More Software Analytics Patterns: Broad-Spectrum Diagnostic and Embedded Improvements

Duarte Oliveira, Faculty of Engineering, University of Porto. Porto, Portugal
João Fidalgo, Faculty of Engineering, University of Porto. Porto, Portugal
Joelma Choma, National Institute for Space Research - INPE. Brazil
Eduardo Guerra, Free University of Bozen-Bolzano. Bolzano, Italy
Filipe F. Correia, Faculty of Engineering, University of Porto. INESC TEC. Porto, Portugal

Software analytics is a data-driven approach to decision making, which allows software practitioners to leverage valuable insights from data about software to achieve higher development process productivity and improve different aspects of software quality. In previous work, a set of patterns for adopting a lean software analytics process was identified through a literature review. This paper presents two patterns to add to the original set, forming a pattern language for adopting software analytics practices that aims to inform decision-making activities of software practitioners. The writing of these two patterns was informed by the solutions employed in the context of two case studies on software analytics practices, and the patterns were further validated by searching for their occurrence in the literature. The pattern BROAD-SPECTRUM DIAGNOSTIC proposes to conduct more broad analysis based on common metrics when the team does not have the expertise to understand the kind of problems that software analytics can help to solve; and the pattern EMBEDDED IMPROVEMENTS suggests adding improvement tasks as part of other routine activities.

Categories and Subject Descriptors: D.2.8 [**Software and its engineering**]: Software creation and management—*Metrics*

General Terms: Software Analytics

Additional Key Words and Phrases: Software Analytics, Decision Making, Agile Software Development, Patterns, Software Measurement, Development Teams

ACM Reference Format:

Oliveira et al. 2021. More Software Analytics Patterns: Broad-Spectrum Diagnostic and Embedded Improvements. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 22 (June 2021), 10 pages.

1. INTRODUCTION

The term *analytics* refers to the extensive use of data, analysis, and systematic reasoning to leverage in-depth information about a given issue and support professionals' decision-making process from different areas [Davenport 2009]. For instance, approaches using analytics have been applied for some time in the marketing area to reach and better understand customers from different markets [Davenport 2010]. Most recently, analytics have also been widely used in the domain of software as a way to support practitioners (*e.g.*, developers, testers, designers, and managers) to make better decisions regarding their processes, products, and services [Zhang et al. 2011].

Zhang et al. [2011] coined the term *software analytics* to refer to the use of analytics to leverage insightful and actionable information from software data and inform the decisions of software practitioners [Zhang et al. 2011]. By *software data*, we mean the data generated as a result of developing a software system. Some of it may be recorded in repositories, such as version-control and issue-tracking systems, and includes artifacts such as source code, bug reports, or test execution reports. Other such data may be generated during the operation of the system

and can be found in artifacts such as log files [Storey 2016]. The typical issues addressed by software analytics are related to software failures and defect prediction, software requirements, code quality, releases and code integration, project management, teamwork, software maintenance, and software evolution, among others [Buse and Zimmermann 2010; Hassan and Xie 2010].

Although widely adopted by software companies, software analytics is often not explored to its full potential. Some studies have led to the proposal of analytics methods and tools, but few of them provide detail on how to adopt software analytics practices in real-world projects [Zhang et al. 2013; Huijgens et al. 2018; Augustine et al. 2018; Snyder and Curtis 2018]. Furthermore, many software practitioners are still reluctant to adopt software analytics practices [Robbes et al. 2013]. For different business reasons, and especially to meet customers' expectations, many teams focus on external quality aspects (the functionalities as perceived by end-users) and devote less time than what they would like to internal quality aspects (*e.g.*, architecture, maintainability, flexibility, security).

In previous works, we propose eight patterns for software analytics (SA) [Choma et al. 2018; Choma et al. 2017], that will help a team to adopt software analytics practices in their projects. These patterns have later helped us to propose the Software Analytics Canvas (SA Canvas) [Choma et al. 2019], with the goal of consolidating the patterns and making their application as practical as possible. The SA Canvas is designed to support planning and managing software analytics activities during software development. By evaluating the canvas in practice, we have identified new patterns.

In this paper, we present two patterns identified during a study of the SA Canvas in two software companies. The first is a large company with multiple projects and strictly-structured teams. Given its business domain, the company takes security and data privacy particularly seriously. Our study was conducted with a small team from a recent project involving four developers, of which three had no experience and no knowledge of software analytics. This study took place over three sprints, totaling 12 weeks. The second company is a software start-up. Start-ups are characterized by rapid evolution, uncertainty about customer and market needs, lack of resources, and small teams [Schmitt et al. 2018]. The study's main objective at this start-up was to evaluate the use of SA Canvas to improve the internal quality of their product. The study involved two teams whose members had no experience in software analytics. The former had four developers mainly focused on back-end development. The latter involved five developers, three of them focused on back-end development, and two focused on front-end development. Both teams followed the Scrum framework and adopted the SA Canvas during five sprints, totaling 10 weeks.

The two patterns presented in this paper arose from the difficulties we observed when applying SA Canvas at these two software companies. They describe solutions for overcoming obstacles when adopting software analytics practices. The first pattern can make the team more aware of the kind of information it can generate through the use of metrics and help it to find the first issues to tackle through analytics. The second pattern can be used when acting upon the analytics' findings, proposing how to implement improvements with a low impact on other tasks.

2. SOFTWARE ANALYTICS PATTERNS

In this section, we first provide an overview of the SA patterns we have published in previous studies [Choma et al. 2018; Choma et al. 2017]. Then, we briefly introduce the two new patterns proposed in this paper, describing how they fit into the previous ones. An overview of the SA patterns showing how they relate to each other is depicted in Figure 1. The blocks with black backgrounds represent those patterns for adopting software analytics that we have published in previous studies, and those with gray backgrounds are the patterns that we are proposing in this article. The blocks with white backgrounds and dashed black borders represent outputs to be expected from applying the patterns. The questions included next to relationship arrows motivate the use of the target pattern.

2.1 Pattern Language Overview

As previously mentioned, the first eight SA patterns emerged from a literature review, in which we searched for best practices in experience reports to identify the typical issues addressed with software analytics. A summary of the eight patterns containing a brief description of each of them is presented below.

(1) WHAT YOU NEED TO KNOW: To solve the issues that the team wants to improve in the system and/or the software development process, in a context where there is a large amount of software data that can inform the decisions of the team, the solution is to define the key issues that the development team wants to focus on, in order to improve the software throughout the project.

(2) CHOOSE THE MEANS: To solve how to gather useful data regarding the issues that the team needs to solve, in a context where a plethora of data is available, the solution is to define the most appropriate means, such as metrics, tools, techniques and other approaches for extracting data from software artifacts that will be useful in future decisions.

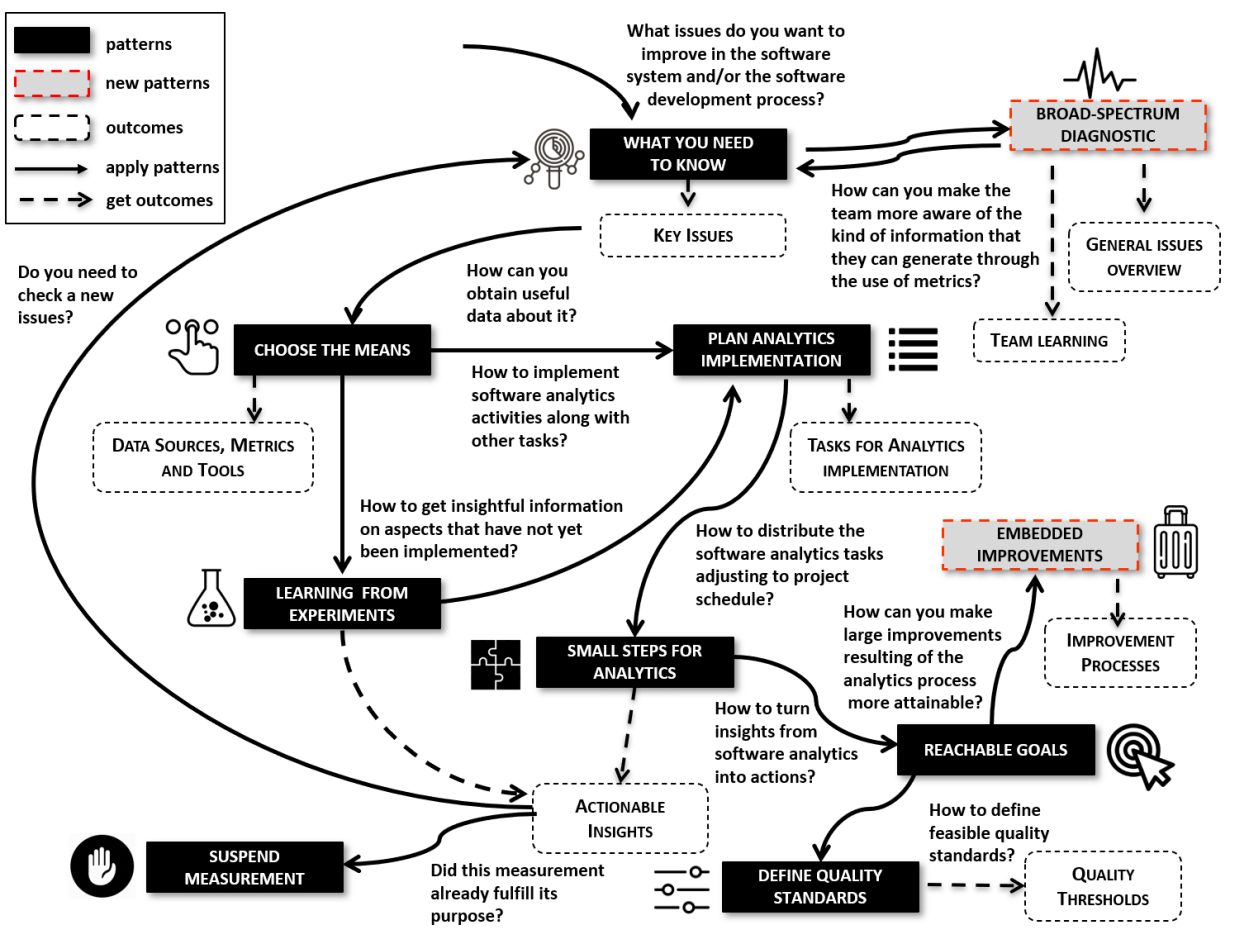

Fig. 1. Overview of the patterns and their relationships.

(3) PLAN ANALYTICS IMPLEMENTATION: To solve how to schedule the software analytics activities fitting them to the project roadmap along with other development tasks, in the context where the tasks directly related to the implementation of software features are the top priority, the solution is add tasks related to the software analytics in the backlog to be prioritized with the regular project tasks.

(4) SMALL STEPS FOR ANALYTICS: To solve how to schedule software analytics in a pace that it does not to overburden the team, in the context where much information at the same time can confuse and make the team lose focus, the solution is to adjust software analytics tasks within the team schedule by breaking down them at smaller portions to be carried out in multi-steps.

(5) REACHABLE GOALS: To solve how to turn software analytics findings into actionable insights to improve software aspects, in a context where performing all improvements based on the analytics automated feedback might lead the team to act without focus, the solution is to take actionable insights from the software analytics findings, and from them, settle reachable goal adjusting the action steps.

(6) LEARNING FROM EXPERIMENTS: To solve how to obtain information to make informed decisions about software issues on some aspect we have not yet implemented or we need to redesign, in a context where the team has nowhere yet to collect and analyze data to support their decisions, the solution is to create an alternative solution and perform an experiment collecting data that allow the comparison with the current solution.

(7) DEFINE QUALITY STANDARDS: To solve how to achieve and maintain a good level of quality for important software aspects, in the context where the improvements can be made incrementally, the solution is to define quality standards and then establish minimal or maximum thresholds for any software aspect that the team intends to monitor.

(8) SUSPEND MEASUREMENT: To solve if an issue still needs to be continually monitored after some initial measurements, in a context where the team does not yet have a monitoring system, or the current system is overloaded with other issues, the solution is to put on standby the measurements that already fulfilled their initial goal, are costly to be continuously monitored, or that do not represent a value to the team at that moment.

According to the proposed patterns, the first step towards adopting software analytics practices is to define WHAT YOU NEED TO KNOW. After that, with the purpose of answering the raised issues, the team needs to CHOOSE THE MEANS that will be used for data gathering and analysis. LEARNING FROM EXPERIMENTS can be a way of testing a particular solution that the team is not sure is the best way from a practical standpoint. During the SOFTWARE ANALYTICS PLANNING, the team plans the analytics activities and prioritizes the tasks in their to-do list along with other development tasks. Because analytics activities can be time-consuming, the team does not have to deploy them at once. Then, the team can set SMALL STEPS FOR ANALYTICS, according to delivery schedule. Based on actionable insights, the team needs to define REACHABLE GOALS to incorporate the improvements in the software or in its development process. Towards continuous improvement, the team will DEFINE QUALITY STANDARDS to guide their improvement actions. The team can apply the pattern SUSPEND MEASUREMENT when measurements no longer make sense or when they have other priorities at the moment.

2.2 The new Software Analytics patterns

From the difficulties we observed at two software companies when trying to implement software analytics in practice, we identified two new patterns that we now add to our pattern language, and represent in Figure 1 as the two gray blocks with dashed red borders.

The BROAD-SPECTRUM DIAGNOSTIC pattern proposes to conduct a more broad analysis based on common metrics when the team does not have the expertise to understand the kind of problems that software analytics can help to solve, contributing to team learning about existing issues overview and awareness about the analytics process. As shown in Figure 1, there are two arrows that connect this pattern to the WHAT YOU NEED TO KNOW pattern. The upper one represents the team's shift in approach when they do not know where to start, or aren't aware of existing issues, or are not yet convinced of the need for software analysis. The bottom arrow refers to

the move the team makes when it already has a BROAD-SPECTRUM DIAGNOSTIC and now the team can move forward using the WHAT YOU NEED TO KNOW pattern to appoint the most pressing issues to solve.

The EMBEDDED IMPROVEMENTS pattern suggests adding improvement tasks as part of the team routine to guarantee a continuous improvement process. That pattern is an alternative to implement SMALL STEPS FOR ANALYTICS and to incrementally achieve the REACHABLE GOALS, being helpful especially when the tasks related to implementing improvements defined in the analytics process are neglected in planning and frequently left out of the iterations.

3. BROAD-SPECTRUM DIAGNOSTIC

Introducing Software Analytics to software teams can be challenging, especially when those teams have low experience with analytics practices and haven't adopted any in their projects. They can have trouble understanding the extent to which they would benefit from such practices, or the kind of analytics results that one can learn from. This can make them abandon the use of analytics at the start or not correctly use Software Analytics.

3.1 Problem

The team doesn't have an understanding of the type of questions that can be answered through software analytics.

Even though Software Analytics practices help solve code quality issues and improve software quality based on code metrics and insights of those metrics, teams that have no experience with these practices often have difficulties understanding what issues can be solved, verified, or improved with the help of metrics and their analysis. This can make it hard for a team to figure out our WHAT YOU NEED TO KNOW as it might be clear to them the kind of things that *can* be known. This may make teams stuck at the very start of trying to adopt Software Analytics practices. It can make teams can lose interest and their perceptions of the usefulness of these practices can decrease, which can make them abandon the use of analytics in their development process.

3.2 Solution

Use tools that make a general diagnostic of the system and show a couple of metrics to help detect a set of initial issues that can be used as a starting point for the software analytics practices' usage.

Start analytics adoption by providing team members with some examples of metrics and issues that can be solved or verified in their projects through Software Analytics. Firstly, choose a general area of concern in which the team believes it might be beneficial to have the application evaluated, such as code quality, test coverage, or runtime execution. Secondly, choose a tool that can collect information and perform a more general assessment of the system in the desired area.

For instance, to have a vision of the overall quality of the system, a static analytics tool, such as SonarQube, can detect potential problems and provide a few metrics, like cognitive and cyclomatic complexity, code duplication, deprecated methods used, or possible security vulnerabilities. Additionally, dynamic analysis tools can also be of use, for example, some cloud providers provide services to collect data from the application runtime environments and provide tools to access and inspect such information.

By looking at this more general and broad diagnostic performed by the default configuration of such tools, the team will gain a practical understanding of what issues can be solved and what insights can be gained from metrics. This analysis should be taken carefully because some general-purpose metrics tools show the same kind of metrics for every project, so it will be up to the team to decide which metrics and issues are more helpful within their context. Many tools give severity levels to the issues they identify, and looking at the most critical ones can be a good starting point.

After improving the understanding of what metrics can be obtained, and the kind of issues that they allow to identify, developers will be in a better position to figure out WHAT YOU NEED TO KNOW Having in mind some of the

issues found in this general analysis, the team may now be able to follow the rest of the analytics process and further improve their understanding of how to use its practices; the patterns in Section 2 can guide them to solve different kinds of problems.

3.3 Consequences

As a consequence, the team members can learn more about Software Analytics and better understand which metrics they can adopt as part of a measurement plan to handle the issues raised as significant. Additionally, it also helps the team to understand how to use the analytics as part of their development and improvement process. In the end, the team will already have a tool integrated with the environment, which can be useful in the future because some metrics are already available to analyze. This tool can be configured and tuned for more specific analyses that will fit better in the project interests.

A negative consequence is that the tool's report output can guide the team in the wrong direction; a superficial analysis of the report can lead them to look to metrics and issues that are not really relevant to the project, cause them to lose interest, or lead them to wrong conclusions.

3.4 Related Patterns

CONTINUOUS INSPECTION [Merson et al. 2013] is about how to detect architecture and code problems as soon as possible by prescribing the use of available automated tools to continuously inspect code and generate a report on the overall code health. The analysis performed in the BROAD-SPECTRUM DIAGNOSTIC should not be included by default in the CONTINUOUS INSPECTION, and that can be done after to DEFINE QUALITY STANDARDS.

SYSTEM QUALITY DASHBOARDS [Yoder and Wirfs-Brock 2014] is useful to show real-time results and display quality values measured during check-in or system build quality tests. Since the goal of BROAD-SPECTRUM DIAGNOSTIC is to be a starting point, its results should not be integrated by default into this kind of dashboard.

After having a broad view of potential issues through the BROAD-SPECTRUM DIAGNOSTIC, the next step would be to define WHAT YOU NEED TO KNOW [Choma et al. 2017] to focus on the most pressing issues and those that will add the most value to the project.

3.5 Known Uses

—We have ourselves applied the pattern in an industrial software project. The context was that of a recent project within a large company, with a team of 3 developers and various clients governing the project and giving feedback. We understood that the team did not have experience with software analytics. Most decisions were not data-driven, they were based on personal experience and on feedback from product owners and clients.
We introduced the Software Analytics practices to the team, but team members struggled to understand the objectives and issues that could be solved using analytics. Namely, on our first try, they did not come to any conclusion regarding WHAT YOU NEED TO KNOW.
We then decided to use another approach. We started using a metrics tool—SonarQube—and showed the team its report, with all the issues it pointed out and some interesting metrics to promote discussion. By analyzing the importance of each of these issues and metrics, it was easier to understand what could be found by the use of software metrics and start a discussion on WHAT YOU NEED TO KNOW given the specific context of this project. With this approach, we were able to move forward, and the team started to bring more issues to the table and understand better the objectives of Software Analytics.

—To identify success factors that help teams to create better deliveries in future releases and failure factors that help teams to prevent bad deliveries, Huijgens et al. [2017] carried out in an exploratory study in an international bank with more than 300 teams and about 750 different applications. In this study, they defined a limited set of software metrics focused on a delivery scope (*e.g.*, epics, user stories). However, to define the most relevant lagging metrics and related strong leading metrics they need to explore other data sources.

—In a project conducted in a Brazilian company, the team already had SonarQube installed in their environment but did not use it regularly. When asked about relevant issues to be handled by the analytics process, the team did not know precisely what kind of issue they could investigate. However, by looking at the result generated by default by SonarQube, the team was able to quickly identify some issues to be further investigated. The information provided by the tool was not enough to solve the issues, but it was important for the team to understand the kind of issues that could be interesting to address.

4. EMBEDDED IMPROVEMENTS

Also know as *Embed Improvements in Tasks*.

Improvement tasks that emerge from the use of software analytics are often large and difficult to estimate. This type of task can often appear daunting to start and can be left untouched in the backlog. In legacy systems, this type of issue might not even be worth fixing since the codebase is old or the quality has not been the main focus, and most developers might not have worked on that specific part of the system.

4.1 Problem

Software analytics practices often generate technical improvement tasks that imply a considerable effort and, therefore, get frequently left behind during planning and may never end up being implemented.

There usually is a lack of understanding from the end-users and product owners, who are responsible for the product direction, regarding code-quality issues. They may not be able to identify internal quality concerns and understand the importance of dedicating some effort to addressing them. Taking on considerably large technical improvements may not allow each iteration to deliver the desired amount of business value.

Moreover, iteration planning often favors tasks with a clear business value, and other kinds of improvement may remain in product backlogs, postponing improvements that may be much-needed but difficult to prioritize by non-technical stakeholders.

Additionally, there can be a significant effort in some tasks derived from software analytics findings. The team can try to split such tasks into smaller ones and implement them over time with Small Steps for Analytics. Although this might be effective, it might not be enough if future developments make the same issue resurface and, sometimes, the issue might simply be too complex to be divided into smaller tasks. The task is indeed being resolved, but this doesn't necessarily imply any plans to prevent the issue from happening.

4.2 Solution

Change the development practices to embed improvements gradually and continuously in other tasks.

Changing the development practices is essential for this pattern to work, and there are two complementary ways a team can adopt this pattern.

This first way is when a team changes their DoD (Definition of Done) [Madan 2019] to prevent an issue from happening again. The DoD is a checklist that needs to be completely done before considering a task as completed. The team will have the responsibility to make sure that the issue is not happening before closing a task. This way, the issue does not need to be brought up during the planning. It will simply be an underlying requirement when doing a specific task. The team must be aware that the issue will still be present in what was previously done and should consider if it also needs to be resolved. As an example, the DoD might require a minimum value for a test coverage metric, if the goal is to improve the tests.

Another way is to define guidelines to be followed by new features and perform refactoring before changing an existing code to introduce the proposed guideline. That will prevent the problem from appearing in new code and, at a small pace, will migrate the existing one to the new approach. That also avoids having an enormous refactoring task only focused on performing the change. To exemplify this alternative, if the team defined migration

to a new library, refactoring can be added as a task from a User Story that would require a change in this code. Moreover, any new development should be done using the new library.

4.3 Consequences

There are several consequences when using this pattern. On the positive side, is the guarantee that the problem won't resurface. Considering the issue on new developments or including a specific check on the DoD, prevents a particular issue from happening again.

Another positive consequence is that there is no need to create additional tasks to fix or improve an issue. As stated before, tasks related to significant issues can be daunting to start. Considering the improvement as part of typical developments, it won't feel like a substantial endeavor, and progress, although small, will always be made toward the end goal. On the other hand, it depends on a disciplined team that will be sure to follow the DoD established by the team.

On the negative side of the consequences, time management is one of the main problems. The team might take more time to finish a task since there are more things to consider. It may reduce the team's velocity in the short run, and the product owners and clients might notice the change. On the other hand, it may increase the team's velocity in the long run, as the improved code quality may allow future developments to be quicker to implement. The team must evaluate if refactoring makes sense at any particular stage, and contract technical debt if that is indeed the better option.

Another consequence that can arise from subsequent uses of this solution is the cluttering of the *DoD*. It might start to be cluttered with small steps to prevent multiple issues that stemmed from the use of the SA Canvas. Although, in a way, this is positive for the system quality, for the developer might be hard to consider everything before closing a task or User Story. To avoid that, these Embedded Improvements should be incorporated by the team naturally and become part of their day-by-day practices.

An additional consequence is that although this prevents the issue from happening again, the issue persists on older code and needs to be taken care of to eliminate the system's issue. If a team wants to eliminate the process, they have to create tasks to address the issue on older code incrementally, which is still a problem if the issue is large and complex.

4.4 Related Patterns

INTEGRATE QUALITY [Yoder et al. 2014] is about how to incorporate quality assurance into the software process by including a lightweight means for describing and understanding system qualities. The use of EMBEDDED IMPROVEMENTS is an approach to implement that, but it is not the only one, since INTEGRATE QUALITY is more general.

REACHABLE GOALS [Choma et al. 2017] pattern helps teams identify achievable goals before planning and implementing their actions. While this pattern focuses on an approach to establishing objectives for the team, EMBEDDED IMPROVEMENTS proposes an approach to achieve them. In this pattern language, SMALL STEPS FOR ANALYTICS [Choma et al. 2017] is also related to this one, however EMBEDDED IMPROVEMENTS is a more specific solution in that direction.

QUALITY STORIES [Yoder et al. 2014] recommends creating stories that specifically focus on some measurable quality issues of the system that must be achieved, which can be useful to the team for prioritizing and including these quality items on the backlog. That can be considered a competing approach to EMBEDDED IMPROVEMENTS, since it proposes to hide these activities embedding them into existing tasks.

4.5 Known Uses

—We have ourselves applied the pattern in an industrial software project. The context was that of a start-up that had come to accumulate significant technical debt and was dealing with different technical challenges. Fixing some of the issues that the teams identified was a considerably large endeavor.

We found EMBEDDED IMPROVEMENTS useful in the context of this start-up in more than one occasion. The teams were challenged with many issues that they already knew existed but still hadn't formally organized and decided how to address. Some of these issues needed considerable time to resolve. One of them was related to dead code or code that wasn't being used anymore. With a simple script that analyzed the server logs, the team managed to find out that 70% of their core application endpoints were not being called anymore. Removing a large portion of code like this implies a few risks, so tackling it in smaller segments was one of the solutions that were put into practice (*i.e.*, SMALL STEPS FOR ANALYTICS). However, the team also wanted to prevent the issue from reappearing, so they started analyzing the possibility of generating dead code in new refinements. If this was the case, the portion of code in question should be removed to prevent leaving dead code in the repository. Another use of the pattern is related to test coverage. Some older projects had low coverage, but adding tests to all components that were still missing was very hard since the code had been done years ago, sometimes by developers who had since left the company. Since creating all these tests was not an option, the team set a goal that all new code should have 100% coverage. This decision is expected to make code coverage increase over time.

—Snyder and Curtis [2018] reported how software analytics were used to guide improvements and evaluate progress during an Agile and DevOps transformation in a software company. They reported that to detect structural-quality flaws and produce the analytic measures, the teams began scanning their builds at a minimum of once per sprint (every two weeks), enabling them to address the most critical issues before release. By scanning several times a week or even daily, the team could fix critical defects in a day or two, rather than waiting until the next sprint.

—The lack of tests, evidenced by a low code coverage metric, was identified as a problem by a development team of a Brazilian company. To address it, the team decided to create more tests for some specific classes and incorporated a new task in the backlog with such an objective. However, this task remained in the backlog after a few planning sessions and was never considered to have enough priority for inclusion in one of the iterations. A reason for this is that it was a task that required considerable effort to be completed. The team then decided to embed the improvement of test coverage in other user stories—the code created or changed in the context of each user story would have to have the desired code coverage. The desired coverage for the system as a whole was not reached immediately, but the code coverage finally started to improve in the next iterations.

5. SUMMARY

This paper presented two new patterns to compose the Pattern Language for Software Analytics. The complete set of patterns includes ways of incorporating software analytics activities within software development projects. The pattern BROAD-SPECTRUM DIAGNOSTICS was identified for the scenario that we faced when the team did not understand the kind of problem that they could solve using analytics, *a priori*. While the pattern EMBEDDED IMPROVEMENTS refers to embedding improvements gradually and continuously by adopting a checklist of tasks to ensure they are done or considering the issues in engineering refinements as a new task or user story.

6. ACKNOWLEDGEMENTS

The authors would like to thank Uwe Zdun who graciously shepherded this paper for PLoP 2021, Justus Bogner who provided insights on earlier versions of the work, and, finally, all the participants of the Napa writers' workshop at PLoP 2021—Richard Gabriel, Joe Yoder, Jonathan Edwards, Christopher Hartley, Tomas Petricek, Michael Weiss and Daniel Pinho. Thank you for generously contributing to the paper through discussion, and by providing insightful feedback that helped us to greatly improve our work.

We would also like to thank the companies who we have worked with, and that made it possible for us to identify these patterns, as well as the Integrated Masters in Informatics Engineering of the Faculty of Engineering of the University of Porto, for supporting this work.

REFERENCES

Vinay Augustine, John Hudepohl, Przemyslaw Marcinczak, and Will Snipes. 2018. Deploying Software Team Analytics in a Multinational Organization. *IEEE Software* 35, 1 (2018), 72–76.

Raymond PL Buse and Thomas Zimmermann. 2010. Analytics for software development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, Santa Fe, New Mexico, USA, 77–80.

Joelma Choma, Eduardo M Guerra, Tiago Silva da Silva, Luciana AM Zaina, and Filipe Figueiredo Correia. 2019. Towards an artifact to support agile teams in software analytics activities. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE)*. KSI Research Inc, Lisbon, Portugal, 88–93.

Joelma Choma, Eduardo M Guerra, and Tiago S Silva. 2017. Patterns for Implementing Software Analytics in Development Teams. In *Proceedings of the 24th Conference on Pattern Languages of Programs*. ACM, Vancouver, Canada, 12.

Joelma Choma, Eduardo M Guerra, and Tiago S Silva. 2018. Learning from Experiments, Define Quality Standards, Suspend Measurement: Three patterns in a Software Analytics Pattern Language. In *Proceedings of the 12th Latin American Conference on Pattern Languages of Programs (SugarLoafPLoP)*. ACM, Valparaíso, Chile, 10.

Thomas H Davenport. 2009. Make better decisions. *Harvard business review* 87, 11 (2009), 117–123.

Thomas H Davenport. 2010. *How organizations make better decisions*. Technical Report. International Institute for Analytics.

Ahmed E Hassan and Tao Xie. 2010. Software intelligence: the future of mining software engineering data. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, Santa Fe, New Mexico, USA, 161–166.

Hennie Huijgens, Robert Lamping, Dick Stevens, Hartger Rothengatter, Georgios Gousios, and Daniele Romano. 2017. Strong Agile Metrics: Mining Log Data to Determine Predictive Power of Software Metrics for Continuous Delivery Teams. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 866–871. DOI:http://dx.doi.org/10.1145/3106237.3117779

Hennie Huijgens, Davide Spadini, Dick Stevens, Niels Visser, and Arie van Deursen. 2018. Software analytics in continuous delivery: a case study on success factors. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, Oulu, Finland, 25.

Sumeet Madan. 2019. DONE Understanding Of The Definition Of "Done". (Dec 2019). https://www.scrum.org/resources/blog/done-understanding-definition-done

P Merson, A Aguiar, E Guerra, and J Yoder. 2013. Continuous inspection: a pattern for keeping your code healthy and aligned to the architecture. In *Proceedings of the 2nd Asian Conference on Pattern Languages of Programs (AsianPLoP)*. ACM, Tokyo, Japan, 6–8.

Romain Robbes, René Vidal, and María Cecilia Bastarrica. 2013. Are Software Analytics Efforts Worthwhile for Small Companies? The Case of Amisoft. *IEEE Software* 30, 5 (2013), 46–53.

Antje Schmitt, Kathrin Rosing, Stephen X Zhang, and Michael Leatherbee. 2018. A dynamic model of entrepreneurial uncertainty and business opportunity identification: Exploration as a mediator and entrepreneurial self-efficacy as a moderator. *Entrepreneurship Theory and Practice* 42, 6 (2018), 835–859.

Barry Snyder and Bill Curtis. 2018. Using Analytics to Guide Improvement during an Agile–DevOps Transformation. *IEEE Software* 35, 1 (2018), 78–83.

M-A Storey. 2016. Lies, damned lies, and analytics: Why big data needs thick data. In *Perspectives on Data Science for Software Engineering*, Tim Menzies, Laurie Williams, and Thomas Zimmermann (Eds.). Morgan Kaufmann, Boston, 369–374.

Joseph W Yoder and Rebecca Wirfs-Brock. 2014. QA to AQ part two: shifting from quality assurance to agile quality:" measuring and monitoring quality". In *Proceedings of the 21st Conference on Pattern Languages of Programs*. ACM, Monticello, IL, USA, 1–20.

Joseph W Yoder, Rebecca Wirfs-Brock, and Ademar Aguiar. 2014. QA to AQ: Patterns about transitioning from Quality Assurance to Agile Quality. In *Proceedings of the 3rd Asian Conference on Pattern Languages of Programs, Tokyo, Japan*. ACM, Monticello, IL, USA, 12.

Dongmei Zhang, Yingnong Dang, Jian-Guang Lou, Shi Han, Haidong Zhang, and Tao Xie. 2011. Software analytics as a learning case in practice: Approaches and experiences. In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*. ACM, Lawrence, Kansas, USA, 55–58.

Dongmei Zhang, Shi Han, Yingnong Dang, Jian-Guang Lou, Haidong Zhang, and Tao Xie. 2013. Software analytics in practice. *IEEE software* 30, 5 (2013), 30–37.

Received June 2021; revised September 2021; accepted February 2022