Using Analytics to Support Decision Processes for Development and Design

Patrick Wagstrom
IBM TJ Watson Research Center
19 Skyline Dr, Hawthorne, NY 10532
pwagstro@us.ibm.com

Anita Sarma
Department of Computer Science & Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588
asarma@cse.unl.edu

ABSTRACT
Globalization has resulted in software teams that are increasingly large, multi-cultural, and geographically distributed; and software systems that are increasingly dependent on technology produced by remote teams or as commercial-off-the-shelf (or open sourced). In such environments making the right technical decision largely hinges on the social context and trust toward the software producer. Creating this shared context and building trust is not easy – especially when teams are distributed across international boundaries, time zones, and culture. Creation of trust has thus far been largely an informal process with little technological support. However, the popularity of online repositories, both public (e.g. StackOverflow, GitHub etc) and private (e.g. internal source code repositories, project plans), that store social and technical information has made it possible to analyze past development and social interaction traces to develop actionable analytics to facilitate building trust.

Author Keywords
Software Development, Design, Analytics

ACM Classification Keywords
D2.6. Programming Environments, D2.9: Management: Programming Teams, H5.2: User-Interfaces: Training, Help and Documentation

General Terms
Human Factors, Management, Design

INTRODUCTION
Despite advances in software engineering and software processes over the last half-century, software projects still fail – often. The prevalence of large-scale projects and distributed teams contributes to this problem by creating an environment where teams are comprised of members who are dispersed in both time and space. Teams are dynamic and fluid, often created by bringing members together for specific tasks, for a specific time period, and comprising members who have never met each other. Such teams suffer from numerous performance degrading issues; chief among them is the absence of trust among team members and lack of shared mental models of work and culture.

Numerous case studies have found distributed software development to inherently suffer from longer times to completion [2,6], higher incidence of bugs [8], and miscommunications. A large portion of these problems can be attributed to a lack of trust and a shared mental model of the work to be completed and the culture across teams. For example, a study found developers to wrongly perceive that remote team members provided less help [3] compared to their collocated members, where in fact, the remote team members had equal contributions. Other studies have found that time zone differences along with different national holidays can lead team members to perceive their remote team members to be late or worse, incompetent [5,7].

Developers without a shared work history, a common situation in large distributed teams, have not had a chance to create a notion of trust (or mistrust). These developers who effectively start with a blank slate may succumb to wrong perceptions in the absence of these past experiences. Further, the lack of prior interactions also creates a lack a shared understanding of what the work composition and required processes, which leads to additional conflicts [4]. In fact, different team members often have a different understanding of the product and its responsibilities making it difficult to explain design rationale, agree on the right technology, or find experts. These factors in turn decrease team efficiency and can lead to increases in software defects, failure to satisfy project requirements, or, in extreme cases, complete project failure.

The above problems are not new and tools exist with goals to prevent technological conflicts [1] and aid in the learning of new technology (APIs) [9]. While these tools have greatly increased the ability of teams to handle technological difficulties, they do not help much with the social difficulties. Tool support to facilitate social issues such as engendering trust or creating shared mental models is nascent at best. We believe that future software development tooling will focus on the inclusion of social interactions and the context of software use along with software artifacts, which can then help in building trust about a technological piece or a team member. Another future direction is the archival of the context in which a piece of software is being used and the information that led to the technology decision.

Future CSD'12, February 12, 2012, Seattle, Washington, USA.

Such archival of usage context and design rationale can then inform the creation of shared mental models even when team members have not worked together.

SOCIAL ARTIFACTS

Software engineering projects have traditionally focused on collecting data from technical artifacts that are created as a byproduct of the process of developing software. For example, mining a software version control repository allows teams (and researchers) to reconstruct the process of software development and identify problem spots in the development process. Other popular artifacts are project defect databases, mailing lists, and real-time chat. Although the last two are social mediums, the extent to which most projects make use of these archives to analyze social interactions is negligible.

Whereas previous environments considered each of the different collections of artifacts as its own unique entity, advancements in the ubiquity and robustness of project hosting and development focused websites often provide integrated methods that link together project artifacts. These sites frequently have APIs for analyzing these social interactions and context of a project. For example, GitHub allows developers to not only search existing code repositories for similarly functional code, but also log social interactions such as interest in a particular project repository, ability to get updates about a project, or follow activities of another developer. Technical question & answer websites (e.g., StackOverflow) facilitate social discussions around technical topics and have become a critical resource when seeking technical information. Further, these sites also record an aggregation of users' interaction and the "quality" of their contributions via reputation scores (GitHub score or StackOverflow reputation points). The use of reputation in these as an external signal of expertise is becoming common as shown by the success of tools that generate resume like documents from StackOverflow and GitHub activity histories. Anecdotal evidence shows that large enterprises are adopting similar technology to provide a database of questions, answers, and expertise for their own internal communities.

These sites have laid the foundation for recording user activities to provide information on user skills and expertise. However, they still exist as islands of information in an increasingly connected world. They provide few connections to other resources and little insight into how to act knowing that a user has a particular set of skills. We believe that the next generation development environments will use public APIs of websites that log social interactions and combine them with other information, such as trends of usage of a particular technology to create analytics that provide concrete actionable information in making decisions regarding individual software development or project management.

INTEGRATION OF ANALYTICS

We propose that future development tools should treat archived social and technical data as first-class entities rather than by-products of the software development process. This is a design shift from the current methods that is necessary to create robust and actionable analytics and derive full value from the collected data. This information, when designed to be easily accessible, can be easily combined with internal and external data sources to create a shared context between team members, engender trust, and provide support for decision making within the team. Here we discuss two such scenarios of analytics use.

Design analytics through the IDE: Most software engineers spend a significant portion of their time within the integrated development environment (IDE). These environments (e.g., Eclipse and Microsoft Visual Studio) are modular and allow extensions, which have been used to create plugins that help in search, debugging, change awareness, and so on. We believe that analysis of past usage of a technology, ongoing trends, reputation of the team, and its process can be tightly integrated with the IDE to help in decision-making by calculating analytics based on the context of use and providing the information in a timely manner.

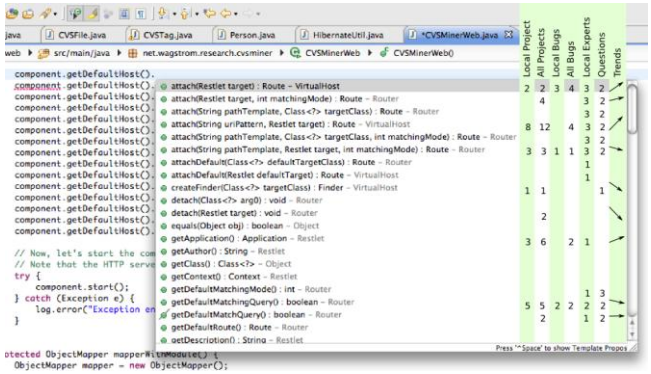

Figure 1. Enhanced Context Aware Method Browsing in an Integrated Development Environment

Figure 1 shows a mockup of an IDE augmented with private and publicly available information and analytics. Let us assume that a developer is exploring which method to use on an object. A standard development tool typically provides an alphabetical list of the applicable methods, their signatures, and documentation if it is available inline with code. In our example, our analytics enhanced IDE has analyzed the existing code base to identify how often each of the methods is used in the project and across all projects in the organization, the quality of the documentation, the track record of the method in terms of defects, and also has queried external question and answer services to provide an immediate link to technical questions about the method. Users can also vote (simple thumbs up or down mechanism) on the information that they found pertinent when making their decision, which will help in archiving the design rationale for future use.

The integration of analytics would also include social information such as the reputation of the technology provider, his/her frequency of contributions, status in the project, etc., The analytics can be contextualized by providing information only about current team members, so that it can help in identifying expertise and engendering trust across current team members (who may be distributed).

Design Analytics for Project Management: Beyond the level of a developer we consider a project architect who is revisiting the design of her project. Software defects can arise because of the lack of a shared mental model of the system, which can occur when the design rationale for a technical decision is not readily apparent. This can cause both interpersonal issues in a team and sometimes severe technical issues – an example of which happened in 2009 when many Linux distributions rushed to patch a critical component that utilized unitialized and out of process memory as an entropy pool for encryption. The rationale behind this decision was not explicitly recorded and a later effort to clean up typical poor coding practices had removed this code and thus all entropy necessary for encryption and all security from the system. Design decisions such as above can be facilitated by using publicly available data surrounding the usage contexts and problems of different libraries, modules, and methods. Further, the tool can help archive information that led to the final decision (e.g., prevalence of questions on public QA sites, the trends of each API in search engines, poster karma, trust in a particular company) through simple voting mechanisms. These design rationales can then be easily made available for future reference and analysis. Unfortunately, at this stage, most projects lack even a basic tool that would be suitable for augmentation in this way.

FUTURE CHALLENGES

The vision that we present here is conservative, but we believe, a realistic vision of changes to come in the next five years of software engineering. While some our suggestions are incremental changes to the process, they lay the foundation for much needed long-term work. A growing concern is the ballooning size of software development teams and the impact this has on the ability of team members to understand the project and to build trust in other team members. Even simple issues, such as understanding that different parts of the world have different work weeks and holidays can easily lead to a breakdown of trust within a team when responses are too slow. When one factors in the myriad of other concerns bearing down participants in a software ecosystem, it quickly becomes clear there is a lot of work needed to build trust and understanding past decisions as discussed in this work.

ABOUT THE AUTHORS

Patrick Wagstrom is a research staff member at the IBM TJ Watson Research Center in Hawthorne, NY. As a member of the Governance Science team he conducts research into improving team performance across the entire software development ecosystem. His recent research focuses on the ways that extended stakeholder (e.g. individuals involved in the software process who do not write code) impact the ability of enterprises to deliver complex software packages.

Anita Sarma is a professor in the Computer Science and Engineering department at the University of Nebraska-Lincoln. She heads the Interaction Design and Coordination lab at UNL. Her research involves understanding and facilitating coordination in distributed work. Her work includes empirical analysis of existing software teams (and project archives), creating novel coordination tools, and end user experiments to evaluate the feasibility of coordination tools.

REFERENCES

1. Brun, Y., Holmes, R., Ernst, M.D., and Notkin, D. Proactive Detection of Collaboration Conflicts. *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ACM (2011), 168–178.
2. Herbsleb, J. and Mockus, A. An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Transactions on Software Engineering 29*, 6 (2003), 1-14.
3. Herbsleb, J.D., Mockus, A., Finholt, T.A., and Grinter, R.E. Distance, Dependencies, and Delay in a Global Collaboration. *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, ACM (2000), 319-328.
4. Hinds, P. and Bailey, D. Out of Sight, Out of Sync: Understanding Conflict in Distributed Teams. *Organization Science 14*, 6 (2003), 612-632.
5. Jarvenpaa, S.L., Knoll, K., and Leidner, D.E. Is Anybody Out There?: Antecedents of Trust in Global Virtual Teams. *Journal of Management Information Systems 14*, 4 (1998), 29-64.
6. Nguyen, T., Wolf, T., and Damian, D. Global Software Development and Delay: Does Distance Still Matter? *Proceedings of the 2008 International Conference on Global Software Engineering*, IEEE (2008), 45-54.
7. Olson, G.M. and Olson, J.S. Distance Matters. *Human Computer Interaction 15*, 2&3 (2000), 139-178.
8. Ramasubbu, N. and Balan, R.K. Globally Distributed Software Development Project Performance: An Empirical Analysis. *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ACM (2007), 125–134.
9. Stylos, J. and Myers, B.A. Mica: A Web-Search Tool for Finding API Components and Examples. *IEEE Symposium on Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006*, IEEE (2006), 195-202.