violations were in.

In order to add another metric, a new instance of a conformance metric is created and the necessary fields are filled.

Users can adapt queries to their own project setup through an administrative user interface. First, the severity of a metric, the factor that a single metric influences the overall score with, can be changed. Second, what pattern is extracted as a violation can be adapted by changing the database query directly. Third, the rating function that calculates a score from violations, can be adapted. For example, thresholds in the formula can be changed, or a new exponential model can replace a linear one, where a small increase in violations result in a drastically reduced score.

5. CONCLUSION

ScrumLint allows executing and visualizing a collection of conformance metrics for a given project. It explicitly

[2] http://neo4j.com/docs/stable/cypher-query-lang.html

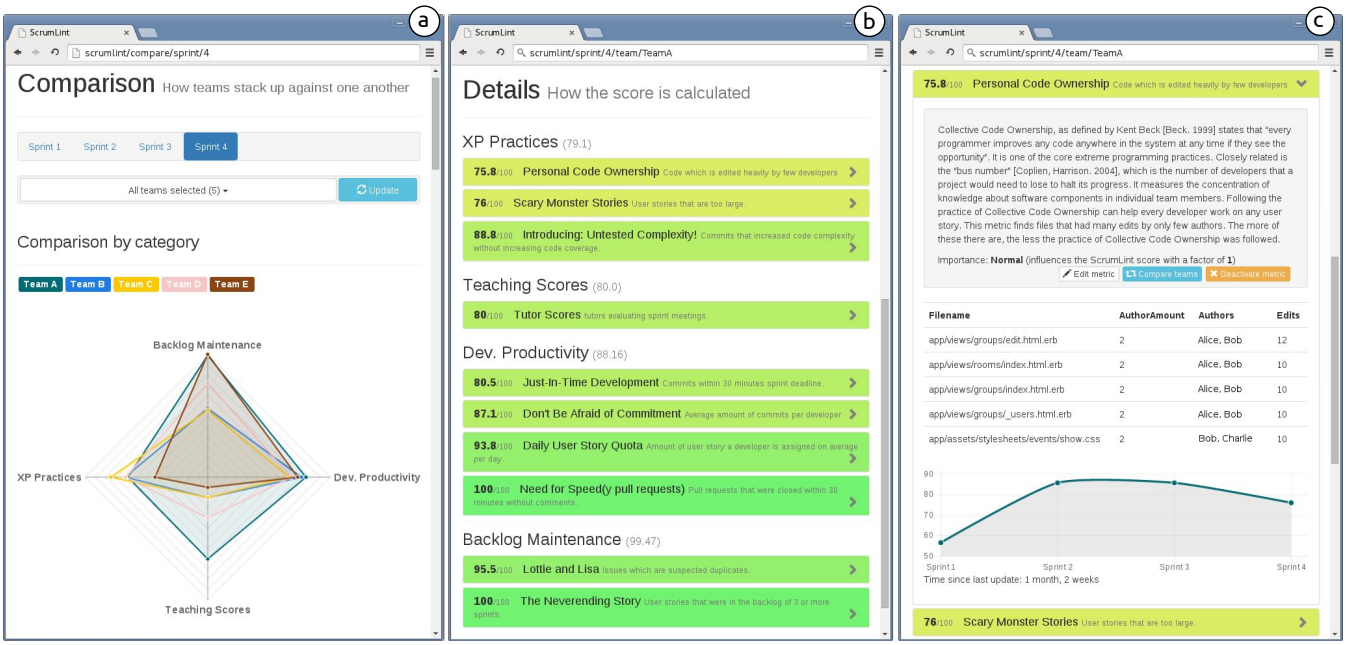

Figure 3: ScrumLint screenshots. Radar chart comparing teams by categories (a). List of conformance metrics ordered by their scores (b) and a specific metric's details expanded (c).

takes into account agile concepts such as user stories, working in agile development teams, and iterations. Results are grouped by iterations, which allows comparing conformance to Scrum practices over time. ScrumLint fits into the existing Scrum cycle, e.g. by supporting Sprint Retrospectives at the end of sprints. As most of the implemented metrics rely on existing development artifacts, existing workflows do not need to change. ScrumLint can alleviate the need to manually analyze development data, allowing the focus on the identified problem areas of the process. Violations can be tracked down to the actual artifact, e.g. a user story, allowing discussions on the basis of concrete data.

As conformance metrics are the basis of ScrumLint, their quality is mainly responsible for the quality of overall results. However, there is little research yet on what constitutes best practices for agile metrics. Zazworka et al. state that the "biggest challenge was to find definitions for the XP practices that contained enough detail" [10]. We were able to define metrics for common Scrum implementation issues based on experiences gathered from running our undergraduate software engineering course over the last years. Using ScrumLint and this relatively small amount of metrics, we were able to extract areas of improvement in the Scrum workflow of student teams for all iterations of the project. We're now interested in employing our tool in a professional setting with refined and extended metrics.

6. REFERENCES

[1] C. Bird, T. Carnahan, and M. Greiler. Lessons learned from deploying a code review analytics platform. Technical Report MSR-TR-2015-22, February 2015.

[2] M. Brandtner, E. Giger, and H. Gall. Supporting Continuous Integration by Mashing-Up Software Quality Information. In *IEEE CSMR-WCRE 2014 Software Evolution Week*, pages 109–118. IEEE, Feb. 2014.

[3] G. Campbell and P. P. Papapetrou. *SonarQube in Action*. Manning Publications Co., 2013.

[4] V. Devedzic and S. R. Milenkovic. Teaching Agile Software Development: A Case Study. *IEEE Transactions on Education*, 54:273–278, 2011.

[5] H. Igaki, N. Fukuyasu, S. Saiki, S. Matsumoto, and S. Kusumoto. Quantitative assessment with using ticket driven development for teaching scrum framework. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 372–381, New York, NY, USA, 2014. ACM.

[6] S. C. Johnson. Lint, a C Program Checker. *Comp. Sci. Tech. Rep*, pages 78–1273, 1978.

[7] C. Matthies, T. Kowark, K. Richly, M. Uflacker, and H. Plattner. How Surveys, Tutors, and Software Help to Assess Scrum Adoption in a Classroom Software Engineering Project. In *38th International Conference on Software Engineering, Software Engineering Education and Training*, Austin, TX, 2016. ACM.

[8] M. T. Sletholt, J. E. Hannay, D. Pfahl, and H. P. Langtangen. What Do We Know about Scientific Software Development's Agile Practices? *Computing in Science and Engineering*, 14(2012):24–36, 2012.

[9] The Joint Task Force on Computing Curricula. *Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. 2013.

[10] N. Zazworka, K. Stapel, E. Knauss, F. Shull, V. R. Basili, and K. Schneider. Are Developers Complying with the Process: An XP Study. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, page 14. ACM, 2010.