Mining for Process Improvements: Analyzing Software Repositories in Agile Retrospectives

Christoph Matthies
christoph.matthies@hpi.de
Hasso Plattner Institute
University of Potsdam, Germany

Franziska Dobrigkeit
franziska.dobrigkeit@hpi.de
Hasso Plattner Institute
University of Potsdam, Germany

Guenter Hesse
guenter.hesse@hpi.de
Hasso Plattner Institute
University of Potsdam, Germany

ABSTRACT

Software Repositories contain knowledge on how software engineering teams work, communicate, and collaborate. It can be used to develop a data-informed view of a team's development process, which in turn can be employed for process improvement initiatives. In modern, Agile development methods, process improvement takes place in Retrospective meetings, in which the last development iteration is discussed. However, previously proposed activities that take place in these meetings often do not rely on project data, instead depending solely on the perceptions of team members. We propose new Retrospective activities, based on *mining* the software repositories of individual teams, to complement existing approaches with more objective, data-informed process views.

CCS CONCEPTS

• **Software and its engineering → Agile software development**.

KEYWORDS

Agile Software Development, Software Process Improvement, Mining Software Repositories, Retrospective

ACM Reference Format:
Christoph Matthies, Franziska Dobrigkeit, and Guenter Hesse. 2020. Mining for Process Improvements: Analyzing Software Repositories in Agile Retrospectives. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3387940.3392168

1 INTRODUCTION

Retrospective meetings are commonly held at the end of a project to review the past work and to identify improvement opportunities. The practice of Retrospectives was embraced by the Agile community, which focuses on light-weight software development methods, iterations, and feedback [6]. Instead of waiting until the end of a project, Agile practitioners began running Retrospective meetings more frequently, e.g. at the end of Scrum Sprints [9]. Today, regular Retrospective meetings are a popular practice in professional software engineering [16].

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05.
https://doi.org/10.1145/3387940.3392168

2 DATA-INFORMED RETRO ACTIVITIES

Team activities for Retrospectives have been proposed to structure meetings and to encourage the sharing of ideas [5]. Derby and Larsen defined five consecutive phases for Retrospectives in software engineering: *set the stage*, *gather data*, *generate insights*, *decide what to do*, and *close* [5]. More recently, Baldauf introduced the *Retromat*, a book [2] and online tool, which includes most of the previously proposed exercises in a structured format. Most proposed Retrospective exercises focus on gathering the perceptions and experiences of team members and extracting improvement opportunities from them. Another view of the project reality is available through the artifacts that are produced by software developers in the course of their daily work [13]. Table 1 lists an extract of popular tools and the data that can be extracted from them. This data is useful for process improvement as it provides evidence for project problems, e.g. when tests fail [20]. Large-scale analysis of this valuable project data is the focus of the *Mining Software Repositories* (MSR) research field [8]. However, their approaches to extract insights from vast collections of software repositories have not yet been applied to software process improvement in small, Agile teams. We propose employing the software project data of development teams, to enable an additional, data-informed view of the executed process in Retrospective meetings. Our vision includes new activities for the *gather data* phase, based on software repository analyses.

In the following, we present two use cases: (i) *Action Item Discovery*, i.e. discovering opportunities for improvement and (ii) *Progress Check*, i.e. assessing the team's progress on improvement actions.

2.1 Action Item Discovery

The outcome of a Retrospective is a list of "action items" [5], that the team will work on in the next development iteration. Of the many proposed activities to gather data, only extremely few have a connection to project data [2]. We propose using data-driven activities to discover new action items. Assessments of project data can be drawn from measurements designed for Agile software engineering best practices. Examples include code coverage over time, [4], the regularity of commits to the VCS [12] or the percentage of stories implemented using Pair Programming [4].

Proposed Activity: Health Check. The Retrospective exercise is based on the established software development best practices of a team's organization, with the goal of revealing violations of these practices in the project data. To *gather data*, project data measurements concerning a practice should be collected. For example, for the "commit early, commit often" principle [1], this can include the average amount of commits per developer or the average time between commits during core working hours. In the *generate insights*

Table 1: Extract of types of tools that produce project data which can be employed in data-informed Retrospective activities.

Tool Type	Function	Examples of Extractable Data Points	Tool Example
Version Control	Track code changes, communicate rationales [15]	Code diffs, committer details, timestamps	*git*
Issue Tracker	Manage detailed information on work items [14]	Developer assignments, status updates	*Jira*
Software Tests	Present the status of current software builds [3]	Integration logs, test run logs, build status	*Jenkins*
Status Monitor	Inform/alert regarding availability of systems [7]	Accumulated uptime, downtime events	*Nagios*
Code Review	Share knowledge, gather critique of peers [19]	Time to completion, reviewer details, verdicts	*Gerrit*
Code Analysis	Provide automated feedback on code quality [18]	Code coverage results, coding style checks	*Lint*

phase, the team members can inspect the results and note whether they are outside the expected range, i.e. when adhering to the rule. The team members can compare their interpretations of analysis results, debate rationales for their observations and can find a consensus on action items for the next iteration, e.g. to commit their work to the VCS after each finished work item. In the case that results are considered to be flawed or false positives, the measurement parameters can be fine-tuned for the next iteration.

2.2 Progress Check

Without a method to gain insight into the effectiveness of Retrospectives and few tangible results, an organization might find it hard to justify the time and expense of performing Retrospectives [10]. Project artifact measurements, based on Retrospective action items, are one avenue to provide these quantifiable improvement results. Once a measurement is defined for a given action item, the results for the current (without the change) and the next iteration (with the enacted change) can be compared.

Proposed Activity: Remedy Appraisal. Suppose that in a previous Retrospective the team identified the issue of a single person committing most of the team's code changes, which slowed down the team. As an action item, all team members were trained in VCS usage. To track progress, the team can decide to employ the number of unique contributors to their code repository as a measurement. In the following Retrospective, the team appraises the effect of the remedy. The VCS can provide evidence of whether the training showed effects and whether more team members contributed code, by rerunning the previously defined measurements and comparing results. Depending on whether the results improve, i.e. show a higher contributor count, the action item can be considered resolved or can be discussed further.

3 CONCLUSION

Modern software engineers depend on digital collaboration, communication and development tools. Integrations between these tools are becoming more prevalent. An increasing amount of information on developers' interactions and behaviors is available in project artifacts, which allows improving cooperative and development processes [17]. However, these concepts have not yet fully established themselves in the domain of Agile process improvement. We propose new Retrospective activities based on project data measurements both for discovering process improvement opportunities and progress inspection. Our proposal represents initial steps in integrating the promises of the field of *Mining Software Repositories* into Agile process improvement approaches. Future work includes research on automating data-informed insights, such as through chatbots supporting Agile Retrospectives [11].

REFERENCES

[1] Jeff Atwood. 2008. Check In Early, Check In Often. http://blog.codinghorror.com/check-in-early-check-in-often/
[2] Corinna Baldauf. 2018. *Retromat - Run great agile retrospectives!* Leanpub.com. 239 pages. https://leanpub.com/retromat-activities-for-agile-retrospectives
[3] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub. In *IEEE/ACM 14th International Conference on Mining Software Repositories*. IEEE, 356–367.
[4] Bernardo Estacio, Rafael Prikladnicki, Michael Mora, Gabriel Notari, Paulo Caroli, and Alejandro Olchik. 2014. Software Kaizen: Using Agile to Form High-Perfomance Software Development Teams. In *2014 Agile Conference*. IEEE, 1–10.
[5] Derby Esther and Diana Larsen. 2006. *Agile Retrospectives: Making Good Teams Great*. Pragmatic Bookshelf. 200 pages.
[6] Martin Fowler and Jim Highsmith. 2001. The Agile Manifesto. *Software Development* 9, 8 (2001), 28–35.
[7] Marc Haberkorn and Kishor Trivedi. 2007. Availability monitor for a software based system. In *10th IEEE High Assurance Systems Engineering Symposium*. IEEE, 321–328. https://doi.org/10.1109/HASE.2007.49
[8] Ahmed E. Hassan. 2008. The road ahead for Mining Software Repositories. In *Frontiers of Software Maintenance*. IEEE, 48–57.
[9] Henrik Kniberg. 2015. *Scrum and XP From the Trenches* (2nd ed.). C4Media.
[10] David G Marshburn. 2018. Scrum retrospectives: Measuring and improving effectiveness. In *SAIS 2018 Proceedings*.
[11] Christoph Matthies, Franziska Dobrigkeit, and Guenter Hesse. 2019. An Additional Set of (Automated) Eyes: Chatbots for Agile Retrospectives. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*. IEEE Press, 34–37. https://doi.org/10.1109/BotSE.2019.00017
[12] Christoph Matthies, Thomas Kowark, Matthias Uflacker, and Hasso Plattner. 2016. Agile Metrics for a University Software Engineering Course. In *IEEE Frontiers in Education Conference*. IEEE, 1–5. https://doi.org/10.1109/FIE.2016.7757684
[13] Christoph Matthies, Ralf Teusner, and Guenter Hesse. 2018. Beyond Surveys: Analyzing Software Development Artifacts to Assess Teaching Efforts. In *IEEE Frontiers in Education Conference*. IEEE, 1–9. https://doi.org/10.1109/FIE.2018.8659205
[14] Marco Ortu, Giuseppe Destefanis, Bram Adams, Alessandro Murgia, Michele Marchesi, and Roberto Tonelli. 2015. The JIRA Repository Dataset. In *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM Press, 1–4. https://doi.org/10.1145/2810146.2810147
[15] Eddie Antonio Santos and Abram Hindle. 2016. Judging a commit by its cover. In *Proceedings of the 13th International Workshop on Mining Software Repositories*. ACM Press, 504–507. https://doi.org/10.1145/2901739.2903493
[16] Scrum Alliance. 2018. State of Scrum 2017-2018: Scaling and Agile Transformation. , 36 pages. http://info.scrumalliance.org/State-of-Scrum-2017-18
[17] Leif Singer, Margaret-Anne Storey, Fernando Figueira Filho, Alexey Zagalsky, and Daniel M German. 2017. People Analytics in Software Development. In *Grand Timely Topics in Software Engineering*. Springer, 124–153.
[18] C. C. Williams and J. K. Hollingsworth. 2005. Automatic mining of source code repositories to improve bug finding techniques. *IEEE Transactions on Software Engineering* 31, 6 (June 2005), 466–480. https://doi.org/10.1109/TSE.2005.63
[19] Xin Yang, Raula Gaikovina Kula, Norihiro Yoshida, and Hajimu Iida. 2016. Mining the modern code review repositories. In *Proceedings of the 13th International Workshop on Mining Software Repositories*. ACM Press, 460–463.
[20] Celal Ziftci and Jim Reardon. 2017. Who broke the build? Automatically identifying changes that induce test failures in continuous integration at Google Scale. In *IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track*. IEEE, 113–122.