Assisting Agile Teams to Reach Quality Goals

MARC SALLIN, Swiss Post
MEINRAD JEAN-RICHARD, Swiss Post

Agile teams are expected to follow the agile principle of technical excellence to enable agility. However, even highly skilled and motivated agile teams are prone to failing to achieve their own goals. Particularly hard to achieve are quality goals that need constant attention, for example, no compiler warnings, full test coverage, or no errors in production logs. After a while, the broken window effect kicks in, and suddenly the effort to reduce the exceptions is too high, and they give up on the goal. We present a simple yet effective practice to assist agile teams in reaching their quality goals.

1. INTRODUCTION

When we were asked to develop a new software system, we were motivated by the latest State of DevOps report and had been striving for continuous deployment [1]. From the DevOps report and our experience, we knew that quality is critical, and we must work towards it from the beginning [2]. Together with the team, we agreed on high goals. We aimed for a 100% unit test coverage, decided not to accept any static code analysis warnings, and wanted to act on errors and warnings in production immediately. In the upcoming days, we reflected on our past experiences. This was not the first time we saw a development team aim for this set of goals. However, we never saw a team successfully reaching it or, even worse, we saw the metrics degrading over time. Initially, everybody is motivated, the pressure is low, and the pull requests are reviewed carefully. Then issues start to pile up. There is a new release of the static code analysis tool, and the existing code base now has some warnings that need to be fixed, but nobody sees them since they do not pop up in a pull request. The application suddenly logs a few errors a day, and the team gets used to the fact that there are always errors. Actual errors that should be investigated disappear in the heaps of false positives. Code coverage drops because of a hastily published bug fix without tests; nobody adds them later. When the time pressure increases, these things get worse. Later, monitoring and support also eat away time when the first version of an application is running in production. At some point, the initial enthusiasm is gone, and the pile of work is too big to act on. Even if somebody still feels responsible for the quality, it gets tiresome quickly to remind the team repeatedly to keep the coverage high, fix quality issues or investigate logged errors.

What is the problem? The team lacks a systematic and lightweight approach to check the overall quality of the project and an incentive to improve upon it. To address these issues, we needed a solution that enables the team to regularly discuss the quality and check on new issues that arise. Every developer should play an active role in these sessions to distribute the responsibility to the team. The idea of the practice “quality report” was born.

2. BACKGROUND

In 2018, Marc Sallin and Meinrad Jean-Richard started as Solution Architects in the IT unit of Swiss Post. The IT unit has approximately 1500 full-time employees, including 330 software developers. The unit solely works on internal projects for the company’s strategic divisions and affiliated function units. Our team comprises a Product Owner, a Scrum Master, two Solution Architects, twelve Developers, and four Business Analysts. The agile team is responsible for the development and 24/7 operations of thirteen applications. After a year, we laid the foundation for a large new software system, responsible for parcel sorting.
It determines who will receive a parcel and ensures it flows smoothly through the Swiss Post parcel sorting centers to reach the responsible parcel deliverer on time. The software system is very important for Swiss Post and needs to be available 24/7, reliable, and functionally correct. We have already worked together in a previous company for three years. We left and met a few years later at Swiss Post. We share a passion for software engineering, complex problems and environments, and high-quality work. High quality enables us to be fast, so we constantly push for automation and lightweight processes while never sacrificing internal and external quality attributes. Both of us are developers at heart; we do many code reviews and still contribute code besides tuning requirements engineering processes and designing software solutions. Nevertheless, we like to take a holistic view and claim to be skilled in all software development life cycle activities. We acknowledge that software does not have an end in itself but that we serve the business. Thus, we leverage technical excellence to serve the business and deliberately and transparently use technical debt.

3. PRACTICE

This chapter describes the quality report practice and is divided into roles, report, and ceremony. The practice is very simple and lightweight. Once a week, the team meets to discuss specific metrics from the software system. Developers get assigned to applications and must report the metrics. That means they either suggest an action to improve towards the goal or justify the current state. One person, the Quality Champion, is responsible for asking questions and insisting if a justification should instead be an action. The Product Owner signs the weekly report to give a sense of importance. It is an open meeting. Often all developers and also some other roles like Business Analysts attend.

3.1 Roles

**Quality Champion**: This person moderates the meeting. The role is to question trends/values, request actions, and follow up the week after. The person in this role also writes the management summary and makes suggestions for the Product Owner. The purpose of that role is to make the team judge the event as important and feel rewarded (e.g. when completing an action). In our case, that role is taken by a solution architect. The solution architect is an important and high-prestige role in our organization - a critical aspect of the concept.

**Lead Developers**: One Lead Developer is responsible for one or more applications. That means investigating the metrics and defining actions or justifying why no actions must be taken. We call this reporting to the Quality Champion.

**Product Owner**: The Product Owner does not attend but reads and signs the reports. He signals importance by just looking at the report and signing it, which incentivizes the team. Moreover, he reads the management summary to understand how the agile team is doing concerning operations and code quality.

3.2 Report

The report is a Confluence wiki page. It contains a section per application with several relevant metrics. The metrics include the number of static code analysis issues, the code coverage, logged errors and warnings, and many more. For every metric, the report contains the value from this and the last week, a trend, and an indicator to show if the reported value is within an expected range. The page is generated by a C# console application that is executed automatically every Monday, one hour before the meeting takes place.

3.3 Ceremony

The ceremony is a recurring online meeting of one hour every Monday morning. We split it into three different phases, described below.

**Preparation Phase**: We start with ten minutes of silence. The Lead Developers go through the metrics of their applications. There are thresholds configured and the metrics are marked as "action" or "ok". Every metric that is marked with "action" needs to be looked at. The developer defines an action like "fix a static code analysis issue" or "further investigate the logged error from xy" or justifies why no action is necessary. This could be "Test coverage is below 100% because we have temporary code we will remove within a month" or "the errors on xy were because of a Kafka outage, all applications recovered automatically". The necessary actions or justifications are written directly onto the Wiki page. Sometimes the silence is broken by questions like "I see Kafka errors on xy, does anybody else have this issue?". This is the main reason why we do the preparation together, and we discovered that joint preparation is more reliable than solo work.
**Reporting Phase:** Each Lead Developer presents the findings of the investigation. The Quality Champion does active listening, requests actions, and does not accept sloppy excuses. There is an agreement with the Product Owner that actions taking a few hours may be done immediately. This is to give the feeling of importance and that we do not create tickets that nobody will work on. This is essential because often, minor things do not have a high priority and accumulate over time, and then, suddenly, it is too much to address.

**Discussion Phase:** The Quality Champion asks if there is something to add. Here discussions about reducing false negatives or improving the process take place. There is also room for junior engineers to ask technical questions about something that was reported.

**Closing:** The Quality Champion closes by mentioning what he will write in the management summary. After the meeting, the Quality Champion writes the management summary, signs the report, and marks the report as "official". The Product Owner gets notified about this event by the Wiki and will sign the report. As the editors get automatic notifications, this is also an important sign of interest.

4. REPORT

In this chapter, we present the generated report and explain the structure and the elements. A generated report has the following main chapters: Approval, Signatures, Table of Contents, Management Summary, and Applications. The first part of the report contains meta information such as the approval indicating the report is completed and the signature of the Quality Champion and the Product Owner. It also contains a table of contents for quick navigation. Fig. 1 shows the header part with the table of contents omitted. The management summary (Fig. 1 on the right) is a mixture of automatically and manually created content. It enables the Product Owner to assess the quality report’s results quickly. The first section is the application summary which gives an overview of the applications and is essentially the same as what was reported for a single application. The Operations and Code Quality sections are manually written by the Quality Champion based on the analysis during the ceremony. The DevOps metrics and Jira metrics sections are automatically generated. The DevOps metrics are measured based on the suggestion of Forsgren et al. [1]. The Jira metrics help the Product Owner and the team to maintain a backlog of a meaningful size.

![Figure 1. Header Part of the Report (left) and Management Summary of the Report (right)](image-url)
The applications chapter of the report contains a sub-chapter for each application. Figure 2 shows one application entry. Each application can have different data sources. E.g., an application with just a front end will not contain logged warnings and errors. Currently, we support the following data sources: Sonar, Jenkins, Splunk, Jira, and Bitbucket. From Sonar we get metrics 1-7. We accept no issues and hence 1-4 have a threshold of zero. For the test coverage, we mark it as okay from 99% upwards because we have some bugs in the coverage library that are being fixed. As we do not accept ignored tests, the threshold is zero for this metric. For code duplication, there is no threshold, we catch it in the pull request reviews as long as the applications aren’t that big. Lines of code, the build time, and the number of tests may be an indicator to split the application. We did not define any upper limits for it. The build time comes from Jenkins and includes the time for the deployment to production, hence we set the threshold to ten minutes. We don’t accept any warnings or errors in the logs and for both metrics the threshold is zero, we get them from Splunk. The same goes for reported bugs in Jira. We get many pull requests from the developers and also from the automatic dependency updates, hence, we define pull requests in Bitbucket as stale when they are older than five days and we want this number to be zero. Last but not least, there are application metrics. The idea is to look at the dashboards with numerous technical metrics (such as memory usage and garbage collection) and investigate if actions are necessary. We will automate this in the future.

For making the life of the Lead Developers easier we have quick access deep links to all sources in the header of the report. The bottom of the report contains headings to add actions and comments with reference to the metrics.

![Figure 2. Header Part of the Report](image-url)
5. IDEAS BEHIND THE QUALITY REPORT

This chapter describes our reasoning behind the practice’s roles, artifacts, and ceremony. Moreover, we present findings that we made during the time using the practice and describe what we tried but did not work for us.

5.1 Why teams do not reach their goals

As briefly described in the introduction, we invested some time determining the reasons for the observed effect - agile teams often do not reach their goals. By reflecting on our experience, we identified the following aspects.

Incentive: In a world of constant pressure for new features and faster delivery, a developer finds himself often incentivized to work on tasks they feel honored for like features or bugs with high direct impact. Organizations often believe they can trade off quality for speed and the prevention paradox hinders them from learning that high quality is the factor that leads to fewer defects and faster shipment of new features.

Constant/regular attention: Certain things must be revisited to avoid losing track of them. It is like a retrospective in which the team regularly inspects and adapts. However, when there isn’t a fixed recurring schedule to do so, the team easily forgets about it, and the daily whirlwind gets into control.

Broken Window Effect: The described effects above lead to deterioration of quality in essential measures like code coverage and static code analysis, warnings, or logs of a particular log level. Deterioration attracts more deterioration [3]; the metrics worsen over time, and one gets less meaning out of them. This is a problem, especially because often one missing unit test or a static code analysis issue is too small to get priority over other issues in the backlog. Therefore, even if it is discovered, the issue will most likely disappear in the backlog and never be solved.

High Effort: Even if there are several tools available to a developer who wants to check the quality of an application it quickly takes a lot of time or is too cumbersome to check all those tools for findings and metrics. After a while, the tools that need more effort to check are skipped and important metrics are ignored – developers do not like repetitive tasks.

5.2 How to solve these problems

Incentive: In an environment like information technology a person has a lot of creative freedom within the tasks they get assigned. Most of the time, deadlines are artificial, and estimates are not the absolute truth. To incentivize persons to invest in high-quality, even if this is not valued in the organization itself, non-monetary incentives should drive employees to spend time naturally. E.g., Dan Ariely [4] found that we do work for less money when we feel appreciated, or we see that our work helps others. Therefore, we created incentives by constructing a sense of importance by bringing the Product Owner on board and by creating the Quality Champion role. The person(s) fulfilling the Quality Champion role must have a very good technical understanding of the possible issues that will come up during the meeting so that they can ask the right questions and thus signal importance. A certain amount of authority for this person is essential to ensure tasks outside of this meeting are done. In most cases, the Product Owner is not the right person for this but has a more representative role. He does not have to understand the issues exactly but must read and acknowledge the report summary. It helps if he asks questions occasionally to ensure the team feels the importance of their work. He has to support the whole procedure and let the team prioritize the task outside the meeting.

Constant/regular attention: We generalized our experience that blockers in a calendar for solo work are often neglected while recurring meetings with other persons are not. Thus, we decided to arrange a regular meeting for the quality report. To prevent diffusion of responsibility we explicitly made a person the “Lead Developer” of an application. Letting all Lead Developers report their findings builds up slight peer pressure for all to investigate issues with their application. We defined challenging thresholds per metric and when the threshold is exceeded, we add a red-colored call for action to attract attention. It is important explicitly let the Lead Developers report why a threshold was exceeded to trigger a root cause analysis. Every call for action must either be justified by a comment, or an action must be defined. Trends are more powerful than absolute numbers because they show progress immediately and the need to reason what shift in absolute numbers towards to goal is worth it diminishes. Hence, we provided colored trend indicators for every metric.

High Effort: Reducing the effort is easily done by automating everything. We decided to collect metrics from tools automatically and put them into one place. That way, we can also automatically add calls for actions when thresholds are exceeded and show the trends. First, we started doing things manually, to see if the practice and the ceremony works but the effort was soon too high and that led to sloppy reporting.
**Broken Window Effect:** To make sure that these small issues are fixed immediately and do not pile up it is essential that the Product Owner agrees that the team can resolve these without consulting him or creating tickets. To increase the importance of these small issues, those actions are written down in the wiki and assigned immediately to a developer. It is the responsibility of the quality champion to ask about actions that were not resolved within time. There are also bigger issues that are discovered, and the root cause needs to be fixed. Therefore, it is also important to have a Product Owner that is willing to spend a fixed amount of time working on technical issues for which the team gives priorities.

5.3 What we also discovered

While working with and improving the practice we found a few things that we think helped to make the practice a success.

**Timeslot:** Doing the ceremony at the start of the week and in the morning has two benefits: Issues that come up during the weekend can be discussed while they are still fresh, and there is enough time to fix minor issues right after the meeting. However, the most critical aspect is that the ceremony is recurring and not optional.

**Collaboration:** An essential aspect of the quality report is collaboration during the meeting. It is important that everybody can easily edit the report and write comments directly into the report. That is why we chose Confluence as the tool to write the report. We made the preparation phase part of the meeting to ensure it was done. This allows for quick questions and the sharing of findings that might affect other applications. Another benefit is that whoever has already made his applications can take over others from Lead Developers who are absent or need more time.

**Operations:** It helps if the developers are responsible for operating the applications and will directly see the benefit of continuously improving the quality to produce fewer errors (and therefore fewer wake-up calls during the night). This will provide an incentive to fix the root cause of a bug or eliminate false positives in the logs and generally guide them into a "you build it, you run it" mentality.

5.4 Things we tried, but did not work out

**Noisy logs:** If there are too many errors and warnings developers get used to it and look at the bulk of the logs and may miss the one relevant log entry. Therefore, we quickly concluded that we need a zero-error log policy and continuously work towards that goal.

**Involve Non-Developer Roles:** After seeing the benefit of our practice, we started to invite operations and testing in the hope that they could contribute to solutions and analysis. Even though they showed up for the sessions, there was almost no feedback or contributions from them, and after some time, they stopped showing up entirely. Meanwhile, we switched to a DevOps approach and now there are only developers and business analysts in our team responsible for building and operating the software system.

**Solo Preparation:** Initially, people were expected to come to the meeting already prepared and discuss the findings during the session, giving them as much time for preparation as needed. The result was that often they came to the meeting unprepared due to other pressing issues. In addition to that, there were often things that occurred in every application (e.g., database maintenance) but because we did not collaborate enough, everybody had to spend time analyzing this.

**No Quality Champion:** We tried delegating responsibility to the team. That means there was no explicit moderator, and nobody requested specific actions or questioned the claims made. The team continued the practice, but false claims about why something happened crept in. This was not by intention but maybe due to the lack of knowledge, they stopped before deeply investigating, e.g., a logged error.

**Manual Inspection:** There is one element on the quality report, "Application metrics". The idea is that the Grafana dashboard with all metrics for an application and the Splunk dashboards are manually inspected for anomalies. We discovered this doesn’t work well because the task is unclear, and there is no "call to action" like a threshold leading to an "action" state or a trend indicator indicating degeneration.
6. TEAM FEEDBACK

We sent a survey to all thirteen developers on the team and got eleven answers. Seven out of eleven developers were responsible for an application as Lead Developers. We used a five-point Likert scale with an additional "Don't know" to ask the developers about their thoughts on maintenance and operational quality, the influence on the team goals, and if they think it is worth the time. The results (see Figure 4) indicate that the quality report practice particularly increases operational stability, but we also see much agreement on maintenance quality. We hypothesize that the weak link between the maintenance quality metrics and the actual maintenance quality of the software leads to less agreement there. There is an overall strong agreement that the practice helps the team to reach its quality goals, which supports the claim of this report. However, we see slightly less agreement on the question about the value of the practice. One developer selected "Don't know", which we do not further analyze, but there is also one "neutral". It would be interesting to hear the reason for this selection, as this surprises when seen in the context of the other answers.

![Figure 3. Team Survey Results](image)

Furthermore, we asked about the most essential aspect of the quality report. The participants mentioned that they value the permanent attention to the quality aspects due to the practice and the recurring and structured meetings. They see metrics collection in a central place as crucial and mention that automation is critical. They mention that sharing knowledge during the analysis is vital and that they like the space for discussing quality improvements.

7. CONCLUSION & OUTLOOK

Even teams that commit themselves to goals might need assistance to reach them due to the effect of the daily whirlwind and pressure from the market. Therefore, we consider it important to implement a practice that brings incentives to take care of certain quality aspects, helps to regularly pay attention, prevents the broken window effect, and reduces the overhead of applying itself. Additionally, we think that collaborative aspects and personal responsibility are essential factors.

After ~150 weeks (about three years) of applying this practice, we are still actively doing it and can constantly hold our goals like 100% test coverage, no static code analysis issues, and a zero-error policy. It helped us to remove noise in the logs and to spot rare events like a race condition that occurred only once in 2 million runs. We never forget bugs but act on them by deliberately closing or treating them with priority, and we are proud not to have stale pull requests. However, it is constant work and needs permanent attention. As the survey results show, the practice is highly valued within the team and that helps us to keep on going.

Besides continuing to use the tool in our team, we have several requests from other teams to use this practice and tool. We started internally implementing the tool as a SaaS solution and will promote the practice. We partnered with Security to onboard more measures like Fortify, OpsGenie, and Artifactory X-Ray. When onboarding other teams, we may conduct action research to see how the practice works in another context.
8. ACKNOWLEDGEMENTS

We would like to take a moment to express our sincere gratitude to everyone who has contributed to the development and success of this report. Firstly, we would like to thank the development team for their willingness to try out the practice and for providing valuable insights and feedback. We would also like to express our gratitude to David and Markus for their diligent proofreading and editing of the report. Your attention to detail and commitment to ensuring the accuracy and clarity of the content is greatly appreciated. A special thank goes to Swiss Post for their support and permission to write and present this report. Lastly, we would like to thank our shepherd for his guidance, support, and feedback throughout the development of this report. Your keen insights and thoughtful questions helped shape and refine our ideas and arguments, and we are incredibly grateful for your contributions.

REFERENCES
Publication date: May 2023.