1-2019

How team awareness influences perceptions of developer productivity

Christoph TREUDE
Singapore Management University, ctreude@smu.edu.sg

Fernando FIGUEIRA FILHO

CHAPTER 15

How Team Awareness Influences Perceptions of Developer Productivity

Christoph Treude, University of Adelaide, Australia

Fernando Figueira Filho, Federal University of Rio Grande do Norte, Brazil

Introduction

In their day-to-day work, software developers perform many different activities: they use numerous tools to develop software artifacts ranging from source code and models to documentation and test cases, they use other tools to manage and coordinate their development work, and they spend a substantial amount of time communicating and exchanging knowledge with other members on their teams and the larger software development community. Making sense of this flood of activity and information is becoming harder with every new artifact created. Yet, being aware of all relevant information in a software project is crucial to enable productivity in software development.

In formal terms, awareness is defined as "an understanding of the activities of others, which provide context for your own activity." In any collaborative work environment, being aware of the work of other team members and how it can affect one's own work is crucial. Maintaining awareness ensures that individual contributions are relevant to the group's work in general. Awareness can be used to evaluate individual actions against the group's goals and progress, and it allows groups to manage the process of collaborative working [1].

C. Sadowski and T. Zimmermann (eds.), *Rethinking Productivity in Software Engineering*,
https://doi.org/10.1007/978-1-4842-4221-6_15

Contributing to a software project requires a multitude of different kinds of awareness, ranging from high-level status information (e.g., What is the overall status of the project? What are the current bottlenecks?) to more fine-grained information (e.g., Who else is working on the same file right now and has uncommitted changes? Who is affected by the source code I am writing at the moment?). Awareness includes both short-term, momentary awareness (awareness of events at this particular point in time, such as the current build status) and long-term, historical awareness (awareness of past events, such as code evolution and team velocity). As the complexity of software systems grows, maintaining awareness of all relevant context is becoming increasingly challenging. To address this situation, many tools have been developed over the last decades to help developers maintain awareness of everything that goes on in a project.

Given the plethora of information available, tools that support awareness for software developers inevitably need to abstract some details and have to aggregate information. This leads to risks. The aggregation of developer activity information has the potentially unintended side effect of quantifying the developer's work, enabling productivity comparisons across developers and time. As an example, imagine a tool that aims to provide high-level information about what a developer is working on at the moment. Such a tool will likely be able to say that a developer is working on three features (by counting the open issues assigned to this developer, for example), but it might not be able to say that a developer is currently working on refactoring a database connector, fixing a bug in the persistence layer of the application, and improving the performance of a query (which would require an automated understanding of the semantics of the open issues). Of course, a tool could simply list all open issues, but this would lead to information overload.

In this chapter, we discuss this tension between awareness information and productivity measures, and we advocate for the design of tools that enable awareness without quantifying information. We also report on the findings from an empirical study in which we asked developers about how to design such tools. The study revealed that awareness can influence developers' perceptions of the productivity of their colleagues and that developers do not feel that productivity can be collapsed into a single metric. We conclude that while automated tools for making sense of everything that goes on in a software project are necessary to enable developer awareness, such tools need to focus on summarizing instead of measuring information.

Awareness and Productivity

We first illustrate the relationship between team awareness and developer productivity, using an existing categorization of awareness types as a guideline [2].

- *Collaboration awareness*: Collaboration awareness refers to the perception of group availability, i.e., whether people are in the same physical place, who is online/offline, and their virtual availability. In software development—and in many other domains—these concepts are directly related to productivity. If a member of a software development team is perceived to be unavailable, it is easy to conclude that they are not productive, whereas a team member who is always online and/or in the same physical place would be perceived as being productive.
- *Location awareness*: Location awareness refers to the geographical and physical nature of spaces, e.g., where someone is physically located. Similar to collaboration awareness, the physical location of team members can be related to perceptions of their productivity. This might be the case if co-workers who share the same office space are perceived as having more or less productivity compared to others, but it might also have cultural implications, e.g., if developers in an outsourcing location are perceived differently simply based on their location.
- *Context awareness*: Context awareness allows a group of co-workers to maintain a sense of what is going on in the virtual space. In software development projects, context awareness can, for example, refer to the context of a shared task, e.g., the progress of a development team toward the next release. If the development team is perceived as not being on track, this type of awareness can easily be used to reach conclusions about a team's lack of productivity.
- *Social awareness*: According to Antunes et al., social awareness is related to the understanding of "social practice, i.e., the others' roles and activities, or what and how the group members are contributing to a task." It is easy to see then how social awareness in a software development team is linked to developer productivity. If a team member's contributions to a task are perceived as not good enough, they will be considered as unproductive, and vice versa.

- *Workspace awareness*: Workspace awareness is defined as the up-to-the-moment understanding of another person's interaction with the shared workspace, i.e., awareness of people and how they interact with the workspace rather than just awareness of the workspace itself [3]. This type of awareness is also directly linked to productivity: if a developer's interactions with the shared workspace, e.g., the issue tracking system of a software project, are not as frequent or fruitful as expected, this developer will be seen as being unproductive.
- *Situation awareness*: Situation awareness refers to being aware of what is happening in the vicinity to understand how information, events, and one's own actions will impact goals and objectives. Applied to software development, this definition could refer to peripheral awareness of the work of other teams that are working on the same product, awareness of updates to libraries that a particular product relies on, or awareness of technology trends [4]. As with the other awareness types, this kind of awareness also links to productivity: if another team is not delivering the feature they are supposed to deliver or a critical bug in a library is not being fixed, developers can be seen as unproductive.

Enabling Awareness in Collaborative Software Development

There are many different kinds of information that developers need to be aware of in any software development project, as discussed in the previous section. However, with the flood of activity and information in a software repository, it is impossible and also often not necessary for a developer to maintain awareness of every aspect of a project. As a result, a mechanism for filtering and aggregating relevant information is needed.

Many tools such as feeds and dashboards (see Chapter 16) have been developed to help developers maintain awareness and aggregate relevant information. However, these tools often focus on quantitative instead of qualitative aspects since it is arguably easier to count the number of open issues than interpret what these issues are about, for example. In the next sections, we discuss developers' opinions on the aggregation of awareness information using both quantitative and qualitative means.

Aggregating Awareness Information into Numbers

Automated tools for extracting, aggregating, and summarizing development activity are essential to provide software teams with crucial awareness information. To investigate how to design such tools, in earlier work [5] we asked developers how they would design quantitative and qualitative aspects of such tools. We first summarize our findings with regard to the quantitative aspects, which revealed the risk of misinterpreting awareness information as productivity measures.

Our study participants stressed that no single metric, e.g., lines of code, number of tasks, etc., would truly reflect the wide range of activities a developer may take action on throughout the development life cycle of a software product. For instance, conceptual work is hardly measurable and may go unnoticed just by monitoring a metric, as shown in this example from one of our study participants: "It's difficult to measure output. Changing the architecture or doing a conceptual refactoring may have significant impact but very little evidence on the code base." Similarly, the difficulty of a task cannot be measured in lines of code.

Software projects may go through different stages in their development cycle. According to our study participants, these variabilities from project to project make it difficult to devise any uniform, one-size-fits-all measurement system that would work across different project contexts and distinct development workflows (challenges detailed in Chapter 2). Also, developers may assume different roles in a single day. For instance, interacting with customers and users was regarded by our study participants as an activity that is difficult to measure, although it is an integral part of development work: "We do systems for people in the first place."

Another problem perceived by our study participants is that measures can be gamed so that any automatic system aimed at measuring productivity would be potentially exploitable. This applies in particular to simple measures such as the number of issues or number of commits: "A poor-quality developer may be able to close more tickets than anyone else, but a high-quality developer often closes fewer tickets but of those few, almost none get reopened or result in regressions. For these reasons, metrics should seek to track quality as much as they track quantity."

Given the limited value of numbers as a means to provide developers with meaningful information, we next investigate the potential of qualitative mechanisms, in particular summarization, to improve the quality of awareness information.

Aggregating Awareness Information into Text

As we have discussed in the previous section, aggregating the work of software developers into numbers has many disadvantages. However, information in a software repository has to be aggregated to enable awareness without having to look at every artifact created, modified, or deleted. With this in mind, in our earlier work [5], we presented our study participants with the following scenario: "Assume it's Monday morning and you have just returned from a week-long vacation. One of your colleagues is giving you an update on their development activities last week." We then asked them what information they would expect to be included in such a summary. In the following paragraphs, we summarize the answers we received from developers.

Many of the events in the day-to-day work of software developers can be categorized according to whether they are expected or unexpected. Expected events comprise status updates that are generally not surprising to a software developer—such as a development task moving from open to closed—while unexpected events are unforeseen, for example the presence of a critical bug. Our participants requested that both kinds of events should be included in summaries of development activity.

Summaries of expected events in software development projects are mostly concerned with how different artifacts, such as development tasks or user stories, move through the development cycle. For example, one participant requested what they called "task state transition history—which tasks were taken, which were done, which were tested." An important dimension of expectations is planning—our participants were also interested to hear about short-term and long-term plans as well as the goals driving these plans.

Basic awareness tools for software developers typically support this kind of awareness of development artifacts and plans. For example, a burndown chart visualizes the actual work being done compared to a plan, and a kanban board shows tasks along with their current status. However, these tools are still limited in their expressiveness: A burndown chart cannot explain why a project is not on track, and it can also easily be misinterpreted as measuring productivity. In addition, it can be gamed, for example by overestimating user stories. Kanban boards can aggregate only to a certain extent—if the number of tasks or work items included in the kanban board becomes too large, it becomes hard to obtain a high-level overview of the project status from looking at the board.

If everything in a software project is progressing as expected, no particular action outside of a developer's routine might be required. However, things tend not to always go according to plan in software projects. Requirements might change, a major refactoring might be needed, or a critical bug might be discovered. In those situations, developers need to act, which explains why anything unexpected should play a major role in a summary of software development activity: "We cut our developer status meetings way down and started stand up meetings focusing on problems and new findings rather than dead-boring status. [The] only important point is when something is not on track, going faster than expected and why."

When we asked our participants about how to automatically detect such unexpected events, several examples were mentioned, in particular related to the commit history: "Commits that take particularly long might be interesting. If a developer hasn't committed anything in a while, his first commit after a long silence could be particularly interesting, for example because it took him a long time to fix a bug. Also, important commits might have unusual commit messages, for example including smileys, lots of exclamation marks, or something like that…basically something indicating that the developer was emotional about that particular commit." While developer tools that summarize expected events already exist—albeit often still focusing on numbers rather than textual content—research on what constitutes important unexpected events in a software project is still in its infancy.

Rethinking Productivity and Team Awareness

Throughout a software project's life cycle, developers generate a vast corpus of software artifacts and perform a multitude of actions; however, only a fraction of those events are relevant to one's own activity. Automated methods for aggregating and summarizing awareness information are important, as they potentially save developers from the cumbersome task of manually inspecting a large number of events—or asking others—to answer the various questions that may arise in one's development work.

Automated methods for aggregating awareness information are likely to produce quantitative over qualitative information since aggregating numbers (e.g., the number of issues per developer) is much easier than aggregating textual information (e.g., what kinds of issues a developer is working on). Unsurprisingly, measures such as lines of code and number of issues open/closed are available in most development

tools, but many developers in our study found them too limited to be used as awareness information and worried that such simple numbers may act as a proxy of their productivity. In short, awareness can influence developers' perceptions of the productivity of their colleagues—and these perceptions are often not accurate if based on the awareness information that tools commonly provide.

From the perspective of who receives awareness information, numeric measures should not be provided in isolation: they should be augmented with useful information about recent changes in the project that happened according to plan, i.e., expected events, and most importantly, they should provide information about the unexpected. As we noticed, awareness tool design has given greater emphasis to the former type of information, leaving information about unexpected events to be gathered by developers themselves. Similarly, awareness tools have fed developers more information about what happened and less information about why things happened.

As empirical evidence shows, the design of automated awareness mechanisms should consider the tension between team awareness and productivity measures in collaborative software development. Developers' information needs are indirectly related to productivity aspects, yet the way information is typically presented by awareness tools (e.g., kanban boards, burndown charts) can have negative effects as they facilitate judgment on the productivity of developers. We found that the ultimate goal of developers is not associated with productivity measurement: they seek to answer questions that are impacting their own work and the expected flow of events. They want to become aware of the unexpected so that they can adapt more easily and quickly.

While tools that help developers make sense of everything that goes on in a software project are necessary to enable developer awareness, these tools currently favor quantitative information over qualitative information. To accurately represent what goes on in a software project, awareness tools need to focus on summarizing instead of measuring information and be careful when presenting numbers that could be used as an unintended proxy for productivity measures. We argue for the use of natural language and text processing techniques to automatically summarize information from a software project in textual form. Based on the findings of our study, we suggest that such tools should categorize the events in a software project according to whether they are expected or unexpected and use natural language processing to provide meaningful summaries rather than numbers and graphs that are likely to be misinterpreted as productivity measures.

Key ideas

The following are the key ideas from the chapter:

- Tools that help developers make sense of everything that goes on in a software project are necessary to enable developer awareness.
- These tools currently favor quantitative information over qualitative information but need to focus on summarizing instead of measuring information.
- Team awareness can influence developers' perceptions of their colleagues' productivity, and developers do not feel that productivity can be collapsed into a single metric.

References

[1] Paul Dourish and Victoria Bellotti. 1992. Awareness and coordination in shared workspaces. In Proceedings of the 1992 ACM conference on Computer-supported cooperative work (CSCW '92). ACM, New York, NY, USA, 107-114. DOI=https://doi.org/10.1145/143457.143468.

[2] Pedro Antunes, Valeria Herskovic, Sergio F. Ochoa, José A. Pino, Reviewing the quality of awareness support in collaborative applications, Journal of Systems and Software, Volume 89, 2014, Pages 146-169, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2013.11.1078.

[3] Gutwin, C. & Greenberg, S. Computer Supported Cooperative Work (CSCW) (2002) 11: 411. https://doi.org/10.1023/A:1021271517844.

[4] Leif Singer, Fernando Figueira Filho, and Margaret-Anne Storey. 2014. Software engineering at the speed of light: how developers stay current using twitter. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). ACM, New York, NY, USA, 211-221. DOI: https://doi.org/10.1145/2568225.2568305.

[5] Christoph Treude, Fernando Figueira Filho, and Uirá 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. DOI: https://doi.org/10.1145/2786805.2786827.