# 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

Software analytics refers to the use of software data, analysis, and systematic reasoning to make better decisions regarding the development process, products, and services [Zhang et al. 2011]. Analytics can help answer important questions within software development contexts by assisting software practitioners in extracting insightful and actionable information from the software data sources (e.g. source code, software requirements specifications, bug reports, commit history, test cases, log files, and user feedback) [Buse and Zimmermann 2010] [Hassan and Xie 2010]. The typical issues addressed to software analytics are related to software failures and defect prediction, software requirements, code quality, releases and code integration, project management, teamwork and collaboration, software maintenance, and software evolution, among others [Storey 2016].

Although widely adopted by software companies, software analytics is rarely explored to its full potential. Some studies have lead to the proposal of analytics methods and tools, but few of them provide details on the 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].

Considering these studies on software analytics and others related to software quality, metrics and measurements, we proposed a set of patterns to support the adoption of software analytics by software practitioners in their projects, iteratively and continuously [Choma et al. 2018; Choma et al. 2017]. Based on these patterns, we designed a canvas-format artifact named Software Analytics Canvas (SA Canvas) to support software practitioners in planning and managing software analytics activities [Choma et al. 2019]. By evaluating the proposed artifact in practice, we have identified new software analytics patterns. In this paper, we present two patterns that focus on how to make the team more aware of the kind of information it can generate through the use of metrics and how the team can turn analytic findings into attainable tasks. These patterns have been identified from case studies in two software companies.

One of the companies that participated in the study is a large company with multiple projects and strictly-structured teams. Given its business domain, the company takes security and data privacy particularly seriously. This study was conducted with a small team from a recent project, in which were involved 4 developers, where 3 of them had no experience and knowledge of software analytics. The study was made during three sprints equalizing to 12 weeks.

The second company involved in the study was a start-up. Start-ups face very high business uncertainties and need to adapt quickly [Schmitt et al. 2018]. When the teams need to make rapid changes and decisions, assessing the technical debt that a change will generate becomes extremely valuable for these companies. This start-up has two development teams where both used the SA Canvas during 5 sprints of development, equating to ten weeks.

## 2. SOFTWARE ANALYTICS PATTERNS

This section presents a set of software analytics patterns that we proposed for adopting software analytics practices in agile projects. As previously mentioned, the 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 want 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 need 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.

(3) PLAN ANALYTICS IMPLEMENTATION: To solve how to schedule the software analytics activities fitting them to 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 to perform 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 need 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.

### 2.1 Pattern Language Overview

An overview of the SA patterns showing how they relate to each other is depicted in Figure 1. The blocks in black represent the patterns for adopting software analytics. The blocks with black dashed borders represent the expected outputs from the application of the patterns. Questions included among the patterns refer to factor that motivates the application of the pattern.

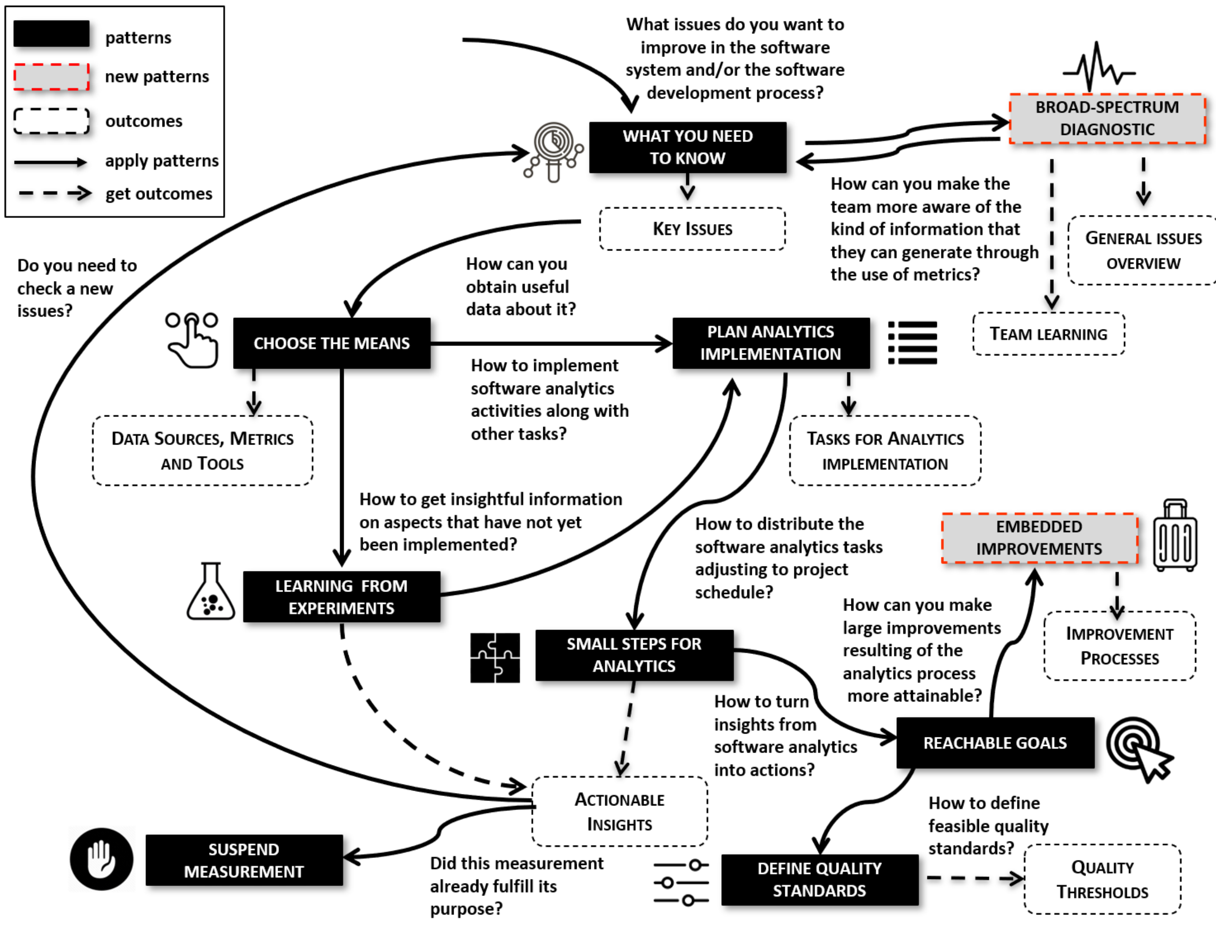
According to the proposed patterns, the first step towards adopting software analytics practices is to define WHAT YOU NEED TO KNOW. After that, with the purpose to answer the raised issues, the team needs to CHOOSE THE MEANS that will be used to data gathering and analysis. LEARNING FROM EXPERIMENTS can be a way of testing a particular solution that the team is not sure if it 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 do not have to be deployed 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 IMPROVEMENT 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.

This paper presents two new patterns which were added to this pattern language. The former pattern named 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, contributing to team learning about existing issues overview and awareness about the analytics process. After a BROAD-SPECTRUM DIAGNOSTIC on existing analytics issues, the team can move forward using the WHAT YOU NEED TO KNOW pattern to identify the most pressing issues to solve. The second pattern named EMBEDDED IMPROVEMENTS 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

**AKA:** *Learning through general diagnostics*
**AKA:** *Learning about metrics*
**AKA:** *General diagnostics for metrics' overview*

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



patterns
new patterns
outcomes
apply patterns
get outcomes
What issues do you want to improve in the software system and/or the software development process?
WHAT YOU NEED TO KNOW
BROAD-SPECTRUM DIAGNOSTIC
KEY ISSUES
How can you make the team more aware of the kind of information that they can generate through the use of metrics?
GENERAL ISSUES OVERVIEW
TEAM LEARNING
Do you need to check a new issues?
How can you obtain useful data about it?
CHOOSE THE MEANS
PLAN ANALYTICS IMPLEMENTATION
How to implement software analytics activities along with other tasks?
DATA SOURCES, METRICS AND TOOLS
TASKS FOR ANALYTICS IMPLEMENTATION
How to get insightful information on aspects that have not yet been implemented?
How to distribute the software analytics tasks adjusting to project schedule?
EMBEDDED IMPROVEMENTS
LEARNING FROM EXPERIMENTS
How can you make large improvements resulting of the analytics process more attainable?
IMPROVEMENT PROCESSES
SMALL STEPS FOR ANALYTICS
How to turn insights from software analytics into actions?
REACHABLE GOALS
ACTIONABLE INSIGHTS
How to define feasible quality standards?
SUSPEND MEASUREMENT
Did this measurement already fulfill its purpose?
DEFINE QUALITY STANDARDS
QUALITY THRESHOLDS

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

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 might **have no experience** with these practices, leading to difficulties **understanding what issues can be solved, verified or improved** with the help of metrics and their analysis. This can get the teams stuck at the start of the adoption of the Software Analytics practices. With these difficulties at the start, teams can **lose their interest** and their perceptions of the usefulness of these practices can decrease which can make them **abandon the usage** of Software Analytics practices in their development process. So, in these cases, it may be important to have a simpler form that provides them with some **examples of metrics and issues** that can be solved or verified in their projects with Software Analytics practices.

## 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 to the software analytics practices' usage.**

Starting with the usage of software analytics practices can be challenging when introducing them to teams with low or no experience with metrics and analytics practices. This is when the BROAD-SPECTRUM DIAGNOSTIC pattern can be used. The first step is to choose a particular area in which the team want the application to be evaluated, such as code quality, test coverage or runtime execution. The second step is to choose a tool that can collect information and perform a more general assessment of the system on the desired area.

For instance, if the goal is to have a vision of the overall quality of the system, a static analytics tool, such as SonarQube, can provide metrics and automated detection of potential problems related to that. Likewise, static analysis tools can also search for potentially vulnerable code that can threaten the application's security. As an example of the usage of runtime metrics, 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 have a more practical understanding of what issues can be solved and what insights can be gained with the help of this information. This analysis should be taken carefully because the tool shows the same metrics to every project, so the team should look at the metrics and issues they find helpful for their context. It is essential to clarify that this general analysis is just a starting point for the team to identify points worth further investigating following a software analytics process. Many of the tools give severity levels to the issues found, and looking at the most critical issues identified by the tools can be a good starting point.

Finally, the team should use the information obtained from the tool report and then use them to identify WHAT YOU NEED TO KNOW. With the issues found in this general analysis in mind, the team can follow the rest of the analytics process, so they can better understand how to use Software Analytics practices and how the patterns described 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 integrate** into their system, **the issues that can be verified and solved** with the usage of Software Analytics practices, but also help them to **understand how to use the analytics** as part of their development and improvement process. In the end, the team will have already 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 analysis that will fit better in the project interests.

A negative consequence is that the tool's report can guide the team in the **wrong direction** because without a good analysis of the tool's results they can be looking to metrics and issues that are not relevant for their project or cause the team to **lose interest** or lead them to **wrong conclusions** based on irrelevant metrics.

## 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 available automated tools to continuously inspect code, 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 have 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

**AKA:** *Improvement Acceptance Criteria*
**AKA:** *Distributed Improvement Tasks*
**AKA:** *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, which 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 to, in each iteration, **deliver the desired quantity of business value**.

Moreover, iteration planning often **favors tasks with a clear business value**, and other kind of improvements 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 still be aware of older developments which still need to be resolved.

Another way one can adopt this pattern is by considering the issues during technology refinements. If the issue exists, it should be addressed when defining a new development or User Story tasks. The solution to a problem the team is trying to solve should already be prepared to avoid making the same mistakes repeatedly. This can be done by defining refactoring tasks before implementing the new development to simplify development while tackling the issue, which facilitates future developments.

## 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 towards 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 on the short run**, and the product owners and clients might notice the change. On the other hand, it may **increase the team's velocity on the long run**, as the improved code quality may allows 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 from this solution is the **cluttering of the *DoD***. It might start to be cluttered with small steps to prevent multiple issues that come from 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.

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 incorporate quality assurance into 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 focus on how an approach to establish objectives for the team, EMBEDDED IMPROVEMENTS propose an approach on how 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 were considerably large endeavors.
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 was put into practice (i.e., SMALL STEPS FOR ANALYTICS). But 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 them was very hard, since the code had been done years ago, sometimes by developers that 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 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 started finally to improve in 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 does not understand the kind of problem that they can solve using analytics, *a priori*. While, the pattern EMBEDDED IMPROVEMENTS refers to embed 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, to Justus Bogner who provided insights on earlier versions of this work, and to the companies who we have worked with and that made it possible for us to identify these patterns. Finally, we would like to thank 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, 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)*.

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, 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 (SLPLoP)*. ACM, 10.

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, 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...* Joint Meeting on Foundations of Software Engineering, ACM, 866–871.

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, 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)*. 6–8.

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*. Elsevier, 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*. 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, 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, 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