CHAPTER 16

Software Engineering Dashboards: Types, Risks, and Future

Margaret-Anne Storey, University of Victoria, Canada
Christoph Treude, University of Adelaide, Australia

Introduction

The large number of artifacts created or modified in a software project and the flood of information exchanged in the process of creating a software product call for tools that aggregate this data to communicate higher-level insights to all stakeholders involved. In many projects—in software engineering as well as in other domains—dashboards are used to communicate information that may bring insights on the productivity of project activities and other aspects. Stephen Few defines a dashboard as "a visual display of the most important information needed to achieve one or more objectives which fits entirely on a single computer screen so it can be monitored at a glance" [4].

Dashboards are cognitive awareness and communication tools designed to help people visually identify trends, patterns and anomalies, reason about what they see, and help guide them toward effective decisions [3]. Their real value and one of the main reasons for their popularity is their ability to "replace hunt-and-peck data-gathering techniques with a tireless, adaptable, information flow mechanism" [9]. The goal of dashboards is to transform the raw data contained in an organization's repositories into consumable information. In software engineering, dashboards are used to provide information related to questions such as "Is this project on schedule?"

C. Sadowski and T. Zimmermann (eds.), *Rethinking Productivity in Software Engineering*,
https://doi.org/10.1007/978-1-4842-4221-6_16

and "What are the current bottlenecks?" and "What is the progress of other teams?" [7]. In this chapter, we review the different types of dashboards that are commonly used in software engineering and the risks that are associated with their use. We conclude with an overview of current trends in software engineering dashboards.

The link between productivity and dashboards becomes apparent when investigating one of the dimensions that Few proposes for the categorization of dashboards: *type of measures*. While not always intended this way, much of the quantitative data presented in developer dashboards can also be interpreted as a measure of developer productivity (discussed in more detail in Chapter 15). For example, a bar chart that shows open issues grouped by team can easily be interpreted as a chart highlighting the most productive team (i.e., the team with the least open issues). The relationship between productivity of a development team and the number of open issues is obviously much more complex, as one of our interviewees in a study on developer dashboards confirmed: "Just because one team has a lot more defects than another that doesn't necessarily mean that the quality of that component is any worse" [7]. Instead, a component might have more defects because it is more complex, because it has a user-facing role, or because it is a technically more central component that other components depend on, exposing it to more unexpected conditions.

Few also proposes a categorization of dashboards based on their *role*, in particular discussing dashboards in terms of their strategic, analytical, and operational purposes. In software projects, the use of dashboards for operational purposes is the most common. Such dashboards are dynamic and based on real-time data, supporting drilling down to specific artifacts such as critical bugs in a software project. Dashboards for strategic purposes (so called "executive dashboards") tend to avoid interactive elements and focus on snapshots rather than real-time data.

Software developers produce many textual artifacts, ranging from source code and documentation to bug reports and code reviews. Therefore, it is unsurprising that dashboards used in software projects often combine different *types of data*, i.e., qualitative and quantitative data. A bar graph showing the number of open issues grouped by team would be a simple example of quantitative data, whereas a tag cloud of the most common words used in bug reports is a simple representation of some of the qualitative data present in a software repository.

Another important dimension highlighted by Few is the *span of data*. When creating a dashboard for a software project, many considerations have to be taken into account; e.g., should the dashboard feature enterprise-wide data or just data from a single project (bearing in mind that projects tend not to be independent)? Should each developer have

their own personalized dashboard, or do all dashboards from a project look the same? In addition, dashboards can cover different timespans, such as the entire lifetime of a project, the current release, or the last week. In software projects, one week is not necessarily like any other. For example, development activity during feature or code freeze is expected to be different from the activity when starting to work on features for a new release.

Dashboards in Software Engineering

Within software engineering, dashboards are used to provide information and metrics on the product under development, as well as to display information or to support the analysis of the development process. Typically, they are designed with a specific stakeholder and goal in mind, and many of these goals relate directly or implicitly to some aspect of productivity, including the product quality, work velocity, or stakeholder satisfaction (see Chapter 5).

In the following text, we present some high-level categories of dashboards (those that support individual developers, teams, projects, and communities), alluding to the stakeholders who use the dashboard and to the kinds of tasks they support within each category, as well as where those dashboards tend to be hosted.

We do not aim to be exhaustive but rather to illustrate the myriad of dashboards that are used to support software engineering productivity. Most software engineering dashboards support operational or analytical tasks, while fewer support strategic tasks. Many of these dashboards are static, but more and more, software dashboards are becoming interactive as they play an increasingly important role in how software productivity is understood, measured, and managed.

Developer Activity

Dashboards may be used to display individual developer activity and performance, such as how coding time is spent (authoring, debugging, testing, searching, etc.), how much focus time the developers have in a given time frame, the number and nature of interruptions they may face, time spent using other ancillary tools, coding behaviors (e.g., speed of correcting syntactical errors), and metrics indicating how many lines of code or features they contributed to a repository. This information, when used by the developers themselves, can assist in personal performance monitoring, as well as personal productivity improvements especially when the dashboards allow the

comparison of such information over time. Such dashboards also help developers reveal bottlenecks from the project code itself (which areas they spend much of their coding time on) or from their own development process (see Chapter 22 for another example of a dashboard to increase developers' awareness about their work and productivity).

Codealike is one example of a dashboard service that integrates with a developer's IDE and supports developers in visualizing their own activities showing time spent navigating the Web (if they opt to use an additional web browser plugin), focus and interruption time, coding behavior over time, and coding effort on specific areas of the project code. WakaTime similarly produces dashboards to show metrics and insights on programming activity (such as programming language usage) and supports private leaderboards to allow developers to compete with other developers if they wish (in an effort to be more productive). RescueTime offers interactive features that allow developers to set personal goals and to alert them when they may go off track (e.g., if they spend more than two hours on Facebook, they receive an alert).

In addition to presenting personal productivity information in dashboards, many of these services go beyond that and will also send information on a regular basis to the developers (or other stakeholders) in an e-mail; they may even produce a metric to represent a productivity score (see RescueTime for an example that allows the developers to customize the productivity score), or they may further block web sites in an attempt to improve personal productivity. The primary feature of these services are the dashboards they provide, but we also see that they start to offer more features that go beyond the restrictive definition of dashboards given by Few.

Team Performance

Although many dashboards are primarily designed for developers to gain insights on their own activities and behaviors, many display or aggregate information across a team for other stakeholders, such as team leads, managers, business analysts, or researchers.

This team-level information may be used to improve the working environment, development process, or tools they use. Many services (such as Codealike) provide specific-team level dashboards showing team metrics and even ranking information across developers. Some services also provide support for teams to actively improve their performance together. However, there is concern that information captured about individual developer behaviors may be inaccurate at capturing all the activities individual developers may do and that the information may be used inappropriately.

Keeping track of and monitoring work at a team level is especially important for distributed teams. The Atlassian tool suite offers dashboards that help not only the individual developers but also the team (see https://www.atlassian.com/blog/agile/jira-software-agile-dashboard) to maintain awareness across the team and to regulate their work at both the individual and team levels [2]. GitHub also supports many dashboards to present project information to teams (as we will discuss). Also, for monitoring, development teams may use task boards for task tracking (such as Trello). Although such task boards are not typically referred to as dashboards, they can be used to give an overview of team performance and support team regulation.

Agile teams use many different tools for tracking project activities as they have to deal with a lot of data to help them manage and reflect on their process, in particular tracking their performance across sprints (e.g., see https://www.klipfolio.com/blog/dashboards-agile-software-development). In agile teams, dashboards especially may play an important role for managers. Managers, who are responsible for keeping track of all things in flight during a sprint, may rely on dashboards that visualize all open issues for a particular project to see who open issues are assigned to and what is the priority of open issues. Burndown charts, shown in dashboards, may show how the team is tracking against a predicted burndown line. Axosoft is another service to support agile teams in visually tracking their progress so that they can plan more accurately.

Teams commonly use TV monitors for displaying dashboards so that the team and managers can maintain awareness at a glance on how sprints are progressing in agile projects, while dashboard services such as the one provided by Geckoboard can be used to show project-level monitoring information on TV screens to help teams focus on key performance metrics.

Project Monitoring and Performance

For showing activity at a specific project level, GitHub, like other repository services, extensively uses dashboards to provide insights to managers, project owners, and other developers who may want to decide on the value of using, depending on or contributing to particular projects (see https://help.github.com/categories/visualizng-repository-data-with-graphs/). Grafana, used by the GitHub Stats monitoring project, visualizes project forks, stars, number of issues, and other project metrics over time. Bitergia also provides many dashboards for visualizing project and organization information pulling data from many diverse tools and integrations.

As many projects nowadays rely on continuous integration and deployment services, many dashboards visualize how code is moving through the pipeline, especially as new features are flighted in A/B testing experiments. Additional DevOps support may be provided by visualizing the performance of running services, tracking outages, etc. (see https://blog.takipi.com/the-top-5-devops-dashboards-every-engineer-should-consider/, https://blog.newrelic.com/2017/01/18/dashboards-devops-measurement/ and https://www.klipfolio.com/resources/dashboard-examples/devops for some discussion on DevOps dashboards).

There are also project-level dashboards that focus particularly on customer management. Zendesk dashboards visualize how customers use specific web applications, as well as how they use their support channels for communicating with the development team, and they visualize satisfaction levels of the end users. Similarly, AppNeta creates dashboards that provide insights on end-user satisfaction with web applications over time. UserVoice also provides dashboards but goes one step further by helping to prioritize customer feedback in the form of a road map to guide future development priorities.

Community Health

Closely related to project-level dashboards, other dashboard services aim specifically at visualizing data at a community or ecosystem level. For example, the CHAOSS web site gathers and visualizes data to support the analytics of community health for open source communities such as Linux. For Linux, the foundation defines interesting health metrics such as number of licenses used among others (see https://github.com/chaoss/metrics/blob/master/activity-metrics-list.md).

Summary

As we can see, the landscape of dashboards that already exist (and could exist) for visualizing software development information is extremely broad and varied. They support a wide array of stakeholders and tasks and are hosted on different media. We also see some dashboards stretching the definition of a dashboard by providing additional features and services. However, we can also anticipate that the power they provide in terms of analytics introduces some risks, which we discuss next.

Risks of Using Dashboards

Despite their usefulness to turn repository data into consumable information, dashboards come with a number of risks. Indeed, just as others in our community are rethinking productivity in software engineering, we suggest that how dashboards are used should be reconsidered at the same time. In the following, we discuss these risks in the context of software engineering projects and software developer productivity.

- *Dashboards favor numbers over text*: While many of the artifacts that software developers work with are textual, such as requirement specifications, commit messages, or bug reports, presenting the content of these textual artifacts on a dashboard is not trivial. Techniques that aggregate textual information—for example, topic modeling or summarization algorithms—do not always produce perfect results, and it is therefore often easier to present numbers instead of text on a dashboard. As a result, a developer dashboard is more likely to contain information on how many issues were closed than information on which feature is the most mentioned in bug reports. To address this challenge, further advances in text processing research, especially applied to the heterogeneous artifact landscape of a software project, are needed.
- *Dashboards might not display relevant context*: The aggregation of information implies missing some of the details, which often means that not all contextual information is available. A dashboard that displays information about a critical bug fix might not contain all the caveats of this bug fix, and a dashboard that compares time spent in a browser to time spent in an IDE might not contain information about which of the activities were related to software development. In addition, no two software projects are alike. While the presentation of aggregated information on dashboards might invite users to compare between projects and companies, these comparisons are often flawed since they miss important context. To some extent, this can be addressed by making a dashboard interactive and allowing its users to drill down to more complete information.

- *Dashboards often don't explain*: A dashboard might be able to show that one team has fewer open issues than another team, that one component has fewer bugs than another component, or that a developer has spent more time in the IDE compared to the previous month. However, many dashboards do not provide explanations for such observations, and without explanations, this information might not be actionable. For example, a team would not know what they need to do to decrease the number of open issues they have, it might not be obvious why one component has more issues than another, and a developer might not know what they can do to improve their productivity.
- *You get what you measure*: Goodhart's law—usually cited as "When a measure becomes a target, it ceases to be a good measure"—describes another risk of the use of dashboards in software development projects. For example, if a dashboard emphasizes the number of open issues, developers will become more careful about opening new issues, e.g., by combining several smaller issues into one. Similarly, if a dashboard conceptualizes productivity as time spent in the IDE, developers might become hesitant to look up information outside of the IDE. In both examples, this was likely not the intent of the dashboard, yet decades of research on gamification have shown that humans tend to game such systems. As one of our interviewees in a previous study [8] told us: "Developers are the most capable people on Earth to game any system you create."
- *Dashboards can only be as good as the underlying data*: Many studies have found that data captured in software repositories does not always accurately reflect the development reality. For example, Aranda and Venolia [1] found that the coordination that happens around software bugs cannot solely be extracted from software repositories as it would lead to incomplete and often erroneous accounts of coordination. In a study on GitHub, Kalliamvakou et al. [5] found that almost 40 percent of all pull requests do not appear as merged, even though they actually have been merged. These are just two examples of cases where looking at repository data alone provides an inaccurate account of different aspects of software

development. If a dashboard is based on such data, it is impossible for this dashboard to display accurate information.

- *Dashboards can only display data that has been tracked somewhere*: While today's software repositories are able to capture many of the actions taken by software developers, there are still many activities that are not captured. For example, a repository would not be able to capture the watercooler conversation between developers that might have provided a crucial piece of coordination for fixing a particular bug. Negotiations with clients taking place outside of the confines of a developer office would be another example of critical information that is often not appropriately captured in a software repository. Information that does not exist in a repository cannot be displayed in a dashboard, and users of dashboards have to be aware that a dashboard might not always provide the complete picture.
- *Performance-related data on dashboards can easily be misinterpreted as productivity data*: Many of the metrics that can be easily visualized on a dashboard, such as number of open issues or number of lines of code, can be interpreted as productivity measures, enabling comparisons between developers, teams, or components that ignore the many complexities of software development. As discussed in the previous chapter, developers have many reservations about such productivity measures. As a result, they will only accept dashboards that do not attempt to reduce the complexity of a developer's contribution to a single number. Stephen Few notes that analytical dashboards need subtle performance measures—until such performance measures have been established, they should not be replaced with their nonsubtle counterparts.
- *Dashboards often do not encode the actual goals well*: There can be a tension between the goals of a software development organization and the items that are surfaced in a dashboard. While the goal of an organization might be long-term value creation, dashboards often use relatively short time spans. Values such as customer satisfaction are not readily extractable from a software repository, even though they might actually align with the organization's goal much better than the number of open issues in a project or time spent in the IDE.

Rethinking Dashboards in Software Engineering

As software engineering becomes more and more data driven and the tools for creating dashboards become easier to use, we expect to see a growth in the role that dashboards play in software engineering and an increase in the number of features they provide. For individual developers, dashboards provide insights on personal productivity, while teams and projects use them for monitoring performance, and managers and community leaders use them for decision making.

We expect that artificial intelligence, natural language processing, and software bots [6] will also impact dashboard design and the features they provide in the next few years. There is certainly opportunity to automate the display of more and more insights on data but also to improve how developers and other stakeholders collaborate with one another through dashboards. Furthermore, artificial intelligence and natural language processing could be used to gather insights on how and when dashboards are used, on the impact they may have on software projects, and on how their design could be improved over time.

We may also wonder if dashboards may even partially replace other modes of information exchange (e.g., PowerPoint slides), and indeed we have observed (informally) that this is the case at some large software companies. Once these dashboards render relevant data, will some stakeholders interpret the view they show as "truth" even though the underlying data or how it is analyzed and presented may be inaccurate, biased or misleading? Do we have sufficient understanding on the significant role they may play in software engineering projects and furthermore on the ethical concerns they may introduce when they accentuate or reveal data that may be sensitive to some stakeholders?

Dashboards and the technologies to create them are likely to become ubiquitous and easier to use over time. Whether they will enhance or possibly harm and detract from productivity or whether they may just give insights on productivity remains to be seen, but care should be taken in how they are created and used. We hope this chapter brings some insights on the diverse way they may be used as well as some awareness of some of the risks as well as opportunities they may bring to our community.

Key Ideas

These are the key ideas from this chapter:

- The landscape of dashboards that exist for visualizing software development information is extremely broad and varied.
- For individual developers, dashboards provide insights on personal productivity, while teams and projects use them for monitoring performance and managers and community leaders use them for decision-making.
- The power that dashboards provide in terms of analytics introduces risks such as the misinterpretation of productivity data and the misalignment of goals.

References

[1] Jorge Aranda and Gina Venolia. 2009. The secret life of bugs: Going past the errors and omissions in software repositories. In Proceedings of the 31st International Conference on Software Engineering (ICSE '09). IEEE Computer Society, Washington, DC, USA, 298–308.

[2] Arciniegas-Mendez, M., Zagalsky, A., Storey, M. A., & Hadwin, A. F. 2017. Using the Model of Regulation to Understand Software Development Collaboration Practices and Tool Support. In CSCW (pp. 1049–1065).

[3] Brath, R. & Peters, M. (2004) Dashboard design: Why design is important. DM Direct, October 2004. Google Scholar

[4] Few, Stephen. 2006. Information dashboard design: the effective visual communication of data. Beijing: O'Reilly.

[5] Kalliamvakou, E., G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. 2014. The promises and perils of mining GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). ACM, New York, NY, USA, 92–101.

[6] Storey, M. A., & Zagalsky, A. 2016. Disrupting developer productivity one bot at a time. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 928–931). ACM.

[7] Treude, C. and M. A. Storey 2010, "Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds," 2010 ACM/IEEE 32nd International Conference on Software Engineering, Cape Town, 2010, pp. 365–374.

[8] Treude, C., F. Figueira Filho, and U. Kulesza. 2015. Summarizing and measuring development activity. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). ACM, New York, NY, USA, 625–636.

[9] Gregory L. Hovis, "Stop Searching for InformationMonitor it with Dashboard Technology," DM Direct, February 2002.