

Open Source, Design, and Collaboration

James D. Herbsleb
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA, USA 15213
+1 412 268-8933
jdh@cs.cmu.edu

Audris Mockus
Avaya Labs Research
233 Mount Airy Road
Basking Ridge, NJ, USA 07920
+1 908 696-5608
audris@avaya.com

1 DESIGN IS COLLABORATIVE

There is abundant evidence that software development in general is an inherently collaborative activity, with much time invested in communication [e.g., Perry et al, 1994]. Design activities seem to be particularly collaborative in commercial development. In fact, evidence suggests that more than half the design effort is spent in group work, a proportion much higher than other development activities (see Figure 1).

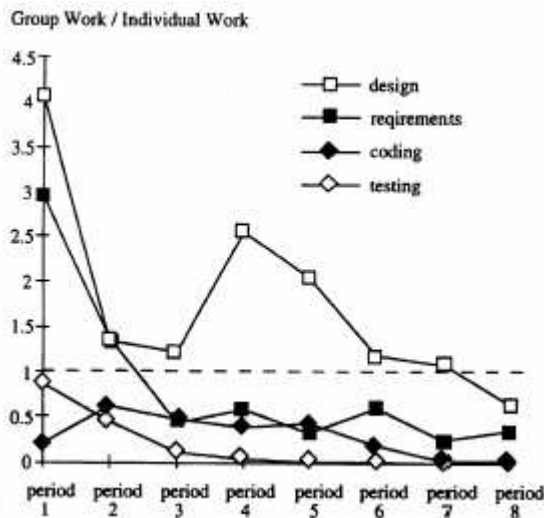


Figure 1. Ratio of group work to individual work in a commercial software development project in the telecommunications industry. Each period is four weeks. From Herbsleb, et al (1995).

2 COLLABORATION SUFFERS OVER DISTANCE

Collaborative work is difficult across distance for a variety of reasons [Olson and Olson, 2000]. With respect to software development, there is evidence that work spread across sites takes much longer than comparable work where all the participants are co-located [Herbsleb & Mockus, 2003a]. Moreover, there are a number of specific differences in co-located and cross-site social networks in a software

development context. For example, as compared to same-site networks, cross-site networks have much less frequent communication, are much less effective in disseminating important information, and are less effective for locating and engaging experts. Furthermore, project members separated by distance experience much less of a feeling of “teamness” than co-located members, and report that their distant colleagues are much less likely to help out with heavy workloads.

3 COLLABORATION AND COORDINATION ARE CENTRAL CONCERNS OF SOFTWARE DESIGN

Not only is a substantial proportion of developers’ time spent in communication, problems in communication and coordination are one of the most pervasive problems in commercial software development [Curtis et al, 1988].

In the area of software design, in particular, many of the foundational ideas primarily address coordination issues. For example, the concept of modular design is clearly among the most important ideas in the field. The whole point of modularity is to create “work items” (the classic Parnas [1972 paper defines modules as work items) that can be assigned to teams to allow them to make their design decisions about their modules independently of the design decisions made by other teams about other modules. Good design cannot be separated from the problems of coordination design decisions.

4 OPEN SOURCE IS COLLABORATION OVER DISTANCE

Open source developments have produced a number of high quality products that compare favorably in terms of defects, productivity, and customer support with commercial development. Given this existence proof, it must be the case that open source tools and practices somehow either overcome the problems of distance, or avoid them by undertaking only the

work where such problems are minimized, or both. We suspect that both of these approaches are implicitly used by successful open source developments.

Overcoming the limitations of distance is accomplished in a variety of ways. The tools and practices make both the code and the discussions and interactions concerning all the important decisions open for inspection. This provides a means for new members to get “up to speed” on current design activities, and creates a design rationale record that is used when the code is modified in the future. Everyone who may have some knowledge or experience to bring to bear can identify occasions when it is needed, and everyone is aware of decisions that have been made. Typical OSS practices require that these public sources be consulted prior to contributing to the public record, helping to ensure that this resource is actually used.

We speculate that other features of open source design contribute to effective coordination. Since tasks are self-assigned, contributors are free to look around at any number of projects, and any number of places within a project, to determine where to make a potential contribution. One would expect this self-selection to result in a good match between skills and task. Effective assignment of people to various facets of design work is a difficult coordination problem that can be readily resolved by this mechanism.

Finally, there is evidence that the division of labor that spontaneously arises in an open source context is particularly effective at resolving coordination problems. Development of new features, which inevitable involves complex design work requiring a high degree of coordination among individuals, is done by a very small number of “core” members [Mockus et al, 2002], while bug fixing, which mainly involves finding the problem and creating a small, circumscribed fix, is done by a much larger group, and testing, which involves essentially no interdependencies requiring coordination, is performed by very large numbers of people.

5 LIMITS OF COMMUNICATION AND COORDINATION IN OSS

In addition to the practices that promote effective communication and coordination in open source developments, we speculate that open source developments tend strongly to avoid design tasks

that are particularly demanding of communication and coordination. We attempt to capture this speculation in Figure 2.

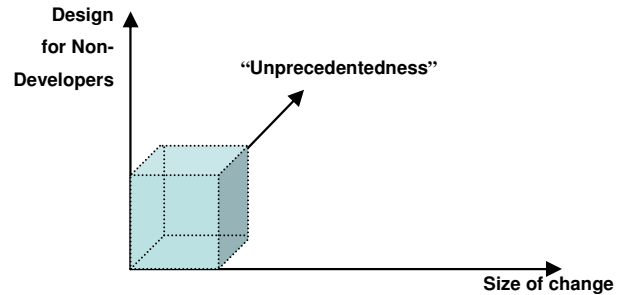


Figure 2. Conceptual graph representing the space (blue cube) where open source design typically operates.

Figure 2 represents the following three speculations:

- It is difficult for open source developments to make large, integrated changes which require distributed groups to work in close coordination (the “size of change” axis).
- Open source developments work best when the developers are typical users of the software, and will work much less well to meet the needs of non-developer users because of the very limited ability to include those considerations in the design process (“design for non-developers”).
- Open source most often evolves an existing application, or uses an existing, similar, application as a model of what to build. The complex of sweeping design decisions that need to be made for unprecedented systems are very difficult to coordinate (“unprecedentedness”).

6 OSS RESEARCH OPPORTUNITIES

Effective collaboration over distance. The tools and practices that have co-evolved in open source design contain many potential lessons. Openness of tools, self-selection of tasks, division of labor built around the dependencies of different types of work, modularity of design, and organization as families of projects are among the features that merit study.

Exploring, overcoming OSS limitations. We have speculated about several limitations to OSS designing, but empirical research is required to understand if these limitations exist, and how severe they are. If they are real and substantial, can they be overcome in order to greatly increase our capacity to create a public good, software, in this way?

Developing, testing, refining theories of coordination in software design. We are badly in

need not only of studies that empirically explore the techniques and limitations of coordination in design, but also of theories that can explain the observed phenomena.

There are several promising lines of work. One new approach is to address coordination in design from a software engineering perspective [Herbsleb & Mockus, 2003b]. For example, it has long been speculated, and there is now evidence that changes that cross module boundaries are more difficult to make than changes that remain within a module. The theory makes a number of other predictions that will be tested in future research.

Social network theory provides a rigorous way to characterize the complex relationships among people engaged in coordinated activity. The characteristics of the social network can have powerful influences on the effectiveness of the engineering activities. For example, one can construct a network in which the nodes are people, and the arcs represent workflow, i.e., the assignment of work by one individual to another (see Figure 3).

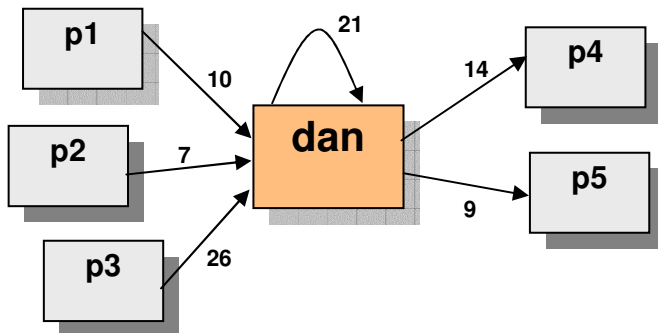


Figure 3. Fragment of a social network taken from a commercial project. The nodes are developers, and the weighted arcs represent the number of work items assigned by one person to another.

Statistical modeling of social network variables and control variables indicates, in line with our theory of coordination, that the indegree (i.e., the number of arcs pointing to a developer) predicts the productivity of that developer [Herbsleb & Mockus, 2003b]. Such network models not only have the potential for helping us to understand the key characteristics of coordination, but in addition, given the availability of appropriate data, the performance consequences of various patterns of coordination can be explored.

Organizational simulation is another promising approach for understanding coordination in design. For example, we are currently exploring the security

implications of open source development by modeling the discovery, communication, patch creation, and patch installation, as well as discovery, exploitation, and exploit release process to understand when open source is likely to enhance or reduce the security of designs.

7 RESEARCH PRIORITIES

7.1 Create a data collection and analysis infrastructure.

Most open source developments capture a vast amount of project data that could be used, in principle, for empirical research to address the research issues we outlined in sections 4-6. There are, however, formidable pragmatic and methodological obstacles.

Like most commercial software projects, open source projects rely on various tools to enable multiple people to work on a project. The usual toolset includes version control systems to coordinate changes to code base and documentation; work flow (change control) systems to track work items and problems; and knowledge management systems to support requirements, design and dissemination of project information. Many projects, such as Mozilla, also use IRC chat for real-time communication. In any case, project repositories can provide virtually every detail about what work was performed and often why it was done. It represents all information that was needed to complete the project and, therefore, is an invaluable resource for empirical research.

A growing number of researchers are using such information, each using slightly different tools, assumptions, and models. There is a need to create infrastructure to validate and model software project data in open source projects in order to facilitate replicable studies on this observable and widely available software project data. It would enable faster progress by removing the need to recreate data extraction and processing infrastructure in each study.

The main tasks that would support conducting replicable analysis of open source project data include retrieval, summarization, and validation of data from mailing lists, CVS logs, ChangeLog files, and defect tracking systems based on Bugzilla: systems and source that are commonly used in open source projects. Because open source projects usually involve individuals who do not meet face to

face, almost all of the activity in the project leaves detectable traces in change logs, mailing lists, chat, or problem reporting systems. Such rich data provides excellent basis to study and compare open source software projects, especially to study various aspects of continuous design.

The project data has a number of distinct benefits that may not be immediately obvious:

- The data collection is non-intrusive, using only existing data and making analysis possible in projects that cannot or would not collect additional data. Long history on past projects is typically available, enabling comparison to what happened in the past and customization and calibration of the methods to the existing environment.
- The information is fine grained, at the problem report/code change/discussion list posting level.
- The information is complete, all project information needed to complete the project such as code changes/defects/message exchanges are recorded.
- The way the project's system is used rarely changes, making data uniform over time.
- Even small projects generate large volumes of changes making it possible to detect even small effects statistically.
- The systems are used as a standard part of the project, so the project is unaffected by observers' intrusion.

Analysis of the project data has a number of serious challenges that require additional research and tools to address. The basic problem is that the systems are used to support development work, and were not designed to support empirical research. Moreover, the usage of tools may vary from project to project. Consequently, data needs to be extracted from various systems, requiring drivers for systems used in various projects to extract information that can be interpreted in a similar way across the projects. The data has to be integrated for different systems used within one project, e.g., individual's ID may be stored as login in CVS and as email in Bugzilla. Individuals often use a number of logins, e.g., because they forget a password and create a new identity. The work flow system is typically not linked to the version control system, making it extremely difficult to know precisely which changes

in the source code correspond to a requested bug fix or change in functionality.

The last, and hardest, part is to produce quantities that are of interest in empirical research, such as basic product and process characteristics like quality, effort, and interval as well as product, work, organization dependency graphs. Typically, this requires building a model of development work and obtaining the desired quantities using predictors from the project data [see, e.g., Atkins et al., 2002].

Social network data, so key to understanding the work patterns, subgroups, knowledge clusters, and communication within the development community, are often extremely difficult to construct accurately. The problems of multiple "identities" (i.e., more than one e-mail address or login per person) is formidable, and can introduce major distortions in one's view of the network. Creating many types of social network graphs requires linking all sources of information related to a particular change, i.e., discussion of the change on the mailing list, the actual change request and associated discussion and disposition, and the changes in the code which implement the change. Right now it is extremely difficult to create these links accurately.

A very important part of empirical research requiring correspondingly largest amount of effort is to validate the extracted data to make sure that it reflects the process used in the project. This, of course, implies the need to develop tools and methods to support such tasks.

Supporting/automating the following capabilities would greatly facilitate a wide variety of empirical analysis:

- retrieve the raw data from the web or the underlying system via archive downloads, CVS logs, and processing Bugzilla web pages;
- verify completeness and validity of different change records by cross-matching changes from CVS mail, CVS log, and ChangeLog files; matching changes to PR reports and identities of contributors;
- construct meaningful models and measures that can be used to assess various aspects of open source projects.

7.2 Applications that exploit the infrastructure to enhance OSS capabilities.

Assuming that open source development is discovered to have substantial limitations, as hypothesized in section 5, the infrastructure described in section 7.1 could be exploited by collaborative and visualization applications to enhance development capabilities. Such applications would have to be introduced gradually to allow socio-technical co-evolution.

There are several specific types of applications that would be very useful to open source developers. One is to provide visualization and work flow tools based on social network models. Since distributed developers have much sparser interactions than co-located development teams, their transactive memory is substantially reduced. Communities surrounding projects could perhaps compensate for sparse interaction, and learn much about their own activities and structure through visualizations of communication networks, knowledge networks, analysis of roles and network properties. In fact, the lead developer of the Cocoon project, one of the Apache projects, has created one type of social network visualization, where people are nodes and edges represent replies within threads in mailing list postings (see figure 4).

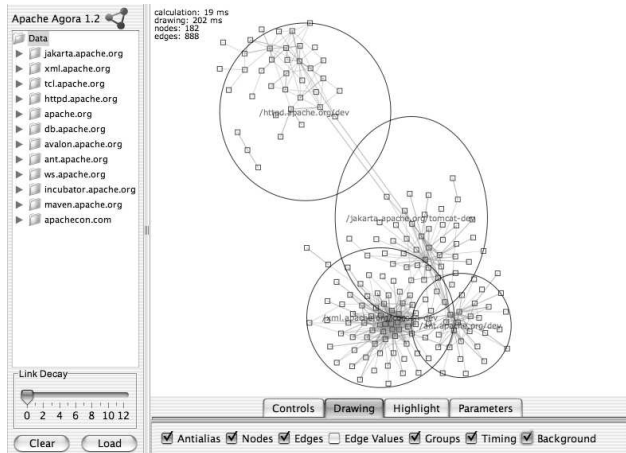


Figure 4. The Apache “Agora” applet created by the lead Cocoon developer (Mazzocchi, 2003).

Having an appropriate infrastructure and modeling capability would allow the constructing of social network models to support many kinds of tasks. For example, based on an analysis of relationships to knowledge clusters, of files previously modified, and other available data, we are creating a system to

match bug reports with people, both to relieve the administrative overhead of routing bugs, and to allow developers to browse for appropriate ways to contribute to projects they have not yet worked on.

Other possible applications involve finding experts through applications that access CVS and change logs, as well as communication networks in order to find potential collaborators, for example. Suppose for example, a distributed group of scientists wish to collaborate on an extension to an analysis package that involves closely coupled work that outstrips the resources available at any one location. Tools that allow browsing of knowledge networks and the quick establishment of ad hoc “virtual teams” with shared workspaces and collaboration tools could greatly enhance the capabilities of open source developments.

It is important to seek to overcome limitations in open source development. Software is a critical component of nearly all technologies, and ability to effectively create high quality is critical to generating scientific and engineering capabilities. Especially where markets do not provide appropriate software, and in settings where resources are scarce, open source development may provide the only means to generate this critical resource.

8 REFERENCES

- D. Atkins, T. Ball, T. Graves, and A. Mockus. Using version control data to evaluate the impact of software tools: A case study of the version editor. *IEEE Transactions on Software Engineering*, 28(7):625-637, July 2002.
- B. Curtis, H. Krasner, and N. Iscoe, “A Field Study of the Software Design Process for Large Systems,” *Comm. ACM*, vol. 31, no. 11, pp. 1268-1287, 1988.
- Herbsleb, J. D., Klein, H., Olson, G. M., Brunner, H., Olson, J. S., and Harding, J. (1995). Object-oriented analysis and design in software project teams. *Human Computer Interaction*, 10, 249-292.
- Herbsleb, J.D. & Mockus, A. (2003a) An Empirical Study of Speed and Communication in Globally- Distributed Software Development. *IEEE Transactions on Software Engineering*, 29(3), 2003, pp. 1-14.
- Herbsleb, J.D. & Mockus, A. (2003b). Formulation and Preliminary Test of an Empirical Theory of Coordination in Software Engineering. In proceedings, *ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 112-121, Helsinki, Finland, September 1-5, 2003.

Mockus, A., Fielding, R., & Herbsleb, J.D. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3), 2002, pp. 309-346.

Mazzocchi, S. Apache Agora. Available at <http://nagoya.apache.org/~stefano/guide.html>. Screenshot downloaded 10/6/2003.

Olson, G. M. and J. S. Olson. Distance Matters. *Human-Computer Interaction*, 15, 139-179, 2000.

D.L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," *Comm. the ACM*, vol. 15, no. 12, pp. 1053-1058, 1972.

D.E. Perry, N.A. Staudenmayer, and L.G. Votta, "People, Organizations, and Process Improvement," *IEEE Software*, vol. 11, no. 4, pp. 36-45, 1994.