The ramp-up challenge in open-source software projects

Davor Čubranić

Department of Computer Science
University of British Columbia
201-2366 Main Mall Vancouver, BC Canada V6T 2B9
E-mail: cubranic@cs.ubc.ca

1 Introduction

In a world where, thanks to technology, work is at the same time becoming globalized and more fragmented, it is no wonder that software engineering in the age of the Internet should face the same two challenges. This is especially true in the case of open-source software engineering: an often extreme example of globalization, open-source development teams allow virtually anyone with Internet access to participate in a project. A team of dozen or so members can easily be spread across five or six countries and a couple of continents [6].

On the other hand, the process of globalization results also in the fragmentation of the development team. Communication in geographically dispersed teams gets increasingly difficult as face-to-face contact becomes rare or impossible: it is not uncommon for members of open-source software projects never to have met their collaborators in person; similarly, two developers working in North America and Europe might have only an hour of overlap in their working hours. As a result, developers have to resort to technology to bridge both distance and time. Even worse, the dynamics of OSS development can be very fluid, a sort of organized chaos: team membership can change overnight, time commitments vary depending on members' other obligations (most of the time they hold a regular job or work on a degree and volunteer on the OSS project), as do their interests—given the mostly volunteer basis, it is almost impossible to make someone do work in which he or she is not interested.

OSS projects have with time evolved organization that is particularly suited to operate under such conditions. It is characterized by pronounced modularity of the system, where each module is "owned" by at most a few developers. Owners typically have absolute control over their modules and the final word on what contributions get accepted (as a well-known example, Linus Torvalds has veto power over what goes into the Linux kernel—anything else becomes a loadable kernel module). This small group of module owners and other most respected and active developers forms the core of the development team, and typically its members are the only ones with write permissions to the code repository, thus controlling what is "the" system. The core group changes slowly—unlike the larger pool of contributors who send in bug reports, enhancement requests, and code patches—and functions as a sort of meritocracy: frequent and valuable contributors increase in stature until they become members of the core group; once inside, they're expected to remain active or they may risk losing their status and corresponding privileges.

2 The ramp-up problem

While this type of team organization has proven itself quite adept at maintaining the control over the development in the face of ever-changing and often unpredictable circumstances, under such conditions it is arguably much more difficult for an engineer to join a project (and gain understanding of the software system) at an advanced stage. In traditional software development, new members of the team are commonly mentored by more experienced colleagues [10]. This support network is largely missing in the case of virtual teams because the communication is more costly and there are no lightweight channels available for quick information exchange. To make the matters even worse, in open source software development it is less likely that the experienced programmers will have the time or resources for the type of one-on-one relationship required when mentoring new project members [2, 10]. Furthermore, there is far less incentive to do so, because they hardly be obligated to do so if they are volunteering on the project, nor do they feel much of a bond with somebody they have never met or heard of.

Without the "oral tradition" on which to fall back, the two sources that remain available to an engineer seeking to understand the software are the code and the documentation. Documentation, in practice, is notorious for often being out of date. Attempting to understand a system by reading its source code is tedious, error-prone, and time-consuming.

More importantly, it becomes impractical as the code base gets large.

Tools that analyze the code to present an engineer with a view of the system which can aid in the process of program understanding have been a topic of research for a number of years now, of course, although they have still not become widely accepted in practice. What I believe deserves further consideration, however, is how to complement them by using another source of information that has been largely overlooked so far: archived communication among the developers.

3 Developers' communication as a documentation artifact

In the course of their work, developers exchange large amounts of information with their peers, and channeling this information to those who may benefit from it although they were not the original participants in the conversation is an attractive proposition.

For example, Miura, Kaiya, and Saeki [9] present a method for building the structure of specification documents that used utterances in requirements elicitation meetings. In their method, analysts develop the specification documents whose structure reflects the structure of the meeting activities. The claimed benefits are specification documents that are more complete, include design rationale, and use inquiry-answer cycles from the meetings to make the specification easier to understand. Experiments they conducted showed that readers using specification documents created using this method had to jump around the document a lot less to find the required information than did the readers who used the originals, although they also cautioned that is possible that the said structure might be more suitable for newcomers to the system.

Raison d'Etre [4] system also uses video, but to provide and organize the informal history and rationale that teams create and share in the course of their work. It is a design history application that provides access to a database of video clips containing stories and personal perspectives of team members. The database is created by recording interviews with the team members at various times through the course of the project and then manually dividing and categorizing them into clips dealing with a single topic. Furthermore, it also includes transcripts of the video clips, searchable by keyword and content.

With increasing amounts of communication going through computer-mediated channels such as email or discussion groups, developers' communication becomes yet another electronic artifact that can be analyzed in a variety of ways (e.g., Dutoit and Bruegge's communication metrics [5] that can be used to gain insight into the development process and identify potential problems). Moreover, such computer-mediated communication can easily be stored on-line and turned into a form of organizational memory, accessible by developers wanting to know whether a problem on which they are working—or a related one—has already been discussed or even solved.

The SAGA project [3], for example, included a system called Notesfile within its software development environment, which offered functionality similar to Usenet news: a distributed, networked collection of notes. Notes were categorized by topic, and each note in a topic had a title (subject) and could have an associated sequence of replies. Notesfile supported "technical discussions, product reviews, hardware and software bugs and fixes, agendas and minutes, grievances, design and specification documents, lists of work to be done, appointments, news and mail."

Similarly, Lindstaedt's GIMME system [7] captured all mail sent to a specific group alias and adds it to the information space accessible via the Internet. GIMME also helped the group organize, share, and retrieve stored conversations by automatically categorizing them according to the subject line and offering retrieval mechanisms that ranged from a simple reverse chronological listing through free-form text queries using the Latent Semantic Indexing algorithm to related message delivery. In addition, users could manually create, rearrange, or delete categories and reorganize the mail among the categories, which allowed the memory to evolve as the application domain's conventions and vocabulary and group's understanding of it changed over time.

While this may not sound like much more than a mailing list archiver on steroids, GIMME's most interesting appplication—described by Lindstaedt and Schneider in [8]—is integration with FOCUS, a tool for explanation and discussion of software prototypes. FOCUS captures prototype demonstrations and explanatory code walkthroughs; the resulting information hyperstructure of execution paths through the code, screen recordings and demonstrator's spoken and typed explanations can be used by other developers to evaluate and reuse the prototype or its portions. The audience can also engage in a sort of electronic dialogue with the demonstrator, and the questions they ask can lead to better explanations focussed on real issues. GIMME adds to FOCUS—originally geared to short- and medium-term collaboration—the storage, organization, and retrieval capabilities necessary to secure the information over longer periods of time and grow it into a group memory.

4 Open-source software applications

I believe these kind of approaches to enriching the documentation and aiding in program understanding can be even more useful for open source software, where virtually all of the communication is conducted through public forums such as newsgroups, mailing lists, and issue tracking systems like

Bugzilla. While these forums are already archived most of the time, finding useful and relevant information in the hundreds of threads and messages is extremely difficult. In my research, I am looking into methods for timely and useful presentation of such information for the purpose of aiding program understanding.

References

[1] M. E. Atwood, B. Burns, D. Gairing, A. Girgensohn, A. Lee, T. Turner, S. Alteras-Webb, and B. Zimmermann. Facilitating communication in software development. In *Proceedings of DIS'95 Symposium on Designing Interactive Systems: Processes, Practices, Methods, & Techniques*, pages 65–73, Ann Arbor, MI, USA, 1995.

[2] L. M. Berlin. Beyond program understanding: A look at programming expertise in industry. In *Empirical Studies of Programmers: Fifth Workshop*, Papers, pages 6–25, 1993.

[3] R. H. Campbell and P. A. Kirslis. The SAGA project: A system for software development. In P. Henderson, editor, *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 73–80, Pittsburgh, PA, 23–25 Apr. 1984.

[4] J. M. Carroll, S. R. Alpert, J. Karat, M. V. Deusen, and M. B. Rosson. Raison d'etre: Capturing design history and rationale in multimedia narratives. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 1, pages 192–197, 1994. Color plates on page 478.

[5] A. H. Dutoit and B. Bruegge. Communication metrics for software development. *IEEE Transactions on Software Engineering*, 24(8):615–628, Aug. 1998.

[6] R. T. Fielding. Shared leadership in the Apache project. *Communications of the ACM*, 42(4):42–43, Apr. 1999.

[7] S. N. Lindstaedt. Towards organizational learning: Growing group memories in the workplace. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, volume 2 of *Doctoral Consortium*, pages 53–54, 1996.

[8] S. N. Lindstaedt and K. Schneider. Bridging the gap between face-to-face communication and long-term collaboration. In *GROUP'97: International Conference on Supporting Group Work*, pages 331–340, 1997.

[9] N. Miura, H. Kaiya, and M. Saeki. Building the structure of specification documents from utterances of requirements elicitation meetings. In *Proceedings of the 1995 Asia Pacific Software Engineering Conference (APSEC '95)*, pages 64–73, Brisbane, Australia, 6–9 Dec. 1995. IEEE Computer Society Press.

[10] S. E. Sim and R. C. Holt. The ramp-up problem in software projects: A case study of how software immigrants naturalize. In *Proceedings of the 20th International Conference on Software Engineering*, pages 361–370, Kyoto, Japan, 19–25 Apr. 1998. IEEE Computer Society Press / ACM Press.