Patterns for Implementing Software Analytics in Development Teams

JOELMA CHOMA, National Institute for Space Research - INPE
EDUARDO MARTINS GUERRA, National Institute for Space Research - INPE
TIAGO SILVA DA SILVA, Federal University of São Paulo - UNIFESP

The software development activities typically produce a large amount of data. Using a data-driven approach for decision making – such as Software Analytics – the software practitioners can to achieve higher development process productivity and improve many aspects of the software quality based on insightful and actionable information. This paper presents a set of patterns describing steps to encourage the integration of the analytics activities by development teams in order to support them to make better decisions, promoting the continuous improvement of software and its development process. Before any procedure to extract data for software analytics, the team needs to define, first of all, their questions about what will need to be measured, assess and monitored throughout the development process. Once defined the key issues which will be tracked, the team may select the most appropriate means for extracting data from software artifacts that will be useful in decision-making. The tasks to set up the development environment for software analytics should be added to the project planning along with the regular tasks. The software analytics activities should be distributed through the iterations in order adding information about the system in small portions. By defining reachable goals from the software analytics findings, the team turns insights from software analytics into actions to improve incrementally the software characteristics and its development process.

Categories and Subject Descriptors: K.6.3 [**Software Management**] : Software development

General Terms: Software Mensurament
Additional Key Words and Phrases: software analytics, decision making, agile software development, patterns, software measurement, development teams.

1. INTRODUCTION

Nowadays, due to the considerable amount of data that are generated during the software development activities, some large software companies are actively working to make their development processes data-driven [Kim et al. 2016] [Baysal et al. 2013a]. Thus, the practitioners have the chance to seek insight from data collected in their projects in order to improve progressively and continuously the software quality such as reliability, performance and security [Bird et al. 2013]. In this context, Software Analytics (SA) is a data-driven approach for decision making that encompasses monitoring, analysis, and understanding of software development data mined from different artifacts to obtain insightful and actionable information. These valuable insights can allow software practitioners to achieve higher development process productivity, and improve many aspects of the software quality, including a good user experience [Maalej et al. 2016] [Zhang et al. 2011].

There are many computing technologies such as pattern recognition, machine learning, and data mining which can to enable software practitioners to perform an efficient data exploration. However, the visualization and interpretation of insights are key in the SA to facilitate decision making. And, one the greatest challenges consists

This work is supported by CNPq (grant 445562/2014-5) and FAPESP (grant 2014/16236-6)

of finding out on how to leverage potentially large amounts of data into real and actionable insights for making decisions at different levels – strategic, tactical and/or operational [Buse and Zimmermann 2012].

More studies in-depth are needed to provide not only appropriate techniques and tools for assist those who make critical decisions in software projects, but also to investigate which are their information needs, according to [Buse and Zimmermann 2012]. In addition, the data analysis should to be focused on data that are really useful rather than data that are convenient to collect [Shull 2014]. Therefore, the practitioners need define what data are really needed before beginning collection in order to avoid any delays in delivery and, at the same time, any wastage of resources.

Some patterns have been identified in the field of data analysis – e.g., pattern designed to trace code changes from user requests – CONCEPT TO COMMIT [McGrath et al. 2013], pattern about significance testing and effect size – EFFECT SIZE ANALYSIS [Giger and Gall 2013], and patterns for cleaning up invalid bug data – LOOK OUT FOR MASS UPDATES and OLD WINE TASTES BETTER [Souza et al. 2013]. However, we have noted that there is still a lack of patterns on how to integrate software analytics in industrial practice. This paper presents a set of patterns describing steps to encourage the use of SA by development teams. We present below a brief description of each pattern as patlets.

(1) WHAT YOU NEED TO IMPROVE: adopting a data-driven approach for decision making, the team can define the issues to guide them in the identification of what kind of data should be extracted in order to obtain meaningful information about the system, process and/or usage data.
(2) CHOOSE THE MEANS: the team can choose the metrics, methods and tools that will be used to extract and analysis of data in order to inform their decision making.
(3) PLAN ANALYTICS IMPLEMENTATION: with the purpose of highlighting useful information and drawing conclusions from it, the team can add the tasks related to the software analytics to their task list to be prioritized with the regular project tasks.
(4) SMALL STEPS FOR ANALYTICS: this pattern suggests to adjust granularity of the analitycs activities to the size of each iteration in order not to impose excessive demands to the teams.
(5) REACHABLE IMPROVEMENT GOALS: this pattern suggests small steps for implementation of the improvements based on software analytics findings.

We have outlined these patterns from results of a systematic mapping that we performed in order to investigate the current trends in the use of SA for decision making in software development context, and identify the main issues that are commonly addressed in this research field. Therefore, the main target audience for these patterns are the software practitioners, such as project managers, analysts and software developers – taking into account the different levels of decision-making. In the rest of the paper, we describe the proposed patterns in more detail.

2. WHAT YOU NEED TO IMPROVE

Also known as Find Important Decisions, Highlight your questions, Find Your Questions, What You Need to Know

Development teams know that metrics and other kinds of information can be extracted from their systems in order to support decision making. Development activities generates a large amount of data. Various artifacts including source code, bug reports, commit history, test executions, etc. could provide valuable insights about software project. There are several tools that can extract such data from the development environment at runtime. However, it is common that even development teams that have them available might not know how to use them.

What issues do you need to improve on the software system, the software process, or with respect to user experience?

—The tools that collect raw data about a software system can be time consuming to install and configure.

—The tools can generate a huge amount of data by default.

—With a huge amount of data about the system, development teams might not know where they should focus.

—Several teams do not using data analysis to inform their decision making.

Therefore:

Define the key issues that the development team want to focus on, in order to guide their selection of the appropriate means for measurement, assessment and monitoring these issues throughout the project.

The team frequently has some decisions that they need to make and sometime they are made mostly based on developer's intuition. The team should identify important issues that they consider that they do not have enough information to solve them. Such issues can be related to the structure of the source code, the development process or the business rules. By raising such issues, practitioners can use them as a guide to define what data should be extracted from the system in order to obtain meaningful information to make their decisions related to these issues.

Some decisions might lead to one-time action, for instance, when the team need to prioritize the implementation of an architectural component to improve the system performance. Or it might be a series of continuous decisions and actions that need to be performed through the iterations, such as for what part of the system do I need to prioritize refactoring.

As an example, imagine that a development team wants to improve their tests and need to decide where in the system they should put their effort. Using WHAT YOU NEED TO IMPROVE, a possible question highlighted by development team might be "What data is required to verify software test adequacy?". Answering this question, the team can set up the development environment according to needed data.

As a consequence, the team will understand the reason behind the data being collected, making a better use of them. Additionally, unnecessary data will not be collected and will not take away the focus of the team on what is more important.

A bad consequence of this pattern is that tools can detect unexpected problems based on measurements that does not have a known reason. Focusing only on a subset of that information, the team can fail to notice a potential problem.

◇ ◇ ◇

The pattern FIND ESSENTIAL QUALITIES [Yoder and Wirfs-Brock 2014] is linked to this pattern. However, the related pattern focuses specifically on software quality, while this pattern also encompasses other issues inherent in the software, its development process and business requirements. For instance, as the supporting to strategic and tactical decisions, a need for a reduction in overwork of developers. Therefore, software analytics is not just for quality attributes. After choosing to make the WHAT YOU NEED TO IMPROVE the team need to CHOOSE THE MEANS to fulfill this needs.

Regarding the known uses, Jian-Guang Lou et al. formulated questions about incident-management as a software-analytics problem [Lou et al. 2013]. For them, incident management of an online service requires the service provider to take actions immediately to resolve the incident, since cost of each hour's service downtime is high. As the use of debuggers to conduct diagnosis on services is usually impracticable, the teams need to highlight other questions in order to detected anomalies and quality issues at runtime of the service.

Robert L. Nord et al. presented a series of questions related to measurement and analysis for software architecture and about how to meet the business goals of software [Nord et al. 2014]. According to them, there is an increasing need to provide ongoing insight into the quality of the system being developed. Thus, the teams questions might be, for instance, about how to improve feedback between development and deployment through means to measure intrinsic quality, value, and cost.

In the case presented by Gregorio Robles et al., the information about the development effort invested in a project was considered a business strategy [Robles et al. 2014]. And, the question highlighted by development team was related to how to obtain software development estimations with bounded margins of error.

3. CHOOSE THE MEANS

Also known as Approach to Answer, Choose the most Appropriate Means, Choose the Right Means

Following the pattern WHAT YOU NEED TO IMPROVE the team may collect information related to issues which need to be trace in order to support their decisions. Often, software practitioners and stakeholder rely heavily on their experience and intuition to make decision when solving problems that have occurred during the project. However, that can lead to wrong path that can have bad consequences in the future. Moreover, there are many information that can be extracted from the development environment at runtime that can provide a concrete evidence and reasons for their decisions.

How can you extract relevant data about the issues that you intent to track during the project?

—The intuition of the development team might not reflect the reality of the software system.

—Decisions based on intuition can be based on false premises.

—Tasks related to non-functional requirements sometimes are not easy to justify to stakeholders without presenting well-founded arguments – based on actual data.

—There are several tools that extracts several kinds of data at runtime from the development environment.

Therefore:

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.

Based on the question that should be answered, the developers can identify data that can provide a concrete evidence to support their decision. This data can come from the source code, from the development environment or from the runtime environment. For instance, to find points that should be modified to improve system performance, they can extract the execution time from the software components. As another example, to find places that need to be prioritized for refactoring, information like commit history for bug correction, object-oriented metrics, and frequency of modification can be considered. From the stakeholders' point of o view, the impact of the addition of a new feature might be more precisely estimated with usage data from similar or related ones.

Based on the sources of data identified, the team needs to find tools and/or approaches that can be used to extract them from the system. There might be ready-to-use tools or, sometimes, the team would need to implement something to retrieve some more specific data from the system. In this moment, the team do not need to implement the extraction, but only to raise possibilities for the extraction.

Sometimes the raw data does not directly provides information to support the decision. You might need to filter, interpreter or combine them to meet your information needs by reflecting on that you want to know. The approach

for this does not need to be defined at this point, but it is important to know what kind of information you need in the end.

Considering the previous example, where the development team wants to improve their tests and need to decide where they should put their effort. Using CHOOSE THE MEANS, developers define that they need to extract testing coverage values and the number of commits that modified each class, since they want to prioritize classes that are highly modified and that have a low test coverage. After that, they find out that they can use a test coverage tool that is available in their continuous integration environment, but they did not find a tool to count the commits for each class. However, they think that it is not difficult to create an script that can extract that information and put in a CSV file. As the last step, an analysis should be made to locate classes with a low test coverage and with a high modification frequency.

As a consequence, the team can have a view of how they can have concrete evidence for a decision. Now, they can consider if it worth to follow the software analytics approach based on the cost of the decision and the penalty of choosing a wrong alternative. Stakeholders that usually do not have experience in software development would be able to understand better technical tasks and their impact.

A bad consequence is that this process might take a precious time from the team and take away the focus from the product itself.

◇ ◇ ◇

The pattern MEASURABLE SYSTEM QUALITIES [Yoder and Wirfs-Brock 2014] is related to this pattern because both deal with quality attributes. As we mentioned earlier, the quality concerning the product is one of the items that can be considered for measurement in this pattern. The most common quality attributes are performance, reliability and usability. However, there are others internal and external attributes related to process, business and/or resources that also can be monitored and measured – e.g., effort, number of coding faults found, cost-effectiveness, communication level, system structure, etc. Some attributes may be relatively easy to measure, while others may be difficult or costly to measure. In addition, the software measurement activities can be either assessment or prediction. After choosing to make the WHAT YOU NEED TO IMPROVE and CHOOSE THE MEANS, the team need to PLAN ANALYTICS IMPLEMENTATION to fulfill their needs.

Regarding the known uses, according to Stella Pachidi et al., the collecting usage data for software development is an important mean to monitor, for instance, which applications are most often used, which features are underutilized, and which features could be improved [Pachidi et al. 2014]. Software usage concerns knowledge about how end-users use the software in the practice, and how the software meet to their needs.

Taking into account that the developers' communication – as contained in emails, issue trackers, and forums – is a precious source of information to support the development process, Luigi Cerulo et al. proposed the extracting this content to support some software engineering tasks for software analytics [Cerulo et al. 2015].

According to Jin Guo et al., software project artifacts such as source code, requirements, and change logs can provide actionable information about which classes are fault prone [Guo et al. 2016].

4. PLAN ANALYTICS IMPLEMENTATION

Also known as Analytics in the Backlog

After the identification on how to extract information from system, the means still need to be implemented. On the one hand, specially on agile teams, the effort of each iteration is prioritized by the stakeholders and a task that is not related to the implementation of the software functionality may consume precious time. On the other hand, a decision making based on actual data could avoid technical debt [Li et al. 2015] and/or optimize the impact of

other tasks. There are several kinds of technical debt. Some of them might come from a a conscious decision, however there can be technical debts that the team is not aware of them. By definition, technical debt is a problem that grows with time, and the effort to handle or correct it also grows with time. Thus, an analytic approach can help to detect and monitor them.

How to implement software analytics along with other tasks, fitting into the project roadmap?

—To install tools and to implement data extraction from the system can be time consuming.
—The project team has a limited time and all tasks related to the extracting and analyzing data from the system will take time from the system implementation.
—Data extracted from the system and development environment can provide valuable information that can avoid rework and possible technical debt.
—The decisions that can benefit from the data extraction might not be necessary to be made immediately.
—To worth the implementation, a software analytics need to add more value than the cost to be implemented.

Therefore:

Add tasks related to the software analytics in the backlog to be prioritized with the regular project tasks.

The tasks for the implementation of the data extraction, filtering and analysis identified when you CHOOSE THE MEANS should be added to the planning. They should be estimated and prioritized. The team should consider that to be implemented, the value added by the information extracted from the software analytics need to be superior from the cost to of its implementation.

The moment when the decision is necessary is important to choose when the analytics should be implemented. For instance, considering a decision that will be necessary to be made in two months, the implementation of analytics to support that decision is not a priority for the next iteration.

Considering the aforementioned example in which developers want information that can help them to focus the effort for improving tests. They estimated the time for installing the tool for measuring test coverage, and it would not consume much time. On the other hand, for the creation of the script to extract the most modified classes and the implementation of the analysis that combine both data would consume a considerable time.

Since the number of classes is not large at the current project phase, the team and the stakeholder decide to add in the next iteration scope only the task for the configuration of test coverage measurement tool. The tasks for creating the scripts and for analyzing the data combined remained in the project backlog, but were not considered a priority at this point.

As a consequence, the implementation of data extraction will be considered in the project planning. Since they are motivated by decisions that should be made in the project context, it would be easier for the stakeholder to understand its need and prioritize appropriately its implementation.

A bad consequence of this practice is that the task for the analytics implementation might remain in the backlog indefinitely, and never be done. To avoid this, the stakeholder need to be aware of how that effort can benefit and bring value to the project.

◇ ◇ ◇

This pattern can be complemented with SYSTEM QUALITY DASHBOARDS [Yoder and Wirfs-Brock 2014] in order to to facilitate the tracking of the software activities, allowing the team to identify and mitigate risks at runtime.

Regarding the known uses, defect prediction is one popular application area of software analytics. Taneli Taipale et al. dealt with the challenges to deploy a defect prediction into practice [Taipale et al. 2013]. In order to solve

this issue, they proposed a defect prediction model and different modes of information representation of the data and the model outcomes, such as a commit hotness ranking, an error probability mapping to the source, and an approach to visualization of interactions among teams.

Antonio Gonzalez-Torres et al. focused on software maintenance issues that require the comprehension of software project details [Gonzalez-Torres et al. 2011]. Thus, they proposed an visual software analytics tool for the exploration and comparison of project structural, interface implementation, class hierarchy data, and the correlation of structural data with metrics, as well as socio-technical relationships.

Minelli and Lanza developed a visual web-based software analytics platform for mobile applications that mines software repositories of apps and uses a set of visualization techniques to present the mined data [Minelli and Lanza 2013].

5. SMALL STEPS FOR ANALYTICS

Also known as Divide Analytics in Iterations

Implementing the metrics and configuring the tools in the development environment can be time consuming. Despite they can be important, the priority is always to develop the target software. Moreover, if the team receive too much information at the same time, they will not be able to interpret it and to do something about it.

How to implement software analytics in a pace that it does not impact in project activities and can be consumed by the team?

—Because of the tight schedule, the team usually does not have much time to implement software analytics tasks.

—Too much information being generated at the same time cannot be easily consumed by the team.

—Some tools have a default configuration that generates a lot of information about the system.

—Sometimes different information that is collected from different points, might be useful for the same question.

Therefore:

Distribute tasks related to the software analytics through the iterations, adding information about the system at small portions to the team.

When you PLAN ANALYTICS IMPLEMENTATION, it should be considered that their implementation should be done in a small pace. On the one hand, you should not left the analytics that answer important questions out of several iterations. On the other hand, the analytics implementation should not take a big part of an iteration.

It should be prioritized the implementation of analytics that collaborate to answer important questions. Specially when different information can be used for the same question, their implementation can be divided in subsequent iterations.

Another side of this is that some tools already have several built-in features that provide different kinds of information for the development team. Despite it can be tempting to have them all with a small effort, this amount of information can take away the attention on what is important. Because of that, it is recommended to configure the tool to collect only the data that is needed, hiding the information that will not be used by the team right now.

Considering the running example, the team decided to implement the SonarQube [1] that can produce a test coverage report integrated in the continuous integration server. This tool can generate many other kinds of information, such as object-oriented metrics and bad practices detection. However, based on this pattern, the team decided to disable the extraction of other kinds of information that would not be used by the team right

[1] https://www.sonarqube.org

now. Considering other goals and questions, the other tool features might be enabled in a small pace in the next iterations.

As a consequence, the team will be able to introduce activities for software analytics without interfering too much in the feature list of each iteration. Additionally, the team will be able to respond and act based on the information generated by analytics without losing their focus.

A bad consequence is that if an analytics can reveal a critical problem, the small rhythm of analytics implementation might postpone the problem detection. Moreover, tools that detect several kinds of problems might also find something relevant that was not predicted or expected by the team.

◇ ◇ ◇

The previous pattern about PLAN ANALYTICS IMPLEMENTATION is related to this pattern, since to distribute the software analytics tasks through the iterations these tasks should be listed in the backlog. Notwithstanding, the main concerns addressed here refer to the proportion each that issue of analytics may be handled without overloading the teams.

Regarding the known uses, Olga Baysal et al. discuss about modern issue tracking systems that provide access to an immense amount of raw information, but which are often irrelevant for certain tasks [Baysal et al. 2013b]. They suggest personalized development tools that highlight only the most important information for developers by reducing information overload.

Pinto et al. proposed a tool that provides architectural compliance checking as part of the continuous integration process [Pinto et al. 2016]. When violations are detected, this tool can lock the integration to the software repository.

Towards to the useful software analytics, Turhan and Kuutti state that a simpler analysis to answer a simpler question provided more actionable insights to the team than the more complex alternative [Turhan and Kuutti 2016].

6. REACHABLE IMPROVEMENT GOALS

Also known as Small Objectives, Measurable Results, Reachable Goals

There will be always something that the metrics indicates that could be better. Just letting the team to work on improvements based on the analytics automated feedback might lead them to act without focus. To have a goal of fixing everything might leave the sensation that the team is not moving forward.

How to turn insights from software analytics into actions to improve incrementally the characteristics in the software system?

—Without a clear goal on how the team should handle the software analytics insights in a given iteration, can make them lose time on thing that are not important right now.
—An ambitious goal that takes time to achieve, might leave the sensation that the team is not making much progress.
—The result from an analytics might point to many places where the team can act.
—If a huge task to fix existing problems, might not be considered a priority because of the time it will consume in the iteration.

Therefore:

Define reachable improvement goals from the software analytics findings, and break the activities down into smaller tasks to fit into iteration.

Based on the feedback received from the analytics, the team should define small actions that can be done. Such actions should have a size that fits in iteration tasks and should have a measurable result. In other words, the change performed by this action should reflect an improvement about analytics findings.

As the first step, the team raised some questions that the analytics contributed to answer. These actions should reflect the decisions that were made based on the data collected. However, these actions should be split in a way that can fit in the scope of an iteration without stopping the features implementation.

It is worthwhile noting that improvement actions can be planned to be carried out in future iterations, hence, they do not need to be immediately implemented. The tests, for example, should be created gradually, and their advances must be continuously measured and monitored.

As a consequence, the development team ends up establishing a culture of continuous improvement. By balancing the amount of work in progress, the team avoids accumulating uncompleted works.

A bad consequence is that the team may not be able to deal with extra amount of work needed to act over software analytics insights and take advantage of its benefits. Additionally, a poorly thought out distribution of work can increases the technical debt rather than reduce it.

◇ ◇ ◇

A related pattern is CHUNKING [Weiss and Mockus 2013] that shows how to conduct an analysis of the set of changes made to a software system over time so as to be able to identify sets of code that have the property that a change that touches a chunk touches only that chunk. This pattern may be useful to help the team coordinate and optimize its improvement actions, since it provides an algorithm to identify uncoupled pieces of software (chunks) that each represent a module on which an individual or a small team can work independently.

Regarding the known uses, Haron and Syed-Mohamad proposed a plug-in for IDE that integrates test coverage, number of defects, number of unresolved defects, defects severity and lines of codes, aiming to provide an analytical view for practitioners to assess and validate the testing results [Haron and Syed-Mohamad 2015].

As to continuously monitoring and measuring activities, Rodrigo Souza et al. noted that improving automated testing tools and using integration repositories are two measures that can improve any project [Souza et al. 2015]. However, they pointed the importance of easier access to up-to-date information about the process, in order to evaluate the impact of yet-to-be-made decisions.

As a practice of continuous inspection, Guerra and Aniche have recommend the use of static and dynamic analysis tools that retrieve information about important quality attributes from the source code, such as test coverage, complexity and decoupling [Guerra and Aniche 2015]. Based on these information, the developers continuously can evaluate and refactoring small portions of code – one piece at a time.

7. SUMMARY

In this paper, we have presented five patterns focused on decision-making through software analytics – an data-driven approach that involves monitoring, analysis, and understanding of data extracted from development process, developers' activities and/or users' needs. Aiming to encourage the use this approach by development teams, the proposed patterns describe steps to integrate the analytics activities into development process. As future work, we intend to validate these patterns. Also, we expect to identify new patterns focusing on how to integrate software analytics in industrial practice.

8. ACKNOWLEDGMENTS

We would like to thank our shepherd Filipe Correia for his valuable suggestions and comments on this paper. We also thank CAPES for the provided scholarship to the first author of this paper.

REFERENCES

Olga Baysal, Reid Holmes, and Michael W Godfrey. 2013a. Developer dashboards: The need for qualitative analytics. IEEE software 30, 4 (2013), 46–52.

Olga Baysal, Reid Holmes, and Michael W Godfrey. 2013b. Situational awareness: personalizing issue tracking systems. In Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 1185–1188.

Christian Bird, Tim Menzies, and Thomas Zimmermann. 2013. 1st International workshop on data analysis patterns in software engineering (DAPSE 2013). In Software Engineering (ICSE), 2013 35th International Conference on. IEEE, 1517–1518.

Raymond PL Buse and Thomas Zimmermann. 2012. Information needs for software development analytics. In Proceedings of the 34th international conference on software engineering. IEEE Press, 987–996.

Luigi Cerulo, Massimiliano Di Penta, Alberto Bacchelli, Michele Ceccarelli, and Gerardo Canfora. 2015. Irish: A Hidden Markov Model to detect coded information islands in free text. Science of Computer Programming 105 (2015), 26–43.

Emanuel Giger and Harald C. Gall. 2013. Effect Size Analysis. In Data Analysis Patterns in Software Engineering (DAPSE), 2013 1st International Workshop on. IEEE, 11–13.

Antonio Gonzalez-Torres, Roberto Theron, Francisco J Garcia-Penalvo, Michel Wermelinger, and Yijun Yu. 2011. Maleku: An evolutionary visual software analysis tool for providing insights into software evolution. In Software Maintenance (ICSM), 2011 27th IEEE International Conference on. IEEE, 594–597.

Eduardo Guerra and Maurıcio Aniche. 2015. Achieving quality on software design through test-driven development. Software Quality Assurance (2015), 201–220.

Jin Guo, Mona Rahimi, Jane Cleland-Huang, Alexander Rasin, Jane Huffman Hayes, and Michael Vierhauser. 2016. Cold-start software analytics. In Proceedings of the 13th International Workshop on Mining Software Repositories. ACM, 142–153.

Nur Hafizah Haron and Sharifah Mashita Syed-Mohamad. 2015. Test and Defect Coverage Analytics Model for the assessment of software test adequacy. In Software Engineering Conference (MySEC), 2015 9th Malaysian. IEEE, 13–18.

Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2016. The emerging role of data scientists on software development teams. In Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on. IEEE, 96–107.

Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2013. Software analytics for incident management of online services: An experience report. In Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on. IEEE, 475–485.

Walid Maalej, Zijad Kurtanovic, Hadeer Nabil, and Christoph Stanik. 2016. On the automatic classification of app reviews. Requir. Eng. 21, 3 (2016), 311–331.

Scott McGrath, Kiran Bastola, and Harvey Siy. 2013. Concept to Commit. In Data Analysis Patterns in Software Engineering (DAPSE), 2013 1st International Workshop on. IEEE, 6–8.

Roberto Minelli and Michele Lanza. 2013. SAMOA-A Visual Software Analytics Platform for Mobile Applications.. In ICSM. 476–479.

Robert L Nord, Ipek Ozkaya, Heiko Koziolek, and Paris Avgeriou. 2014. Quantifying software architecture quality report on the first international workshop on software architecture metrics. ACM SIGSOFT Software Engineering Notes 39, 5 (2014), 32–34.

Stella Pachidi, Marco Spruit, and Inge Van De Weerd. 2014. Understanding users' behavior with software operation data mining. Computers in Human Behavior 30 (2014), 583–594.

Arthur F Pinto, Nicolas Fontes, Eduardo Guerra, and Ricardo Terra. 2016. ArchCI: An Architectural Verification Tool into Continuous Integration. In Proceedings of the 2016 Brazilian Conference on Software: Theory and Practice (CBSoft) – Tools Session. 121–128.

Gregorio Robles, Jesús M González-Barahona, Carlos Cervigón, Andrea Capiluppi, and Daniel Izquierdo-Cortázar. 2014. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 222–231.

Forrest Shull. 2014. Data, Data Everywhere... IEEE Software 31, 5 (2014).

Rodrigo Souza, Christina Chavez, and Roberto Bittencourt. 2013. Patterns for cleaning up bug data. In Data Analysis Patterns in Software Engineering (DAPSE), 2013 1st International Workshop on. IEEE, 26–28.

Rodrigo Souza, Christina Chavez, and Roberto A Bittencourt. 2015. Rapid releases and patch backouts: A software analytics approach. IEEE Software 32, 2 (2015), 89–96.

Taneli Taipale, Mika Qvist, and Burak Turhan. 2013. Constructing defect predictors and communicating the outcomes to practitioners. In 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, 357–362.

Burak Turhan and Kari Kuutti. 2016. Simpler questions can lead to better insights. In Perspectives on Data Science for Software Engineering, Tim Menzies, Laurie Williams, and Thomas Zimmermann (Eds.). Morgan Kaufmann, Boston.

David M Weiss and Audris Mockus. 2013. The chunking pattern. In Data Analysis Patterns in Software Engineering (DAPSE), 2013 1st International Workshop on. IEEE, 35–37.

J Yoder and R Wirfs-Brock. 2014. QA to AQ Part Two: Shifting from Quality Assurance to Agile Quality. In 21st Conference on Patterns of Programming Language (PLoP 2014), Monticello, Illinois, USA.

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.

Received May 2015; revised September 2015; accepted February 2015