Enabling Real-Time Feedback in Software Engineering

Larios Vargas, Enrique; Hejderup, Joseph; Kechagia, Maria; Bruntink, Magiel; Gousios, Georgios

DOI
10.1145/3183399.3183416

Publication date
2018

Document Version
Final published version

Published in
Proceedings of the 40th International Conference on Software Engineering

Enabling Real-Time Feedback in Software Engineering

Enrique Larios Vargas
Software Improvement Group
e.lariosvargas@sig.eu

Joseph Hejderup
Delft University of Technology
j.i.hejderup@tudelft.nl

Maria Kechagia
Delft University of Technology
m.kechagia@tudelft.nl

Magiel Bruntink
Software Improvement Group
m.bruntink@sig.eu

Georgios Gousios
Delft University of Technology
g.gousios@tudelft.nl

ABSTRACT

Modern software projects consist of more than just code: teams follow development processes, the code runs on servers or mobile phones and produces run time logs and users talk about the software in forums like StackOverflow and Twitter and rate it on app stores. Insights stemming from the real-time analysis of combined software engineering data can help software practitioners to conduct faster decision-making. With the development of CodeFeedr, a Real-time Software Analytics Platform , we aim to make software analytics a core feedback loop for software engineering projects. CodeFeedr's vision entails: (1) The ability to unify archival and current software analytics data under a single query language, and (2) The feasibility to apply new techniques and methods for high-level aggregation and summarization of near real-time information on software development. In this paper, we outline three use cases where our platform is expected to have a significant impact on the quality and speed of decision making; *dependency management*, *productivity analytics*, and *run-time error feedback*.

ACM Reference Format:
Enrique Larios Vargas, Joseph Hejderup, Maria Kechagia, Magiel Bruntink, and Georgios Gousios. 2018. Enabling Real-Time Feedback in Software Engineering. In *Proceedings of 40th International Conference on Software Engineering: New Ideas and Emerging Results Track, Gothenburg, Sweden, May 27-June 3 2018 (ICSE-NIER'18)*, 4 pages.
https://doi.org/10.1145/3183399.3183417

1 INTRODUCTION

Decisions in software engineering are typically made in dynamic circumstances [3]. The main effect of the changing nature in software development projects is that *the **time** dimension has to be taken into account explicitly*. For that reason, **feedback** is considered to be one of the most crucial features of dynamic decision tasks and a valuable resource that, if used properly, can facilitate the decision-making process [9].

In an era where many fields of economic production strive for higher efficiency through data-driven decision making, software production has yet to live up to the challenge. Modern software projects are more than just the code that comprises them: teams follow specific development processes; the code runs on servers or mobile phones and produces run time logs; users talk about the software in forums like StackOverflow and GitHub and rate the product in app stores, in blog posts and on Twitter; the software is part of a collection of similar applications and depends on external code or API's to deliver its functionality. To optimize the delivery and the user experience of software, modern organizations need to integrate and combine hundreds of metrics in real-time.

While tools and methods for extracting data from software development processes, products and ecosystems, do exist, three key aspects are missing: *integration*, *composition* and *real-time operation* [8]. As a result, it is challenging for organizations to capitalize on the wealth of data that software projects produce. Consequently, software analytics are seldom integrated as a feedback loop in software projects.

To remedy this situation, we propose the introduction of real-time software analytics as a core feedback loop for software teams. Our hypothesis is that the implementation of CodeFeedr, a Real-time Software Analytics Platform, will represent a significant contribution in the field of software engineering in the following aspects: (1) speeding up decision-making by reducing the time between action and feedback or viceversa, (2) allowing the monitoring of software development infrastructure in real-time and relating production measurements with development actions, (3) supporting the integration of a multitude of data sources that comprise a modern software project, and (4) enabling stakeholders to create up-to-date customized information views of the software development workflow.

2 THE NEED FOR REAL-TIME FEEDBACK

In the field of software engineering, Real-time analytics has been applied in: (1) Autonomous Systems and (2) Adaptive and Self-Managing Systems. However, there has been little research done on real-time feedback analytics applied to the software development life cycle. In this context, we present three use cases where real-time feedback analytics can have a significant contribution.

2.1 Dependency Management

Open source software (OSS) libraries in large centralized code repositories such as `npm` or `Maven` are increasingly becoming more and more interconnected and interdependent. A side-effect of including a highly interconnected library in a project, is that the projects *dependency tree* of *transitive* dependencies can quickly grow large over time. A growing number of *transitive* dependencies can introduce complexities to conventional dependency management and

ICSE-NIER'18, May 27-June 3 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5662-6/18/05…$15.00
https://doi.org/10.1145/3183399.3183417

have recently lead to severe security and trust implications. For instance, the *Equifax*[1] incident leaked over 100.000 customers credit card information due to a critical *security bug* in the Apache Struts library. *Equifax* was not able to update to a patched version of the library in time after the vulnerability was known because it was underestimated the impact. Thus, we identify the following features to lack in dependency management:

The lack of end-to-end visibility of interdependent libraries. Dependencies in a dependency tree change and evolve independently. A dependency tree may look different after a fresh build due to the flexible dependency constraints, for example *semantic versioning* ranges used in `npm`. A dependency can automatically include a more recent version, that may add additional dependencies to the tree without the developer being aware of it. Further, the additional dependencies may also be outdated, removed, include more or new dependencies that may be unstable or have many critical bugs. These different evolution characteristics have to co-exist in a dependency tree, making it overwhelming and difficult for developers to digest. Moreover, developers have little control over *transitive dependencies* and have to accept the decisions or risks taken by other library maintainers.

Difficulty to evaluate the impact and risk associated with a dependency. The active use of dependency checkers and monitoring information feeds allow developer teams to keep up-to-update with bug reports and new releases. This can yield a low signal-to-noise ratio since it is difficult to comprehend the benefits or the urgency of updating a dependency to a newer version. For instance, a dependency may be used through out a software portfolio, a project may use *outdated* dependencies that require major re-write or is in conflict with other used dependencies.

A received bug report only indicates affected versions and not the actual use of a dependency. This makes it difficult to know whether a transitive dependency puts a software project under risk due to a security bug. Developers need to use *subjective judgment* in these scenarios that could have devastating consequences.

Bug or change-impact propagation in an interconnected code repository. A challenging part for library maintainers is to estimate the *damage* (e.g breaking changes) made to clients due to changes made or identified bugs in the library. Understanding how a library is used in other libraries and applications can help maintainers to better understand the *risks* before making changes. This could be helpful in the event of a security bug, solving the bug should be seen as a collaborative effort between maintainers and clients. Therefore, ways to minimize breaking changes for clients could be achieved by understanding how clients directly or indirectly use affected code segment in a library.

The ever-changing nature of centralized code repositories explicitly impacts regular software project at the heart of the dependency level. Therefore, it is important that changes are captured in real-time and those changes are provided as *feedback* to developers and library maintainers. In doing so, we believe that lightweight code analysis can capture the risk and bug propagation across an ecosystem and in dependency trees at the client-level. For instance, library maintainers can identify a potential critical bug in the source coude if they receive at the refactoring stage as a real-time feedback how many clients are directly and in-directly affected by the changes.

[1]https://blogs.apache.org/foundation/entry/apache-struts-statement-on-equifax

2.2 Productivity Analytics

Effective software productivity measurement is becoming a crucial factor for decision-making in the current software development practices, which have the goal to achieve faster application delivery using new infrastructure, tools and methods focused on continuous integration (CI), continuous delivery (CD) or continuous deployment. This scenario calls for switching towards a near real-time or just in time approach for measuring productivity.

We envision that real-time analytics can have a significant contribution in the following challenges in productivity analytics:

Extending the scope of productivity measurement. Most approaches in this domain use only the outputs generated in the coding stage, usually in terms of Source Lines of Code (SLOC), functionalities or story points completed over time. However, we can get a better understanding of stakeholders' work by looking at their interactions at a more detailed level and associating them to a specific stage in the software development.

Absence of environments to improve performance. Meyer et al. [12] highlights that knowing about the current progress of work items and time spent in each activity provide with metrics for self-monitoring and feedback. Additionally, Calvo et al. [4] affirms that there is a strong correlation between self-awareness of one's current state and person's performance or behavior. Current research in this area, such as TimeAware [10], is an example of how real-time self-monitoring systems can contribute to improve personal productivity.

The difficulty in identifying developers' behavior patterns to enhance productivity. During the software development workflow, multiple tools capture real-time data about stakeholder's activities such as local environments, control versioning systems, CI servers, testing environments and CD servers. What is missing is a unified data layer for mining patterns from developers activities data that can be correlated to productivity factors to help tuning productivity measurements.

An immediate and more effective visibility of the interplay between code quality and productivity. Source code is continuously changing in terms of enhancements, fixing bugs and new requirements. It is a challenging task to understand and measure the impact of changes over time. Furthermore, the possibility of introducing an unintended fault or defect during those changes is relatively high. This situation calls for urgent measures to improve source code visualization by aligning near real-time data from the software delivery pipeline with software quality factors and productivity analytics metrics.

Assessing productivity using feedback resulted from aggregating near real-time and archival data can contribute to conduct real-time operational performance monitoring of software delivery pipelines.

2.3 Run-time Error Feedback

As software evolves rapidly, the need for new methods and techniques that can ensure systems and applications' availability becomes more intense. Software reliability is important for critical systems, such as medical devices, smart vehicles, aircrafts and so on, as well as for mobile applications that we use in every day life.

Traditional approaches for developing robust programs include: software verification, prevention mechanisms using programming languages' features, as well as debugging, but these techniques are time-consuming and suffer from false positives.

Next-generation real-time feedback mechanisms are able to effectively support the *productive* development of modern software applications by *reducing* execution failures. Systems based on real-time analytics can give feedback that addresses the following challenges.

Classification of crash causes. To predict and prevent future execution failures there is a need for grasping knowledge from *crowdsourcing* data. Such data include: source code, commits, issues, and crash reports and stem from online sources, such as: Q&A consulting sites (e.g. *stackoverflow.com*), issue tracking systems (e.g. Bugzilla, Jira, GitHub, etc.), and so on. Then, there is a need for *efficient processing* of data from software repositories and crash report management services for *r*eal-time feedback on the classification of common failure reasons and the prevention from similar failures.

A real-time analytics platform can be used in order to enable *real-time integration* analysis of diverse data. Given that these data are in the form of text, natural language processing techniques (e.g. similarity metrics, textual analysis, and parsing) can be applied here. Therefore, such a real-time system can automate the processing of data from a variety of sources improving decision-making.

Recommendations for the prevention of software crashes. For preventing software and its consumers from suffering from future execution failures, researchers have devised algorithms that learn from past software failure patterns and predict possible new crashes. However, currently, it is not easy for developers to use these theoretical methods. There is a need for more *practical* solutions.

A real-time analytics system will provide developers with *alerts* for possible execution failures while they program new software. For this, machine learning and software engineering techniques can be used. Behind the scenes, a prediction model can be applied for the identification of failure prone modules during software production based on learning the bad and good design and coding patterns from source code, crash data, and so on. Additionally, applying program and root cause analysis on appropriate data, an automatic fault localization method can be used for the efficient reproduction and prevention of future software crashes. Thus, a tool such as a plug-in for an IDE that can give real-time feedback regarding the quality of software programs, assisting developers to write more robust software.

Prioritization of bug fixes. Crash reports produced during execution failures are very valuable for understanding and fixing software bugs. However, crash data might include a lot of noise, be incomplete, and hide important information hindering their analysis and comprehension. Therefore, several approaches try to interpret the messages that these reports convey to achieve accurate error recovery. However, nowadays, there is also a need for fixing software problems *as soon as possible*.

A real-time analytics solution can automatically identify patterns in crash data, revealing actual reasons of failures. "Intelligent tools" that are able to pinpoint critical faults, which should be fixed soon, can eliminate developers' effort to conduct manual root cause analysis and guarantee fast as well as precise bug fixing.

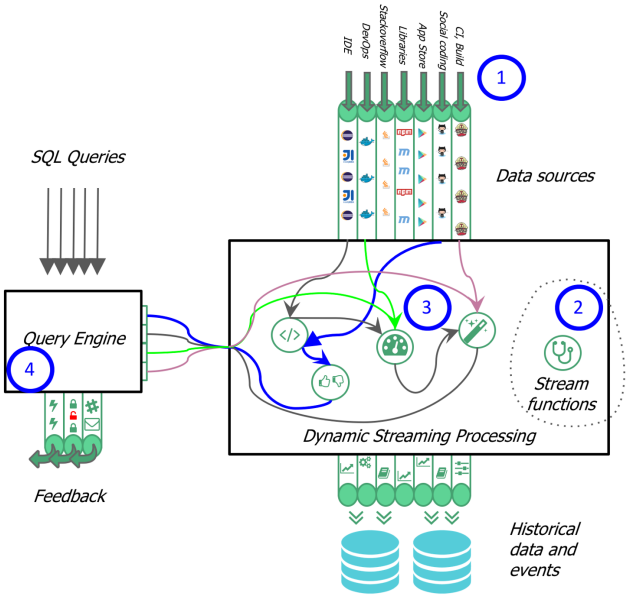

Figure 1: Feedback-driven platform architecture.

3 DESIGN GUIDELINES

The three uses cases show a currently unfulfilled need of software projects: an integrated, near-real time, feedback loop based on software analytics. To fill this gap, we envision a platform that will pursue the following design guidelines:

Provide instant feedback. This will enable DevOps teams to monitor their infrastructure in real-time and relate production measurements with development actions.

Integrate all potential data sources that comprise a modern software project. Contrary to current software big data efforts that prioritize source code or repository analysis, CodeFeedr will integrate all possible data sources, including natural language based ones. Moreover, it will create a process and a data schema that will enable other researchers to integrate arbitrary data sources.

Devise novel ways of aggregation and summarization of software analytics. CodeFeedr will enable various stakeholders (developers, DevOps, managers) to create up-to-date customized information views (textual or even graphical).

4 PRELIMINARY RESULTS

To realize our vision of an integrated real-time feedback loop in software analytics, we present the architecture of the initial implementation of a system to process software engineering data in real-time. An overview of the high-level architecture is presented in Figure 1. In the following numbered paragraphs, we explain the role of each key component for our platform:

1) Stream processing of data sources. To deliver feedback in near real-time, data needs to be ingested, processed and analyzed upon arrival and piped-out in an acceptable time frame. Software processes that trigger events such as *commit to a repository*, *release of an npm package*, *stack trace reports in a created JIRA issue* or *a security advisory* are the *first-class* citizens in the platform. However, certain software processes focus on learning from past data or events. This could be abandoned software projects that are no

longer maintained or learning from past mistakes to improve better bug resolution. Therefore, it is important to be able to replay such data or create stateful models of data sets.

These requirements are ideal for *event stream processing* to deliver a near real-time processing of software processes.

2) Software tool as a stream function. Event stream processing frameworks support business-related functionally such as data aggregation and summarization. In a feed-back driven platform, many use cases would depend on advanced techniques such as *program analysis* to evaluate whether a set of code changes reduces or improves the code quality to a project before committing the changes.

The platform should support the integration of current software tools or techniques as *stream processing functions*. This implies that tools that handle tasks such as *automatic test suit generation*, *type checker* and *taint analysis* will be treated as a function. This also further entails that certain of these tools need to be adapted to process *events* as input data. This is different from processing the entire projects as input data (i.e. the analysis must be incremental).

3) Dynamic data source and function integration. Software processes are ever-changing, and so are data sources and software tools. An identified requirement of our platform is the dynamic integration of new data sources and processing functions (e.g software tools). Further, the data sources and functions should be aware about each others compatibility such that users can compose their stream topology of functions and data sources freely. This implies that data sources and functions need some form of schema or type information to ensure compatibility. As an example, a function that processes commit messages from Github should dynamically recognize and be able to process commit messages from BitBucket.

We are currently building the means to support *dynamic* recognition of data sources and functions, and also being able to *modify* running stream topologies to e.g add Bitbucket data source in an already running Github commit processing topology.

4) SQL query Interface. SQL being a declarative language designed for static data has an expressiveness that could be ideal for processing software data. SQL is the most popular query language, therefore being able to express SQL queries over data streams could make it useful not only for developers but also stakeholders who are more acquainted to work with databases. Here is an example of how a query could be expressed in natural language:

> For [all | 100 recent | 50 most discussed] pull requests on project A, prioritize those that are most important to [me or my team]

The result would be feedback-driven and continuously updated.

5 RELATED WORK

Our approach connects two major research domains: streaming data processing and software repository mining.

The software repository mining community identified early on the need for platforms to analyze data from software repositories on a large scale, for instance, Kenyon [2] and Boa [6]. Additionally, Microsoft developed Codemine, a software development data analytics platform for collecting and analyzing engineering process data, its constraints, and organizational and technical choices [5]. However, all those projects have in common the integration of important archival data sources; and, they do not aggregate any information apart from source code, issues and emails. Furthermore, they also do not use real-time analysis in their data sources.

In the field of mining software repositories (MSR), it is important to mention three cases that inspired our work in the development of CodeFeedr: (1) *GHTorrent's* [7] approach to follow GitHub's event stream processing for events happening in real time on all project repositories across GitHub, instead of mining the repository histories in a static way. On the other hand, (2) *CodeAware's* [1] effort to provide an integrated mechanism for giving early feedback to engineers and to automate follow-up actions. This approach uses a sensor-actuator-based ecosystem for distributed and fined-grained artifact analysis. Furthermore, (3) *data-driven requirements engineering* [11], in which software practitioners could systematically use explicit and implicit user feedback describing user experiences in an aggregated form to support requirements decisions.

6 CONCLUSION

The development of CodeFeedr, a Real-time Software Analytics Platform represents our vision to tackle current challenges for modern software projects. Our platform enables a real-time feedback loop environment in order to: (1) speed up decision-making, (2) monitor software development infrastructure in real-time and (3) create up-to-date customized information views of the software development workflow.

CodeFeedr's architecture facilitates integration, aggregation, analysis and summarization of software analytics data as streams. These features will enable software practitioners to cut across production/run time layers in order to optimize software delivery, performance and quality.

REFERENCES

[1] Rui Abreu, Hakan Erdogmus, and Alexandre Perez. 2015. CodeAware: Sensor-based Fine-grained Monitoring and Management of Software Artifacts. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 551–554.

[2] Jennifer Bevan, E. James Whitehead, Jr., Sunghun Kim, and Michael Godfrey. 2005. Facilitating Software Evolution Research with Kenyon. *SIGSOFT Softw. Eng. Notes* 30, 5 (sep 2005), 177–186.

[3] Janet E. Burge, John M. Carroll, Raymond McCall, and Ivan Mistrik. 2008. *Rationale-Based Software Engineering*. Springer-Verlag, Berlin, Chapter 5, 67–76.

[4] Rafael A. Calvo and Dorian Peters. 2014. *Positive Computing: Technology for Well-Being and Human Potential*. The MIT Press.

[5] Jacek Czerwonka, Nachiappan Nagappan, Wolfram Schulte, and Brendan Murphy. 2013. CODEMINE: Building a Software Development Data Analytics Platform at Microsoft. *IEEE Softw.* 30, 4 (jul 2013), 64–71.

[6] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2013. Boa: A Language and Infrastructure for Analyzing Ultra-large-scale Software Repositories. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 422–431.

[7] Georgios Gousios. 2013. The GHTorent Dataset and Tool Suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, Piscataway, NJ, USA, 233–236.

[8] Georgios Gousios, Dominik Safaric, and Joost Visser. 2016. Streaming Software Analytics. In *Proceedings of the 2nd International Workshop on BIG Data Software Engineering (BIGDSE '16)*. ACM, New York, NY, USA, 8–11.

[9] Jose H. Kerstholt and Jeroen G.W. Raaijmakers. 1997. *Decision Making: Cognitive Models and Explanations*. Routledge, Abingdon, Oxon, Chapter 12, 205–217.

[10] Young-Ho Kim, Jae Ho Jeon, Eun Kyoung Choe, Bongshin Lee, KwonHyun Kim, and Jinwook Seo. 2016. TimeAware: Leveraging Framing Effects to Enhance Personal Productivity. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 272–283.

[11] Walid Maalej, Maleknaz Nayebi, Timo Johann, and Guenther Ruhe. 2016. Toward Data-Driven Requirements Engineering. *IEEE Softw.* 33, 1 (jan 2016), 48–54.

[12] Andre Meyer, Laura E. Barton, Gail Murphy, Thomas Zimmermann, and Thomas Fritz. 2017. The Work Life of Developers: Activities, Switches and Perceived Productivity. *IEEE Transactions on Software Engineering* 43, 12 (dec 2017), 1178–1193.