Global Software Development in the FreeBSD Project

Diomidis Spinellis
Department of Management Science and Technology
Athens University of Economics and Business
Patision 76, GR-104 34 Athens, Greece
dds@aueb.gr

ABSTRACT

FreeBSD is a sophisticated operating system developed and maintained as open-source software by a team of more than 350 individuals located throughout the world. This study uses developer location data, the configuration management repository, and records from the issue database to examine the extent of global development and its effect on productivity, quality, and developer cooperation. The key findings are that global development allows round-the-clock work, but there are some marked differences between the type of work performed at different regions. The effects of multiple dispersed developers on the quality of code and productivity are negligible. Mentoring appears to be sometimes associated with developers living closer together, but ad-hoc cooperation seems to work fine across continents.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Programming teams*; K.6.4 [**Management of Computing and Information Systems**]: System Management—*Centralization/decentralization*

General Terms

Management,Measurement

Keywords

Global development, Open source, Quantitative analysis

1. INTRODUCTION

FreeBSD [20] is a sophisticated operating system available for a number of modern architectures. It is a complete operating system (rather than just a kernel, like Linux) derived from BSD Unix, the version of Unix developed at the University of California, Berkeley. FreeBSD, known for its stability and reliability, runs the servers of large portals like Yahoo and hosting providers like the Host Department; parts of it also form the basis for Apple's Mac OS X. Given the global nature of the FreeBSD development model, the objective of this work is to examine its extent, determine its effects on quality and productivity, and explore how geographic distance affects cooperation among the project's members.

GSD'06, May 23, 2006, Shanghai, China.

1.1 Related Work

Global software development has been the subject of two monographs, a special issue of IEEE Software, and a number of ICSE workshops; see [15, 4, 12, 5, 17, 9] and the references therein.

In 1999 Herbsleb and Grinter identified the challenges faced by splitting the development among various sites by observing that the structure of code often mirrors the structure of the organization that developed it [10]. They also proposed the adoption and use of direct communication channels between developers as a way to overcome the problems they described. The importance of these channels in software development had already been established in 1995 by Kraut and Streeter [16]. In the introduction of the 2001 IEEE Software theme issue, Herbsleb and Moitra outlined the problems of global software development by identifying several dimensions of it: strategic, cultural, and technical issues; knowledge, project, and process management; and inadequate communication. Many of those challenges were first identified in the context of MIS software development in a 1997 conference paper by Erran Carmel [3].

A number of researchers have used data from free and open source software [6, 24] and commercial projects [1, 13] to examine issues of global software development. Most of those studies adopt an empirical qualitative approach. Closer to the approach I adopt in this paper is the 2003 work by Herbsleb and Mockus [11], who used data from a change management system in conjunction with a survey to arrive at the finding that assignments to distributed teams take a lot longer to complete than corresponding assignments to collocated developers.

The work reported here leverages on the FreeBSD development model to provide additional quantitative data points for the study of global software development. Specifically, I investigate practices across a large number of widely dispersed developers and I integrate data from the configuration management system, the actual source code, the geographic coordinates of the developers, and the issue reporting system.

1.2 The FreeBSD Development Model

FreeBSD is developed and maintained as open-source software by a team of more than 350 individuals located through-

out the world. Work can be roughly divided into code written for the system kernel, the operating system utilities, the porting of third-party programs, and the documentation. The global development effort is coordinated through a number of facilities [7, 23].

- A configuration management system repository, based on CVS, houses the current version of all the project's source and documentation files, maintenance branches of older versions, and more than 10 years of historical data. The complete repository is available for public download and for browsing through a web-based interface.
- A problem reports database, based on the GNATS system, contains descriptions of open and closed issues, the individuals dealing with them, and details of the resolution history.
- More than 100 open and closed mailing lists provide a broadcast mechanism for developers and end-users. The lists cover various development areas (such as security or testing), hardware and processor architectures, and releases of the system.
- A so-called *tinderbox* system continuously performs complete builds of the current source code, providing an early indication of any problems committed to the source repository.
- A public web site contains the Developer's Handbook, up-to-date release engineering information, a browsable version of the CVS repository, mailing list archives, and a (read-only) interface to the problem reports database.
- A network of machines accessible to all FreeBSD committers over the internet provides developers with a common workspace for compiling and testing their code on different machine architectures.

Developers are mostly unpaid volunteers, although companies with vested interests in the system also have developers contribute as part of their job. An elected core team is responsible for deciding the project's overall goals and direction, approving proposals for new developers to join the project, and resolving differences. Separate teams handle release engineering, third-party ports, donations, and security. Developers have the right to modify any part of the system (a *commit privilege*), subject to a few formal and many informal restrictions. For example, developers are not allowed to commit changes while a *code freeze* is in place without prior approval from the release engineering team. Also, heavy modifications on code actively maintained by another developer, or changes directly undoing another developer's work are frowned upon.

Developers typically start as enthusiastic contributors of project code; at some point another developer will take interest in their work and recommend them for granting commit privileges. New developers are initially assigned a *mentor* who oversees their work, and approves the changes they make to the code.

2. METHODOLOGY

This work explores elements of the FreeBSD global development model through a quantitative analysis of data obtained from the CVS repository, the problem reports database, and the developers' geographic coordinates.

I derived the CVS data from a snapshot of the CVS repository taken on November 9th, 2005, and examined through the CVS client front-end in conjunction with a number of Unix tools, such as *awk*, *sed*, *sort*, and *join*. The repository covers about 19 million lines of code and documentation, and contains 938,00 commit messages from 475 different committers, made in a period of about 10 years.

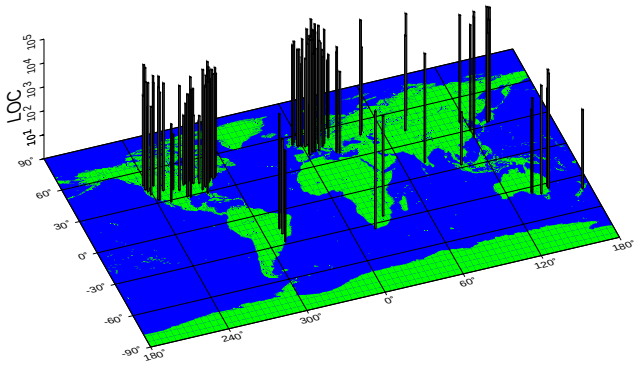

Figure 1: Development effort throughout the world

I also examined the problem reports database by directly scanning the GNATS system files on the project's shell login server, in December 2005. At that time the repository contained around 90,000 reports.

Both CVS commit messages and GNATS reports are tagged with the login name of the corresponding developer. By establishing a relationship between developers and their locations throughout the globe one should be able to derive a number of interesting results. To that end I used a file distributed together with the FreeBSD port of the XEarth application, which contains the latitude, longitude, login name, and location name of many FreeBSD developers. Through appeals to the developer community and a wholehearted response, I was able to increase the coverage of the commit lines that could be attributed to a specific global location from 71% on November 17th, 2005 to 79% on January 9th, 2006. At the time of this writing I had secured location data for 292 developers. I made no attempt to take into account developers who moved place during the time covered by the CVS data.

In a number of cases I calculated distances between developers. For this purpose I used the spherical law of cosines formula, which gives the distance d between two points with latitude δ and longitude λ on a sphere with radius R as,

$$d = R \cos^{-1}[\sin\delta_1 \sin\delta_2 + \cos\delta_1 \cos\delta_2 \cos(\lambda_2 - \lambda_1)]$$

The formula does not take into account the Earth's ellipsoidal shape, but is accurate enough for the purposes of this work.

3. ASPECTS OF GLOBAL DEVELOPMENT

An important question facing potential adopters of global development methodologies is whether these can work in practice. By studying the extent and nature of global development in FreeBSD—by all accounts a successful project—we can establish a baseline for the state of the art.

The project's 292 developers for which I obtained location data live in 206 different locations throughout the world; the lines of code contributed by each location are depicted as vertical bars on the map in Figure 1. As we can see from the Figure, most developers reside in North America and

Europe, with pockets appearing in Asia, Australia, South Africa and South America.

3.1 Specialization by Area

One question worth resolving is whether various areas of the world specialize in specific activities: work on a specific part of the system, or a specific type of work. Given that FreeBSD is a volunteer effort where developers do whatever work they enjoy, one could argue that division of effort across various regions would reflect organizational, cultural, or technological factors that managers of global development projects should take into account when planning the distribution of work.

The main development of FreeBSD appears to be happening in North America (46% of the committed lines), Europe (39% of the committed lines), and Asia (10% of the committed lines). Australia, South America, and Africa trail with 2.5%, 1.6%, and 0.8% respectively. The bulk of the development occurs in the project's source code (49% of the committed lines), with the porting and packaging of third-party software accounting for another sizable 47%. Documentation and the project's web site take the remaining 2.1% and 1.7% respectively. We can observe two interesting facts in the division of labor across regions.

First of all, the work performed by Asian committers is by an overwhelming proportion (80%) related to the porting and packaging of existing applications. On could attribute this difference to cultural factors, but I don't believe that this work has sufficient data to make such claims.

The second interesting aspect I found concerns differences between maintenance and development work. New development in the FreeBSD occurs in the current, active development branch of the configuration management system. Developers are also encouraged to back-port, where possible, enhancements and error corrections to the stable branches of the older maintained versions following a procedure known as "merge from current" (MFC). Contributions to the stable branches are more likely to represent maintenance work. Surprisingly, the distribution between the two work types across regions is not the same. Whereas in North America work is roughly equally divided between contributions to the active and the stable branches (56% versus 44%), the distribution in Europe and Asia favors the active branch with about 70% of the lines committed to it.

I can propose two theories for explaining these differences. One is that there are more FreeBSD-based production systems, such as Yahoo, in North America than in the other regions where more developers use FreeBSD on their personal workstations. Production systems tend to run stable versions of the system, and we can therefore expect North American developers to have an active interest in maintaining them. Another explanation might involve the composition of the release engineering team. Members of that team are responsible for coordinating the FreeBSD releases, and one might think that they have a higher interest than other developers in maintaining the stable versions, either on their own, or with the help of local developers they know. However the team's composition is not so sharply polarized as to explain this difference. Although five of the six current team members have US ties, two have also UK ties, and one lives in Japan.

The picture is also complicated by the different distribution appearing in the developers who resolve entries in the FreeBSD issue database. Here Europe with 41% of the resolved issues leads North America, which accounts for 36% of the resolved issues, while Asia and Australia trail with 12% and 7%, respectively. Apparently, working through entries of the issues database is a task orthogonal to the one of maintaining the stable versions.

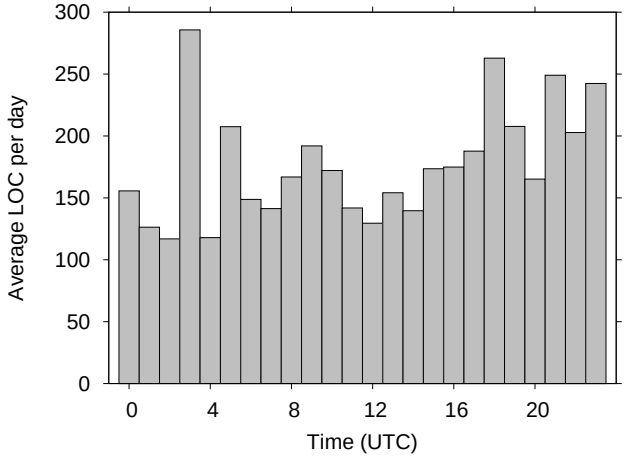

Figure 2: Round-the-clock development

3.2 Round-the-Clock Development

One often-claimed advantage of global software development is the ability to develop software round-the-clock in a continuous 24 hour cycle. In Figure 2 we can see that this goal is indeed realized in the FreeBSD project. For the past ten years, FreeBSD developers committed on average 177 lines on every hour of each day; this number fluctuated between a minimum of 116 lines (at 02:00 UTC) and a maximum of 285 lines (at 03:00 UTC).

A question related to round-the-clock development is the granularity of the work items processed. Is work on a given file passed from one location to the next, are far-away developers cooperating on large modules, or are the development responsibilities divided into different areas? Answering this question allows us to establish a method of round-the-clock development that has worked in practice. To tackle this problem I observed the commit logs at the level of individual files, complete modules, or the whole system, through various sliding windows covering 8-hour and daily intervals. By looking at commit messages in a given window I counted the number of days in which commits occurred, the number of days with 8-hour-far commits by the same committers, the number of days with next-day commits, and the number of days with 8-hour-far commits by different committers. These numbers allow us to see both normal work patterns and patterns likely to be associated with round-the-clock development.

At the level of files, round-the-clock development does not appear to be very prevalent. Only in 1% of the days was a commit followed after 8 hours by a commit from a different developer; in contrast work periods of the same developer spanning more than 8 hours occur in 1.7% of the days changes are committed into a file. Apparently (and quite reasonably) developers prefer to stretch a long period of work than hand out the work to be completed by somebody else. When we turn our attention to modules the situation is reversed: 5.1% of a module's work days contain

commits spanning more than 8 hours by the same developer, while a whole 12.5% of the work days contain commits by different developers. Here it looks like developers from different parts of the world cooperate together on the same module, working across different timezones. Not surprisingly, the same story appears at the system level where in 96% of the days commits by different developers will span 8-hour periods, while in only 50% of the project's work days will a developer work for more than 8 hours. Incidentally, work around a normal 8-hour day period (rather than work stretches longer than 8 hours) appears to be the prevalent pattern across all three divisions. Commits by the same developer on the next day occur in 3.6% of the days for files, in 18% of the days for modules, and in 97.8% of the days for the complete system.

4. PRODUCTIVITY AND QUALITY

Before embracing global software development one would like to know how this type of work will affect the productivity of programmers and the quality of the deliverables. In theory, we can see factors that could affect these variables in positive ways (for example round-the-clock development [14]), in negative ways (for example lack of face-to-face communication [16, 10]), and in ways that are indeterminate (think of cultural diversity [19]). I therefore tried to extract from the data at hand measures that could be used to examine correlation between metrics of global software development, and productivity or software quality.

4.1 Productivity Effects

Productivity is typically measured as output per unit of input. Unfortunately, though not surprisingly, the FreeBSD project does not keep data on hours each volunteer developer works on the code. As the next best thing I tried to see whether the geographical distance between developers working on a module affects the number of lines that are committed in it. For various modules of the FreeBSD system I measured the average number of lines committed over a rolling one-month interval, the length of time development went on for a given module, the number of commits performed, the number of different committers, and the average geographical distance between the committers. From the 1,300 modules I examined, I removed modules being developed outside the FreeBSD project (contributed by other groups), and modules with less of 1,000 lines committed over their entire development history. That left me with 463 modules.

As a base case I examined the correlation between the average number of lines committed per month and the percentage of commits made by different committers. Intuitively one might expect—Brooks's law [2] notwithstanding—that more people working on a given module would contribute more code. (In the next subsections we will examine the quality of that code.) Indeed, I found a positive correlation between those two measures: a Pearson's product-moment correlation of 0.67 in a 95% confidence interval between 0.62 and 0.72. This base case establishes that committers can indeed be used as a proxy for measuring the productivity's input.

Next I examined the correlation between the average number of lines committed per month and the average geographical distance of the developers committing them. Given that the base case established a correlation between committers and the number of lines committed, we would expect that any relationship between the distance of those committers and the work produced would show up as a correlation in this case as well. However, a two-sided Pearson's product-moment correlation test on those two measures came up only with an extremely low correlation of -0.14 in a 95% confidence interval between -0.22 and -0.04. The corresponding coefficient of determination (r^2) explains less than 2% of the variance between the two measures, and we can therefore conclude that in our case the geographic distance between developers does not significantly affect productivity.

4.2 Effects on Code Style

I also examined how a large number of (geographically dispersed) committers might affect the quality of the produced code. If the software's quality deteriorates when software is globally developed, managers should appreciate this problem, and establish procedures for dealing with it. The quality of code is determined by many elements [25], and measuring it is far from trivial [27, 22]; For the purpose of this study I chose to examine adherence to the FreeBSD code style guidelines [8] as a proxy for the overall code quality. I chose this metric because I could easily measure style adherence by formatting each source code file with the *indent* program configured according to the FreeBSD style guide, and calculate the percentage of lines that *indent* would change (the size of a minimal set of differences between the actual file and the formatted one). Furthermore, by having CVS generate a listing of the source code file with every line annotated with the name of the author who last modified it, I could count the number of developers who had worked on the file.

Armed with those two measurements, I used again Pearson's product-moment method to examine correlation between the two. The correlation coefficient for the 11,040 pairs of measurements was a miserly 0.05 in a 95% confidence interval between 0.03 and 0.07. We therefore see that in our case, the involvement of geographically dispersed programmers in the development of code does not affect the quality of the produced code.

4.3 Effects on Defect Density

Finally, I examined whether the global development of a file by various developers was associated with an increased number of problem reports filed for it. Such a correlation could indicate that global development in the FreeBSD project leads to an increased number of bugs in the code, due, for example, to communication problems between the various developers. Although problem reports are kept in a database different from that of the FreeBSD configuration management system, rectified problems are typically marked in a CVS commit message by a reference to the corresponding problem report (PR). Because serious problem reports are by definition sooner or later rectified, I could establish a measure of the density of problem reports in a file by dividing the number of commit messages tagged with a PR number with the total number of the file's commits. I could then examine the correlation of that ratio with the number of different developers that had committed code to the corresponding file.

I collected data for 33,392 source code files, 457,481 commit messages, and 12,505 PRs. On average, each file was associated with 13.7 commits, 0.37 PRs, and 4.2 different

	Min.	1/4	Median	Mean	3/4	Max.
Any	0	2,215	7,793	6,702	9,380	19,390
M-M	0	745	3,856	5,080	8,801	18,650

Table 1: Developer and Mentor-Mentee Distances

developers. A two sided Pearson's product-moment correlation test between the PR density and the number of committers gave an insignificant correlation between the two values (0.07) in a 95% confidence interval between 0.06 and 0.08. Therefore, the data from the FreeBSD project does not support the hypothesis that global software development is associated with a higher bug density in the code produced.

5. HUMAN INTERACTIONS

It would be short-sighted to study global software development only in terms of the resulting product. Organizations also serve and fulfill innate human needs and drives of their members, such as those of acquiring, bonding, learning, and defending [18]. I therefore used data from the FreeBSD project to examine how the distance between developers affects the network of their relationships. I focused on two types of associations between developers: cooperation toward a common goal, and learning in a mentor-mentee relationship. For both types of association I measured the geographical distance between related developers and compared it to the average distance between developers (6,701km). A markedly lower distance between cooperating developers and the average could mean that developers prefer to cooperate with those nearby. From such a result one could theorize that geographical distance puts a strain in those associations.

I obtained a list of cooperating developers by scanning the commit logs with a rolling window of a single day, looking for different developers who had committed code on the same file within that day. I assumed that such instances would indicate cooperation between those developers, because of the changes' proximity in the code space and time. From the data I established 5,847 instances of cooperation between developers, and from those instances I kept the 4,010 for which I had at hand the geographic coordinates of the two developers. The average distance between cooperating developers is 6,489km, a number very close to the average distance between any two developers. This fact indicates that in the FreeBSD project technical developer cooperation is seldom influenced by the location of the developers.

During the time new FreeBSD developers work with a mentor, they tag all their commit messages with a line indicating the name of the mentor who reviewed and approved the corresponding change. By scanning those messages I established a list of 167 mentor-mentee pairs. From those I kept the 107 for which I had the locations for both members of the couple, and used the developer coordinates to calculate the distance between the mentor and the mentee (Figure 3). As one can see in Table 1, which summarizes the km distances between any two developers and developers in a mentor-mentee relationship, the mean and median distances between mentors and mentees are lower, but not dramatically lower than the those between any random FreeBSD developers. In the mentoring case, even the numbers in the first quartile designate distances within a small country or state, not a city.

It therefore seems that some mentor-mentee relationships are established between people in the same area (see for example the *fjoe–danfe* pair on the top right of Figure 3), but such relationships can (and do) also work across continents.

6. DISCUSSION AND CONCLUSIONS

The findings in the previous sections indicate that software development by a widely dispersed loosely-coupled team of developers is a practical proposition. The global distribution of the team members allows round-the-clock development to take place, with no apparent ill effects on productivity, the quality of the code, and the density of defects. Ad-hoc cooperation on specific work items does not seem to be affected by distance. On the other hand, I found that the mentoring relationship appears in some cases to be easier to cultivate between individuals living closer together.

Some of my findings may be counterintuitive and even contradict those of earlier studies that found diminished productivity among distributed teams [11]. One should however take into account two limitations of this work. First of all, because the FreeBSD project does not offer traditional offices where collocated developers can cooperate with face-to-face contact, the base case for the results reported here is developers working on their own or with developers living in the same area. In addition, because FreeBSD is a project built mainly by volunteers vetted by their peers, a number of factors differ from what one would expect to find in an average software development shop: all developers are extremely motivated and highly competent, developers freely choose the type and amount of work they will undertake and when they will deliver it, and developers are typically also users of the FreeBSD system. These factors should be further examined in the context of the relationship between open and closed source software development [26, 21].

Based on the findings I outlined, two interesting questions that one can pose are:

1. What is the meaning of distance in the context of global software development?
2. What lessons can commercial software developers derive from the results?

Clearly, when we examine problems of global software development, there's more to distance than geographical separation. This is something we should take into account both when we are looking at the distance between various locations, and when we are using an existing, supposedly non-global, setup as a baseline. Distance can appear in a number of different orthogonal dimensions. The *physical distance* we saw at the end of Section 2 can be further elaborated by looking into the *travel distance* between developers (commuting from one edge to another in a large metropolis or across a border may be more difficult than traveling from one central-European city to another). We should also subdivide physical distance into *on-work* and *off-work* distance. I use the last two terms refer to the distance between developers when they are working (they could work on the same or on different sites), and when they are resting (they could share communal spaces, or hang out in the evening and on weekends at the same venues). Then there is *cultural distance*: this refers to differences in the language (or even dialect) spoken by different developers, social norms and conventions, and the culture's predominant work ethic. *Timezone distance*

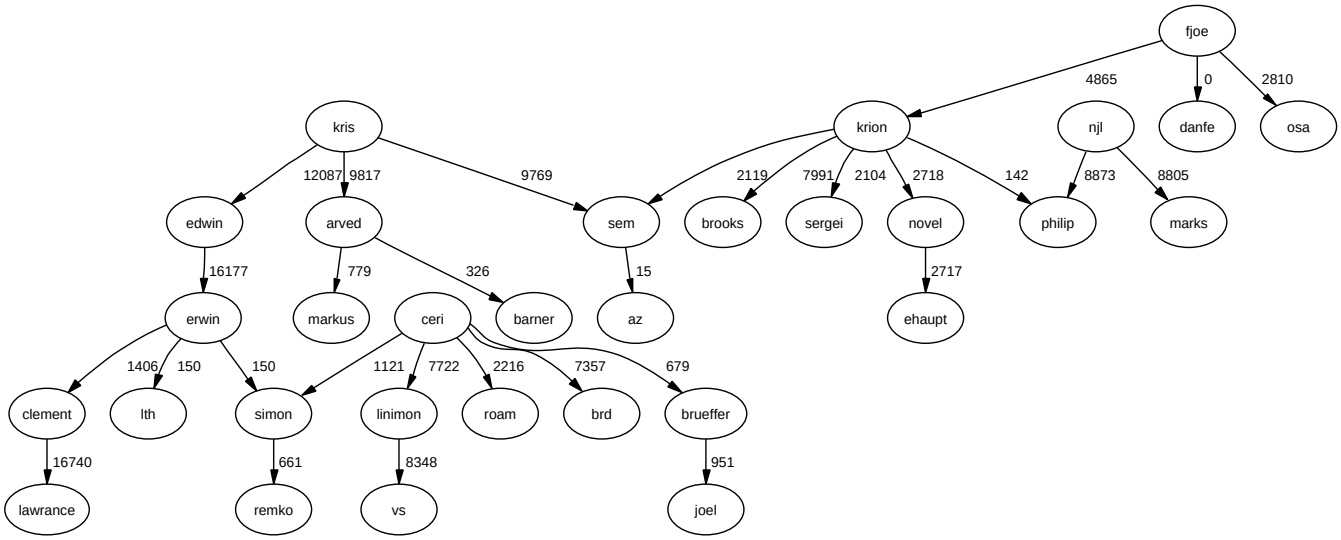

Figure 3: Distances (in km) in a part of the FreeBSD mentor-mentee graph.

can also crop up in remarkably different ways: developers can share a timezone but be far apart, because they live on different latitudes, or because they work on different shifts. Finally, developers' access to various *collaboration technologies*, such as a configuration management system, an issue database, the phone, instant messaging, wikis, and mailing lists, is another underappreciated measure of distance.

The extent to which the results presented here apply to commercial software development is debatable. The lack of a control group where developers would work in the same office complex sharing lunch at a common cafeteria didn't allow me to verify whether FreeBSD is paying the price of global software development even in cases where its programmers live in the same city. On the other hand, many existing commercial software development efforts are also dispersed among physically separated offices or even sites. For such cases, this work has demonstrated that in an environment where developers routinely use a number of essential collaboration technologies, geographic distance becomes immaterial.

Thus, the results described in the previous sections are relevant to practitioners, and they also open some new research questions. Given the generally positive results of this study, commercial software development projects could, at the very least, try to adopt and emulate some of the global development practices of the FreeBSD project. On the research front one could also apply the research methodology of this study to commercial software development projects and see whether the same findings can be replicated there.

Acknowledgments

I wish to thank the members of the FreeBSD community, for allowing me to participate in the project and for providing me with data and comments for this work. Panagiotis Louridas provided many useful comments on earlier drafts of this work.

7. REFERENCES

[1] M. Akmanligil and P. C. Palvia. Strategies for global information systems development. *Information and Management*, 42(1):45–59, 2004.

[2] F. P. Brooks. *The Mythical Man Month*. Addison-Wesley, Reading, MA, 1975.

[3] E. Carmel. Thirteen assertions for globally dispersed software development research. In *Proceedings of the 30th Hawaii Int. Conf. on System Sciences (HICSS-30) — Volume 3: Information System Track — Organizational Systems and Technology*, page 445, 1997.

[4] E. Carmel. *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall, Upper Saddle River, NJ, 1999.

[5] D. Damian. Workshop on global software development. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 667–668, New York, 2002. ACM Press.

[6] M. S. Elliott and W. Scacchi. Free software developers as an occupational community: resolving conflicts and fostering collaboration. In *GROUP '03: Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work*, pages 21–30, New York, 2003. ACM Press.

[7] J. Feller and B. Fitzgerald. *Understanding Open Source Software Development*. Addison-Wesley, Reading, MA, 2001.

[8] The FreeBSD Project. *Style—Kernel Source File Style Guide*, Dec. 1995. FreeBSD Kernel Developer's Manual: style(9). Available online `http://www.freebsd.org/docs.html` (January 2006).

[9] E. Hargreaves, D. Damian, F. Lanubile, and J. Chisan. Global software development: Building a research community. *SIGSOFT Software Engineering Notes*, 29(5):1–5, 2004.

[10] J. D. Herbsleb and R. E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 85–95, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[11] J. D. Herbsleb and A. Mockus. An empirical study of

speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29(6):481–494, June 2003.

[12] J. D. Herbsleb and D. Moitra. Global software development. *IEEE Software*, 18(2):16–20, March/April 2001.

[13] J. D. Herbsleb, D. J. Paulish, and M. Bass. Global software development at Siemens: Experience from nine projects. In *ICSE '05: Proceedings of the 27th International Conference on Software Engineering*, pages 524–533, New York, 2005. ACM Press.

[14] P. Jalote and G. Jain. Assigning tasks in a 24-hour software development model. In *11th Asia-Pacific Software Engineering Conference*, pages 309–315, 2004.

[15] D. W. Karolak. *Global Software Development: Managing Virtual Teams and Environments*. Wiley—IEEE CS Press, New York, 1998.

[16] R. E. Kraut and L. A. Streeter. Coordination in software development. *Communications of the ACM*, 38(3):69–81, 1995.

[17] F. Lanubile, D. Damian, and H. L. Oppenheimer. Global software development: Technical, organizational, and social challenges. *SIGSOFT Software Engineering Notes*, 28(6):2–2, 2003.

[18] P. Lawrence and N. Nohria. *Driven: How Human Nature Shapes Our Choices*. Wiley, New York, 2001.

[19] E. MacGregor, Y. Hsieh, and P. Kruchten. Cultural patterns in software process mishaps: incidents in global projects. In *HSSE '05: Proceedings of the 2005 Workshop on Human and Social Factors of Software Engineering*, pages 1–5, New York, 2005. ACM Press.

[20] M. K. McKusick and G. V. Neville-Neil. *The Design and Implementation of the FreeBSD Operating System*. Addison-Wesley, Reading, MA, 2004.

[21] J. W. Paulson, G. Succi, and A. Eberlein. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30(4):246–256, Apr. 2004.

[22] C. Payne. On the security of open source software. *Information Systems Journal*, 12(1):61–78, 2002.

[23] N. Saers. *A project model for the FreeBSD Project*. PhD thesis, University of Oslo, May 2003. Available online `http://niklas.saers.com/thesis/thesis.html`.

[24] R. J. Sandusky and L. Gasser. Negotiation and the coordination of information and activity in distributed software problem management. In *GROUP '05: Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work*, pages 187–196, New York, 2005. ACM Press.

[25] D. Spinellis. *Code Quality: The Open Source Perspective*. Addison-Wesley, Boston, MA, 2006.

[26] D. Spinellis and C. Szyperski. How is open source affecting software development? *IEEE Software*, 21(1):28–33, January/February 2004.

[27] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris. Code quality analysis in open source software development. *Information Systems Journal*, 12(1):43–60, 2002.