Socio-Technical Dependencies in Forked OSS Projects: Evidence from the BSD Family

M.M. Mahbubul Syeed[a], Imed Hammouda[b]
[a] Department of of Pervasive Computing, Tampere University of Technology, Finland.
Email: mm.syeed@tut.fi
[b] Chalmers and University of Gothenburg, Sweden.
Email: imed.hammouda@cse.gu.se

***Abstract*— Existing studies show that open source projects may enjoy high level of socio-technical congruence despite their open and distributed character. Such observation is yet to be confirmed in the case of forking, where projects originating from the same root evolve in parallel and are typically lead by different development teams. In this paper, we empirically investigate the endogenous and exogenous characteristics of BSD family projects related to socio-technical congruence. Our motivation is that BSD family, as a representative example of forked projects, share a common development ground for both the code-base and the development community, which may influence their evolution from a socio-technical perspective. Our study results show that the BSD family maintain a certain level of collaboration throughout the project history, mainly due to a shared portion of the community. This partly explains the relative harmony of socio-technical congruence levels in the BSD projects.**

***Index Terms*— Open Source Software, Evolution, Conway's Law, Socio-Technical Congruence, Forking**

I. Introduction

SOFTWARE development requires effective communication, coordination, and collaboration among developers working on interdependent modules of the same project. The need for coordination is even more evident in Open Source Software (OSS) projects where development is often more dispersed and distributed [1]. As argued in the literature, such coordination and communication may be influenced and guided by the cooperation needs devised by the design of the software [2]. This suggests that there might exist a two way mapping between the communication patterns of the developer community and the architectural dependencies among the components of the software, in which one can be used to approximate the other.

This collaboration can effectively be examined and verified through the notion of socio-technical congruence which defines the match between the coordination needs established by the technical domain (i.e., the architectural dependencies in the software) and the actual coordination activities carried out by project members (i.e., within the members of the development team) [3].

This work was supported in part by TiSE Graduate School, Tampere, Finland and Nokia Foundation Grant, Finland.

In fact, socio-technical congruence provides an empirical verification of a well-known but insufficiently understood phenomenon known as Conway's Law [4] and describes to which extent the law is enforced in a given software development project [3] [5]. Such empirical verification has been a primary motivation for many research efforts in the realm of socio-technical congruence [6] [7]. However, research on the topic has always assumed that development of a software project is performed by the same organization or group of developers. In the case of open source, projects may evolve in parallel, lead by different development teams. This is known as "forking" [8]. To the knowledge of the authors, no research has been performed yet on socio-technical congruence in the context of forked projects.

In an earlier work we have studied socio-technical congruence, and the significance of Conway's Law, in the FreeBSD open source project [9]. Our previous study showed that the congruence measure is significantly high in FreeBSD and that the congruence value remains stable as the project matured. In this work, we extend our earlier study to cover the BSD project family, empirically investigating the endogenous and exogenous characteristics of BSD projects. BSD projects are popularly known in the research community. For instance, in [10], change history information is extracted from BSD projects for the visualization of change dependencies. Similarly, FreeBSD project has been studied to verify the viability of incremental development approach, and to identify the common characteristics of successful OSS development process in relation to their quality, in [11] and [12], respectively.

Within the endogenous characteristics we investigated the notion of socio-technical dependency through the measure of socio-technical congruence in the individual projects. In the technical domain, the architectural dependencies have been constructed out of source code syntactic information such as functional dependency, attribute referencing and header file inclusion dependency. On the social side, the coordination network has been built out of email conversations between developers.

Among the exogenous characteristics, we examined to what extent the forked projects collaborate and communicate with each other. As a measuring criteria of such

doi:10.4304/jsw.9.11.2895-2909

collaboration, we quantitatively measured the alignment among the source code and the developer community of the forked projects. The rationale here is that forked projects hold the same root for both the code-base and the community, thus sharing a common development ground. Hereof it is worth to empirically investigate the extent to which such common ground is maintained during projects evolution.

The remaining of the paper is organized as follows. Section II introduces a number of key concepts that this study uses. Section III introduces the research questions explored and Section IV presents our study design. Results are reported and discussed in Section V, followed by final discussion and related work in Section VI. The overall impact of missing data on the reported results and the replication guidelines are presented in Section VII. Possible limitations and threats to validity are highlighted in Section VIII. Finally, Section IX concludes the paper and sheds light on future research.

II. Definitions

In this section we define a set of concepts used in this study.

A. Conway's Law

Conway's Law in its purest form states that "organizations which design systems are constrained to produce systems which are copies of the communication structures of these organizations" [4]. In other words, the software product architecture reflects the organizational structure of its development team [4] [3]. In [13], Conway's Law is considered homomorphic and thus claimed to be true in reverse as well. This means the communication pattern within a developer community should reflect the architectural dependency in the developed software. Thus, Conway's Law can effectively be interpreted as the basis for studying the social and technical interdependency within a software project [14].

B. Socio-technical congruence

The contemporary phenomenon "Socio-technical congruence" is actually the conceptualization of Conway's Law. Socio-technical congruence can be defined as the match between the coordination needs established by the technical domain (i.e., the architectural dependency in the software) and the actual coordination activities carried out by project members (i.e., within the members of the developer community) [3]. This coordination need can be determined by analyzing the assignments of people to a technical entity such as a source code module, and the technical dependencies among the technical entities [3]. Accordingly, developers within the community should communicate if there exists a communication need. For example, developers working on the same module or on the interdependent modules should be coordinating.

C. Developer Contribution

In this work, developer contribution to a software project can be defined as code contribution or any form of commit made to the code base.

D. Explicit Architecture

The explicit architecture of a software presents the relationship among components of a software (e.g., modules, files or packages) based on the actual design and implementation. For this work, functional dependency, attribute referencing and header file inclusion dependency at code file level are used to derive the Explicit Architecture of a software product.

E. Explicit Coordination Network

The explicit coordination network is a social network in which two developers have a relationship if they have direct communication history as seen by the mailing archives representing the social and technical interactions among the developers.

F. Implicit Architecture

The implicit architecture defines an architecture of the software where any two components (e.g., packages or code files) are related if there are developers who have either (a) contributed to both components, or (b) have direct communication at organizational level (e.g., a one to one email conversation). For instance, consider that developer D1 has contributed to packages P1 and P2, and developer D2 has contributed to package P3. Also consider that both developers have direct communication at organizational level as shown in Fig. 1(a). Thus according to the definition, packages P1, P2 and P3 are linked to each other in the Implicit Architecture (Fig. 1(b)).

G. Implicit Coordination Network

The implicit coordination network is the developer relationship network in which two developers have a relationship if they have contributed either (a) to a common code file or (b) to the code files that have direct relationships in the Explicit Architecture. For instance consider that developers D1 and D2 have contributed to package P1 and developer D3 has contributed to package P2. Also consider that P1 and P2 have a functional dependency (i.e., a direct relationship) as shown in Fig. 2(a). Then, according to the definition, developers D1, D2 and D3 are linked to each other in the Implicit Coordination Network as shown in Fig. 2(b).

H. Forking

In the context of open source development, *forking* occurs when a part of a development community (or a third party not related to the original project) starts a completely independent line of development based on the source code of the original project [8] [15]. To be considered as a fork, a project should have:

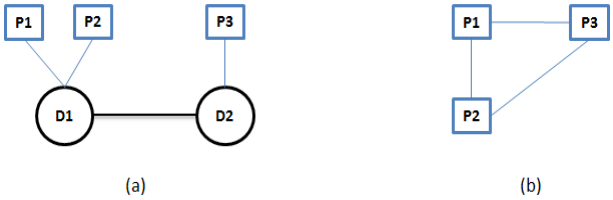


Figure 1. (a) Explicit Coordination Network with contribution to code base (b) Corresponding Implicit Architecture

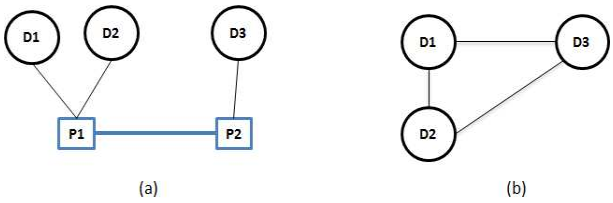


Figure 2. (a) Explicit Architecture with contributing developers (b) Corresponding Implicit Coordination Network

- A new project name.
- A branch of the software.
- A parallel infrastructure (web site, version control system, mailing lists, etc.).
- And a new developer community.

Based on this definition, we propose the following set of relationships within a pair of forked projects: (a) parent-child, in which one project is forked from the other, (b) siblings, if two projects are forked from the same parent project, and (c) lineages, for all descendant relationships in which (a) and (b) do not hold. For example, in Fig. 3, NetBSD and OpenBSD have a parent-child relationship, FreeBSD and NetBSD are sibling projects, whereas FreeBSD, and OpenBSD are the lineages of 386BSD.

III. Research Questions

Our choice of research questions is motivated by our agenda to measure the exogenous and endogenous characteristics of forked OSS projects related to socio-technical congruence.

(RQ1) How does the software architecture compare and evolve across forked OSS projects?

When a project is forked, the source code of the parent project is copied [8]. Thus it is natural that at the initial stage the source code, and hence the architecture, of both systems are similar. Based on this observation, we are interested in exploring the extent to which the forked projects share common architectural structure during their evolution. In doing so, we calculated and compared the architectures of the forked projects at three abstraction levels. Namely, at package level, at first directory level and at n^{th} directory level (i.e., the last directory where the files reside). We argue that this three level comparison can provide a holistic view of the architectural overlapping. For instance, it might be possible that forked projects maintain homogeneous architectural design at higher level of abstraction (e.g., in package level), yet getting liberated at detailed architectural level (e.g., n^{th} directory level).

(RQ2) How does the community compare and evolve across forked OSS projects?

Traditionally, the developer community divides when a project is forked [8]. This is typically followed by a community rebuild and restructuring process in both projects. Community members in both projects might communicate and coordinate in such circumstances in making both projects survive. Thus, our intention here is to examine how these fragmented communities act in building the projects: do they contribute to both projects? Does such collaboration sustain during the evolution of the projects?

(RQ3) How does the socio-technical congruence evolve within the forked OSS projects?

Socio-technical congruence is a natural consequence and a desired property for collaborative development activities, like OSS projects [16]. Conventional wisdom suggests that correspondence between the social and technical domain of a project may reduce the communication overhead and may increase productivity [7]. Furthermore, lack of collaboration is classified as a negative stimuli to performance [17] and has an influence on lowering productivity [5]. Consequently, socio-technical congruence can be a decisive property of a successful project [3] [5] [16]. With strong congruence measure projects can get more cohesive, organized, and self-dependent with higher productivity. Thus our intention here is to stress these reported observations in forked OSS projects by examining the extent to which Socio-Technical Congruence holds in forked projects.

IV. Study Design

This section presents in detail our study design, covering discussion on the case study selection, required data sets, data acquisition, cleaning, and analysis process.

A. Case and Subject Selection

To explore the three research questions, we performed a case study with three large (more than 1,414,641 LOC [18]), long-lived (around 20 years of evolution history) OSS projects that were forked from their predecessors. These case study projects are FreeBSD [19], NetBSD [20] and OpenBSD [21]. All three projects originate from the 386BSD project, which is the version of UNIX developed at the University of California, Berkeley. FreeBSD and NetBSD were directly forked from 386BSD during late 1993, and therefore have a sibling relationship. OpenBSD was forked from NetBSD in 1995, thus having a parent-child forking relationship. Whereas, FreeBSD and OpenBSD are lineages of 386BSD. As a consequence, the core of these projects encompass the code base of 386BSD. The forked relationship among these projects are shown in Fig. 3 and the lifetime of these projects till 2013 are shown in Fig. 4.

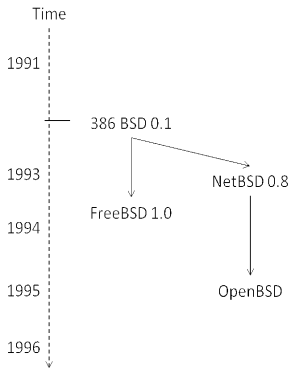


Figure 3. Rough time line of the forked BSD projects

Our selection of the BSD project family was influenced by the following factors: (a) the code base of these projects have undergone continuous development, improvement, and optimization for twenty years [19], (b) these projects have been developed and maintained by a large team of individuals [20], (c) the properties of a forked project hold for these projects, (d) these projects have extensively been used in earlier research on the evolution of OSS projects [22] [23] [18], and (e) results reported in this study can be stressed to OSS projects having similar properties, e.g., forking history, domain, community structure, and size.

B. Data Sets

OSS projects often consist of a number of software development repositories. These repositories contain a plethora of information on both the underlying software and the associated communication and development process [24] [25]. In the literature [26] a great

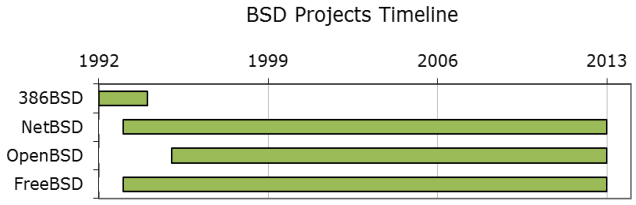


Figure 4. Life time of the BSD projects

emphasis was given to leveraging these repositories for deriving technical dependencies as well as developers' coordination patterns. The repository data are often longitudinal, allowing for analysis along the whole project evolution phases. Such data sources are highly accepted and utilized medium for empirical studies on OSS projects [27] [28] [29]. In this study we utilized the following repositories.

Source code repository: We downloaded the source code of each stable release of the three projects. FreeBSD maintains its source code in Subversion version control system, whereas NetBSD and OpenBSD use CVS. In Fig. 5 we provide the details of the stable releases, the data collected from each release, and the corresponding download sources.

Mailing list archive: In OSS projects, email archives provide a useful trace of task-oriented communication and co-ordination activities of the developers during project evolution [30]. In the studied projects, email archives are categorized according to their purpose including commit records, stable release planning, chat, user emails, and bug reports. The archives contain the commit history and the email conversations since the initiation of the projects. In this study we used a complete list of commit records and email conversations from the beginning of each studied project. Consequently, data from relevant email archives was extracted and refined from each project, detail of which is presented in Fig. 6.

C. Data Collection

From source code repositories: The source code of each stable release of the selected projects was downloaded to a local directory. Fig. 5 lists the stable releases that were downloaded for each project. To extract data from each of the releases, a parser was written in Java. The parser searched through each directory of a stable release, read through the files in a directory and parsed relevant data. Each code file in a release contains a copyright directive. Under this directive the contributing developer name, email, and the copyright year is mentioned. The developers that were found in the process were considered as the initial contributors to that file. To get a complete list of contributors for a stable release, developers names were extracted from the commit history log and were merged with this contributor list. This process is described in following

Case Study Project	Stable Release Number	No. Of Stable Releases Studied	Programming Language	Data Extracted	Use in Data Analysis
FreeBSD	2.0.5, 2.1, 2.2, 3, 4, 5, 6, 7, 8, 9	10	C/C++	1. File name. 2. File directory path. 3. File package name. 5. Contributing dedevloper information from Copyright tag of each code file. E.g., developer name, year, email address. 6. Number of code files, number of other files, number of packages	1. Generated the partial list of developers for each stable release. This list was combined with the developer's list found in the commit history to generate the complete list for that release. 2. Identified to which code files a developer contributed for a stable release. 3. For each stable release, the Explicit Architecture, and Implicit Coordination Network were generated.
NetBSD	0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 3.0, 4.0, 5.0, 6.0	14	C/C++		
OpenBSD	2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2	29	C/C++		
Download sources:	FreeBSD	http://svn.freebsd.org/base/stable			
	NetBSD	http://www.netbsd.org/docs/guide/en/chap-fetch.html#chap-fetch-cvs-netbsd-release			
	OpenBSD	http://www.openbsd.org/anoncvs.html			
Last Accessed:	January, 2013.				

Figure 5. Stable Releases of BSD Projects (FreeBSD, NetBSD and OpenBSD)

Case Study Project	Email Archives Containing SVN/CVS commits	Duration	Data Extracted	Use in Data Analysis
FreeBSD	cvs-bin, cvs-contrib, cvs-distrib, cvs-doc, cvs-eBones, cvs-etc, cvs-games, cvs-gnu, cvs-include, cvs-kerberosIV, cvs-lib, cvs-libexec, cvs-lkm, cvs-other, cvs-ports, cvs-release, cvs-sbin, cvs-share, cvs-sys, cvs-tools, cvs-user, cvs-usrbin, cvs-usrsbin, cvs-all, svn-src-stable-6, svn-src-stable-7, svn-src-stable-8, svn-src-stable-9, svn-src-stable-other	1994-2012	a. Data extracted from commit records (if the mail is a SVN/CVS commit). 1. committer name. 2. commit subject. 3. commit date and time. 4. commit directory path. 5. commit file(s). 6. package name(s). b. Data extracted from other email archives. 1. Email subject 2. Sender name 3. Sender email 4. Receiver name 5. Receiver email 6. Date and time Posted.	1. Generated the developer list for each stable release from commit records. 2. Identified developer contributions for each stable release. 3. For each stable release, Explicit Coordination network and Implicit Architecture were generated.
NetBSD	source-changes, source-changes-d	1994-2012		
OpenBSD	source-changes	1995-2012		
Last Accessed:	January, 2013.			

Figure 6. FreeBSD, NetBSD and OpenBSD email archives

sections. Information that was extracted using the parser is listed in Fig. 5, column 5. The parsed data for each stable release was then stored in a spreadsheet for further analysis.

From email archives: Data that is maintained in the email archives can be broadly classified into two groups, (a) email archives that maintain CVS/SVN commit records, and (b) archives that store general community discussions (e.g., on stable release planning, chat entries). Fig. 6 presents the total number of email archives that were extracted for each project along with specific names of archives containing the commit records, data collection period, collected data, and their analysis purpose.

For extracting data from each email entry, a data extraction program was written in Java. This data extractor used the web interface of the email archives. Thus each email was read as an HTML page and the data was extracted using the Jsoup HTML parser [31]. Data extracted from each email entry is listed in Fig. 6, column 4. This data was then stored in spreadsheets according to the archive name and year. After that, email data was sorted according to each stable release as follows: (a) emails and commit records were categorized into a specific release if the release number was mentioned in email subject (e.g., SVN commit emails provide release number in email subject for FreeBSD) and (b) other emails for which the release numbers were not mentioned (e.g., freeBSD-stable, freeBSD-chat and some of the CVS commit emails), the posting dates were checked. In this case, for instance, an email was categorized to stable release 3 if its posting date falls between the release date of stable release 2 and 3. The rationale here is that developers would commit to the code base and discuss on its release strategy before it is officially released.

For the CVS/SVN commit email, we parsed the commit path to the repository. The commit path was either mentioned in the subject or in the email body (in specified format). We extracted information like the directory path, package name, and if provided, the name of the modified code file(s) and the stable release number. The name of the committer for each of these CVS/SVN commit emails was considered as a contributor to the

code base. Contributors found in this process were combined with the contributors found in the code base to get a complete list of contributing developers for each stable release.

Data preprocessing: Data that was extracted and parsed following the above process contained anomalies in many cases. For instance, developer names and email addresses might contain punctuation characters like semi-colons, inverted comas, brackets, unnecessary white space, and hyphens. Furthermore, parsers may have parsed data inappropriately in some cases. For example, the text *copyright rights reserved* can be treated as part of developer name while parsing copyright directive from a code file. To clean such anomalies data and punctuation characters, data cleaning programs were written in Java. To ensure the correctness of this process, we performed a manual checking on a randomly selected data to verify their correctness.

D. Data Analysis

This section is focused on topics related to the construction of the communication networks, architectures, and their use in measuring the socio-technical congruence utilizing the collected data.

Data analysis is restricted to the stable releases of the projects. This means, analysis point of this study is the stable release dates for a project. This choice of analysis point (instead of discrete time stamps) is made due to the following reasons: (a) a stable release reflects clear milestone for a project, which can also be counted as a step towards successful evolution, and (b) the source code for this study is available for stable releases only, which makes it obvious choice to take release dates as analysis points.

Developer Contribution: Developer contributions were measured release-wise in two ways: (a) from the copyright information provided in each source code file of a release and (b) from the commits made by a developer for a release. Fig. 7(a) shows a sample contribution made by developer *John Birrell* in FreeBSD stable release 3.

Explicit Architecture: The Explicit Architecture of a stable release was constructed based on functional dependency, attribute referencing, and header file inclusion dependency at code file level. For doing this, we used a tool named Understand [32]. This tool takes a source code repository as input and generates the corresponding Explicit Architecture. This tool has been used in previous research, e.g., in [33] [34]. The explicit architecture for each stable release of a project was derived at two abstraction levels, e.g., at code file level and at package level. An example of these two architectures for FreeBSD release 3 is shown in Fig. 8.

Explicit Coordination Network: Following the definition in Section II-E, the Explicit Coordination Network was derived for each stable release of a project. Email conversations for each stable release were used for this purpose. Fig. 7(b) shows example relationships in the Explicit Coordination Network of FreeBSD stable release 3. The weight column in this figure shows the number of email conversations that took place between two developers.

Implicit Architecture: The implicit architecture was generated following the definition in Section II-F. A partial snapshot of the package level Implicit Architecture for FreeBSD stable release 3 is shown in Fig. 9(a). In this architecture, a link weight between two packages designates the number of times the conditions (from Section II-F) hold. The significance of this network lays in the fact that developer communication patterns within the community may simulate the actual architectural dependency. That is, two developers should have communication if they are contributing to same or interrelated components of the software.

Implicit Coordination Network: This network was generated according to the definition presented in Section II-G. A snapshot of this network for FreeBSD stable release 3 is shown in Fig. 9(b). The network shows the actual communication need among developers, based on the design of the software (i.e., the Explicit Architecture). This network is essential due to the fact that if two subsystems exchange information, it is likely that communication among the developers of the two subsystems exists [4].

Measuring concurring and congruence among architectures and networks: Comparison among the architectures and communities was measured for two purposes: (a) to measure how the software architectures and communities compare and evolve across forked projects, and (b) to identify how the socio-technical congruence evolve within each forked project.

We applied the following similarity measure to serve both purposes. This approach is analogous to the fit measure used in organizational theory method [5]. An identical approach was applied in [9] for measuring the congruence in FreeBSD project.

$$Concurring/Congruence = \frac{Ref_{A/N} \bigcap Analogous_{A/N}}{\left|Ref_{A/N}\right|} \times 100) \quad (1)$$

In the above equation, $Ref_{A/N}$ is the reference architecture or network (either explicit or implicit), and $Analogous_{A/N}$ it the analogous architecture or network (either explicit or implicit) with which concurring or congruence will be measured.

This equation measures concurring between the two architectures or networks with respect to the reference one, $Ref_{A/N}$. Therefore, the numerator of equation (1) identifies the commonalities between the two given ar-

Developer	Package	File Dir	File name	Email	Release
John Birrell	3/lib/	3/lib/libc_r/uthread/	uthread_attr_getstackaddr.c	jb@cimlogic.com.au	stable-3
John Birrell	3/lib/	3/lib/libc_r/uthr	uthread_attr_getstacksize.c	jb@cimlogic.com.	stable-3
John Birrell	3/lib/	3/lib/libc_r/uthread/	uthread_attr_init.c	jb@cimlogic.com.au	stable-3
John Birrell	3/lib/	3/lib/libc_r/uthread/	uthread_attr_setcreatesuspend_np.c	jb@cimlogic.com.au	stable-3
John Birrell	3/lib/	3/lib/libc_r/uthread/	uthread_attr_setdetachstate.c	jb@cimlogic.com.au	stable-3

(a)

Developer name	Developer name	Relationship weight
J Wunsch	Bruce Evans	15
Peter Dufault	Brian Somers	61
Tom Samplonius	Andreas Klemm	6
Mikael Karpberg	Bill Fenner	3

(b)

Figure 7. (a) Sample contributions made by developer John Birrell (b) Sample relationships in Explicit Coordination Network

Source File	Destination File	Source File path	Destination File path
adjkerntz.c	sys/time.h	3/sbin/adjkerntz/adjkerntz.c	3/sys/sys/time.h
adjkerntz.c	sys/param.h	3/sbin/adjkerntz/adjkerntz.c	3/sys/sys/param.h
chkey.c	rpcsvc/ypclnt.h	3/usr.bin/chkey/chkey.c	3/include/rpcsvc/ypclnt.h
ftpd.c	arpa/telnet.h	3/libexec/ftpd/ftpd.c	3/usr.bin/tn3270/distribution/arpa/telnet.h

(a)

Source Package	Destination Package	Relationship Weight
3/sbin/	3/sys/	302
3/usr.bin/	3/include/	82
3/libexec/	3/usr.bin/	4

(b)

Figure 8. (a) Code file level Explicit Architecture (b) Package level Explicit Architecture

source package	destination package	Relationship weight
3/contrib/	3/etc/	424
3/gnu/	3/release/	251
3/contrib/	3/share/	456

(a)

Developer name	Developer name	Relationship weight
Paul Traina	Michael Smith	21
Sun Microsystems	Philippe Charnier	20
John D. Polstra	Julian R. Elischer	6

(b)

Figure 9. (a) Implicit Architecture (b) Implicit Coordination Network

Pacakge name	Package name	Relationship weight
3/usr.bin/	3/include/	82
3/libexec/	3/usr.bin/	4
3/lib/	3/usr.sbin/	2
3/sbin/	3/sys/	302

(a)

Package name	Package name	Relationship weight
3/contrib/	3/games/	142
3/usr.bin/	3/include/	365
3/usr.sbin/	3/tools/	149
3/sbin/	3/sys/	806

(b)

Pacakge name	Package name	Weight (Implicit architecture)	Weight (Explicit architecture)
3/usr.bin/	3/include/	365	82
3/sbin/	3/sys/	806	302

(c)

Figure 10. (a) Explicit Architecture (b) Implicit Architecture (c) Congruence

chitectures or networks, then is divided by the size of the reference architecture and expressed in a scale of 100.

The application of Equation (1) to specific cases is presented next.

Comparing the Architecture: To measure and compare the architectural concurring among the three projects, we performed a stable release wise comparison of the explicit architectures for each pair of forked projects. Thus in this case, both $Ref_{A/N}$ and $Analogous_{A/N}$ represent two comparable explicit architectures taken from two projects. To be comparable, the stable releases

of two projects should be released around the same time period. For instance, consider the stable releases of FreeBSD and NetBSD projects. FreeBSD has 10 stable releases whereas NetBSD has 14 (Fig. 5, column 2). Thus to compare two releases, each taken from the two projects, we determined the release date-wise correspondence. Therefore, FreeBSD release 6 and NetBSD release 3.0 have a correspondence as they were released in November, 2005 and December, 2005, respectively.

The intersection operation in numerator of equation (1) is calculated at three abstraction levels of the explicit architectures, namely, package level (p), first directory level (d_1) and code file directory level (d_n). For package level, the intersection operation results in the number of packages that are common (by comparing the names of the packages) between two releases. On the other hand, for the directory level, e.g., d_1 and d_n, the intersection operation provides the total number of directories that have the complete match in their directory paths. As an illustrative example, consider FreeBSD release 6 and NetBSD release 3.0 which have 19 and 22 packages, respectively. Thus $|FreeBSD - release - 6|$ = 19 and $|NetBSD - release - 3.0|$ = 22. The intersection operation between these two explicit architectures resulted in 16 packages having the same names.

Finally, the concurring value was calculated taking each of these architectures as a reference architecture. This value depicts the extent to which each of these stable releases coincide with the other. In continuation to the above example, FreeBSD release 6 has 84.21% (16/19*100) and NetBSD release 3.0 has 72.72% (16/22*100) concurring with each other. These values were then plotted in a trend chart to visualize how such concurring evolves with the projects. An example of this process is presented in Fig.11 and discussed in Section V-A.

Comparing the Community: To compare the communities among the three forked projects using the similarity measure in Equation (1), we carried out the following: first, the release wise developer list was generated for each project. This step was discussed in section IV-C. Second, for a given pair of releases, the union operation in the numerator identifies the number of contributors in both releases whose names are lexically identical. Finally, for each of the stable releases, concurring value was calculated considering each as a reference network. These values were then plotted in a trend chart. An example of this process is presented in Fig. 14 and discussed in Section V-B.

Socio-technical Congruence: To measure socio-technical congruence using the similarity measure in (1) the following approach was applied: the intersection operation in numerator was carried out between (a) Explicit Architecture and Implicit Architecture, and between (b) Explicit Coordination Network and Implicit Coordination Network. This operation identifies the number of edges (or relationships) that are identical for both the architectures or the networks.

The former measure (in (a)) illustrates the match between the architectural dependency and the architecture produced due to the communication structure of the community. The latter measure (in (b)) in turn depicts the match between the actual coordination activities in the community and the coordination need established by the architectural dependency of the software. These measures verify Conway's Law and the reverse Conway's Law, respectively. Both the measures were determined for each stable release for all three projects. A partial snapshot of the congruence between Explicit and Implicit Architectures of FreeBSD stable release 3 is shown in Fig. 10.

Then to identify the extent to which the implicit architecture and implicit network approximate the corresponding explicit one, we calculated the similarity measure in (1), taking each of the explicit architecture and network as the reference one. The resulting values were plotted in a trend chart for each project to conceptualize their evolution pattern. An example of this analysis is presented in Fig. 17 and discussed in Section V-C.

E. Implementation and Verification

Tools Used In the Study: A number of existing tools and OSS packages were used in this work. For instance, we used the tool *Understand (version: 3.1.659)* [32] to generate the Explicit Architectures. To read/write excel files Apache POI [35] was used. Also, Jsoup HTML parser [31] was used to parse the HTML files.

Implementation and Verification of the Developed Programs: We implemented several data extraction, cleaning, and analysis programs in Java for this work. Data extraction programs were used to extract data from relevant sources and cleaning programs were used for removing the anomalies in the collected data. To verify the correctness of these programs, a two pass evaluation were conducted. First, the programs were tested with a limited number of data samples taken from each of the projects. Notified bugs (e.g., errors in the parsed data for an HTML tag) were fixed accordingly. Second, a manual checking on a random sample of the actual collected data was done. The accuracy of collected data in the second pass was reported to be over 97%.

Additionally, analysis programs were written for generating the architectures, communication networks, release-wise comparisons, and for measuring congruence. These programs in turn were tested following a similar method as stated above.

V. Result Analysis

The target of this study is three-fold. First, we verify the extent to which the forked projects collaborate in

both technical and social domain. Second, we measure the socio-technical congruence in each project to conceptualize the socio-technical dependencies. Finally, we study the projects' pattern of evolution during their maturation.

A. Pattern of Architecture Evolution

In this section we present the results of the evolution of the architectural design for each forked project in relation to the other projects. In verifying this, pairwise comparison of the architectural designs (each taken form the compared projects) were made at three abstraction levels. This action was performed according to the procedure presented in Section IV-D. The result of this comparison is presented in Figs. 11, 12 and 13, one for each pair of projects. These figures show the concurring of architectures (plotted in the Y-axis) for each comparable stable release pair (plotted in the X-axis) of the projects.

Overall architectural evolution revealed similar patterns for all three types of forking relationships, e.g., sibling projects, parent-child projects, and lineages. At higher abstraction level (e.g., package level) the architectures of the forked projects maintain high correspondence between them, which remains consistent as the projects evolve. However, at the detailed architectural level (e.g., at directory levels d_1 and d_n), the design and implementation became more disjoint and independent.

For instance, in Fig. 11, the package level concurring between the architectures of FreeBSD and NetBSD projects remain high throughout their release history. For FreeBSD it remains between 61,9% and 84,21%, whereas for NetBSD it is between 57,69% and 80% with slight drifts between the ranges. Contrary to this, directory level overlapping (d_1 and d_n) point out a different trend. In both of these cases, a consistent decrease in concurring can be noticed. For example, for NetBSD and FreeBSD the overlapping at d_1 directory level begins with 82,81% and 56,1% respectively, which gradually decreases to 37,39% and 41.72% respectively. Likewise, at d_n level, the overlapping goes down to 3,63% and 3,34% from 29,77% and 10,82% respectively.

For the other two cases (Fig. 12 and 13), a similar trend was noticed with minor distinction during the early stages of the projects. For instance, in Fig. 13 the overlapping of all three architectural level starts with a very low ratio, which however had a sharp rise in the next release. For the subsequent releases, the pattern remains similar to the observations stated earlier.

Additionally, at any given point of the comparison, the adherence to common architectural design falls off significantly from abstract to detail level of the design. For instance, in 2012, the FreeBSD package level overlapping is 75%, which is however around 41,72% and 3,34% for directory level overlapping d_1 and d_n, respectively. This observation holds for all the three projects.

These observations indicate that the BSD forked projects preserve a common structure at higher level of design, which are however, get liberated progressively at the detailed architectural design. However, thorough analyses of architectural design need to be conducted to fully affirm this claim.

B. Pattern of Community Evolution

Forking of a project causes a split in the community. The fragmentation of the community is typically followed by a rebuild and restructuring phases in both projects (the original and the fork). However, both projects share the same source of code-base, which could stimulate the development communities of the two projects to contribute to both. This observation lead us to investigate the extent to which the community members (from each project) contribute during the evolution of both projects.

The investigation was done according to the process defined in Section IV-D. The results are presented in Fig. 14, 15, and 16, one for each pair of projects. The findings reveal that the level of participation of the community members in the compared projects remains consistent within a given range. Also, a similar pattern of participation is noticed for the three types of forking, confirming the earlier observation in Section V-A.

Relating these observations to individual cases show that for the FreeBSD and NetBSD projects (Fig. 14), the community overlapping remains between 23,49% and 44,9%, whereas for NetBSD it is between 26,47% and 44,23%. Within this range of participation there exist several drifts. For instance, in 1999 and 2007 (Fig. 14), a decrease in participation can be observed.

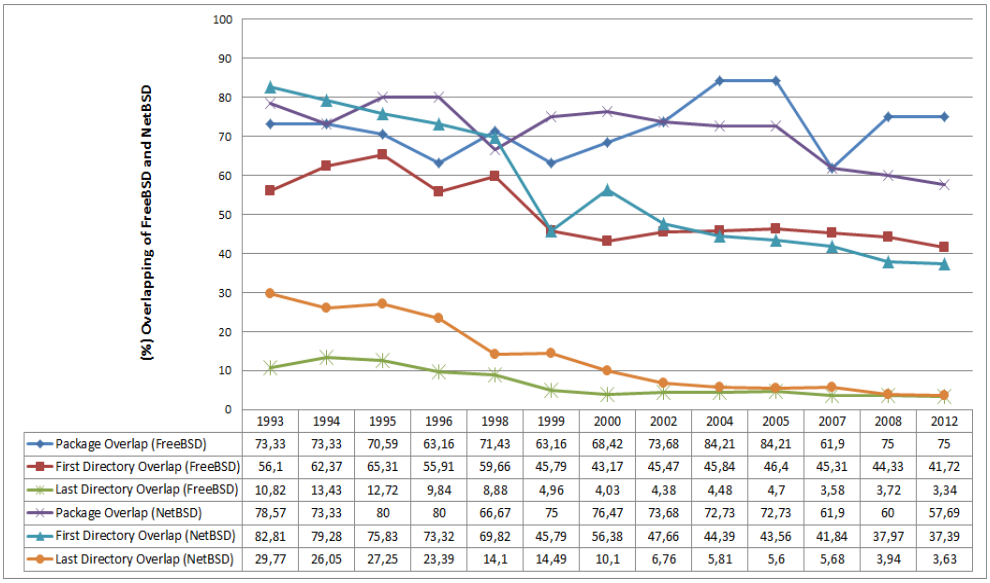
For the other two cases (Fig. 15, and 16), the pattern of overlapping follows a similar trend, except for the first two releases. This observation is similar to that discussed in Section V-A. For instance, the level of contribution rises sharply after having a low participation at the early release. Apart from this, the participation level (in Fig. 15) for NetBSD remains between 42,69% and 50,3%, and for OpenBSD between 34,42% and 38,31%. Similarly, for FreeBSD and OpenBSD (Fig. 16) it is 30,58%-35,05% and 27,18%-30,38%, respectively.

These results lead to the point that a certain group of community members maintain contributions to all the projects. The number of participation also remains stable throughout the evolution.

C. Evolution pattern of Socio-technical Congruence

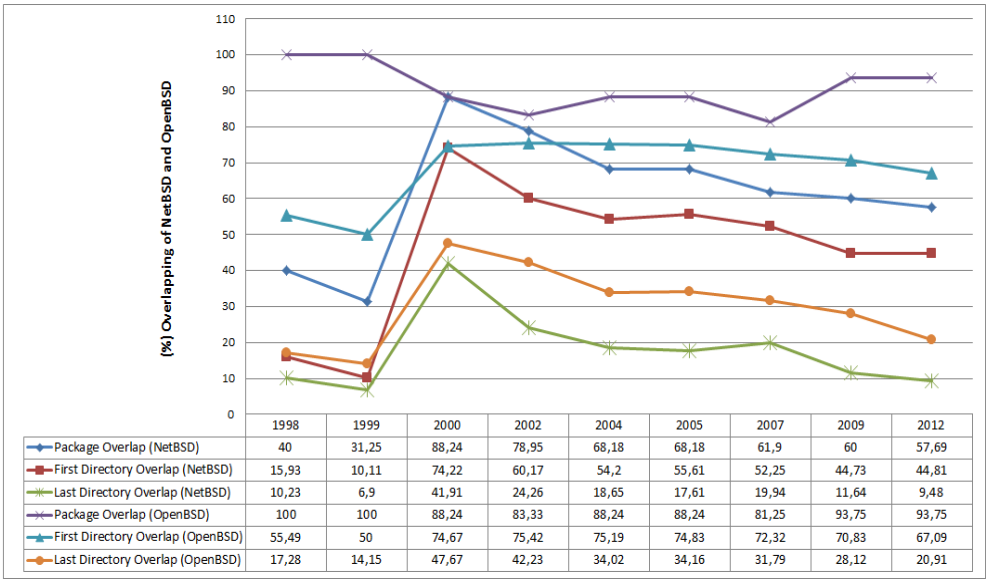
The measurement of Socio-technical Congruence for a project is a two step process. First, the extent to which the communication patterns of the members of the developer community resemble the actual architectural dependencies is verified. And then, the resemblance of the architecture to the community communication is investigated. In doing so, we derived both the implicit and explicit architectures and community collaboration networks, and measured the corresponding congruence. This process was discussed in detail in Section IV-D.

The evolution of congruence at architectural level for the three projects is shown in a trend chart in Fig. 17. In this figure, the congruence approximation is plotted in the



	1993	1994	1995	1996	1998	1999	2000	2002	2004	2005	2007	2008	2012
Package Overlap (FreeBSD)	73,33	73,33	70,59	63,16	71,43	63,16	68,42	73,68	84,21	84,21	61,9	75	75
First Directory Overlap (FreeBSD)	56,1	62,37	65,31	55,91	59,66	45,79	43,17	45,47	45,84	46,4	45,31	44,33	41,72
Last Directory Overlap (FreeBSD)	10,82	13,43	12,72	9,84	8,88	4,96	4,03	4,38	4,48	4,7	3,58	3,72	3,34
Package Overlap (NetBSD)	78,57	73,33	80	80	66,67	75	76,47	73,68	72,73	72,73	61,9	60	57,69
First Directory Overlap (NetBSD)	82,81	79,28	75,83	73,32	69,82	45,79	56,38	47,66	44,39	43,56	41,84	37,97	37,39
Last Directory Overlap (NetBSD)	29,77	26,05	27,25	23,39	14,1	14,49	10,1	6,76	5,81	5,6	5,68	3,94	3,63

Figure 11. Architectural evolution between the sibling forked projects (FreeBSD and NetBSD)



	1998	1999	2000	2002	2004	2005	2007	2009	2012
Package Overlap (NetBSD)	40	31,25	88,24	78,95	68,18	68,18	61,9	60	57,69
First Directory Overlap (NetBSD)	15,93	10,11	74,22	60,17	54,2	55,61	52,25	44,73	44,81
Last Directory Overlap (NetBSD)	10,23	6,9	41,91	24,26	18,65	17,61	19,94	11,64	9,48
Package Overlap (OpenBSD)	100	100	88,24	83,33	88,24	88,24	81,25	93,75	93,75
First Directory Overlap (OpenBSD)	55,49	50	74,67	75,42	75,19	74,83	72,32	70,83	67,09
Last Directory Overlap (OpenBSD)	17,28	14,15	47,67	42,23	34,02	34,16	31,79	28,12	20,91

Figure 12. Architectural evolution between parent-child forked projects (NetBSD and OpenBSD)

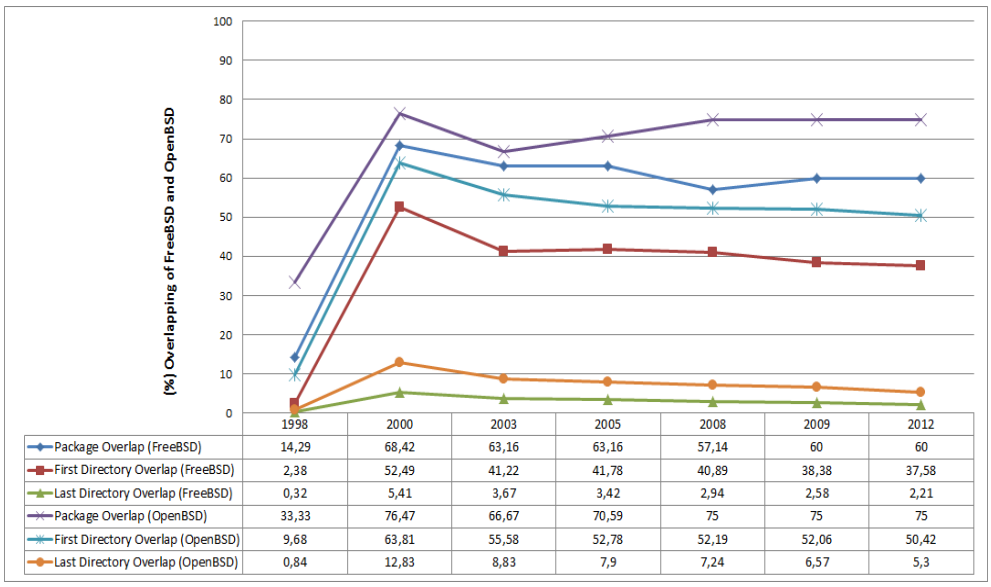
Y-axis (in percentile value) against each stable release of the projects (plotted in the X-axis).

For FreeBSD (the blue line in Fig. 17), the approximation of the congruence consistently has risen starting from 60,5% at the first stable release and has gone up to 89,4%. It had a sharp rise during the early five releases and got stabilized for the later six releases. During this period the congruence level remained between 84,83% and 89,4%. We considered the first four congruence values as outliers as a project usually goes under considerable restructuring and reformation after it is being forked.

For OpenBSD (the green line in Fig. 17) we observed a similar trend of congruence to that of FreeBSD. For the initial two releases the approximation of congruence were around 75%, that increased sharply to 88,38% on the third stable release. Till then onwards it remained stable within the range 85,56% and 88,78%.

In contrast to these two projects, NetBSD (the maroon line in Fig. 17) had a different pattern. In NetBSD the congruence approximation started with 85% and remained stable around 80,77% to 87,5% for the first twelve releases. Nevertheless, for the recent releases (e.g., the last two stable releases), the project experienced a decrease in congruence which has gone bellow 80%.

Accumulation of these results portrays that the approximation of the Explicit Architecture by the congruence is considerably high in all these three projects, which remains stable throughout the evolution. This implies that the architecture derived from the communication pattern of the developer community effectively represents the actual architecture of the software. That is, to a considerable extent the communication of the contributing developers



	1998	2000	2003	2005	2008	2009	2012
Package Overlap (FreeBSD)	14,29	68,42	63,16	63,16	57,14	60	60
First Directory Overlap (FreeBSD)	2,38	52,49	41,22	41,78	40,89	38,38	37,58
Last Directory Overlap (FreeBSD)	0,32	5,41	3,67	3,42	2,94	2,58	2,21
Package Overlap (OpenBSD)	33,33	76,47	66,67	70,59	75	75	75
First Directory Overlap (OpenBSD)	9,68	63,81	55,58	52,78	52,19	52,06	50,42
Last Directory Overlap (OpenBSD)	0,84	12,83	8,83	7,9	7,24	6,57	5,3

Figure 13. Architectural evolution pattern between lineage forked projects (FreeBSD and openBSD)

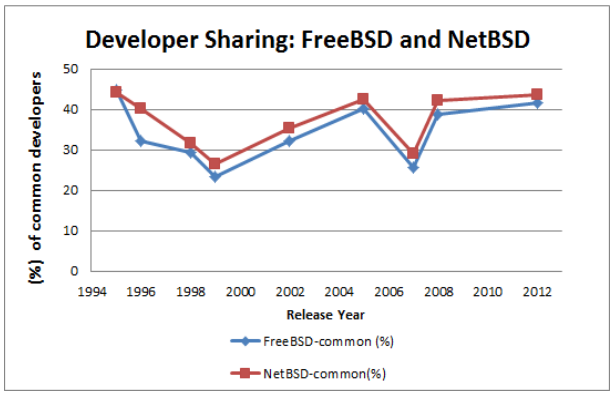


Figure 14. Community concurring pattern between FreeBSD and NetBSD projects

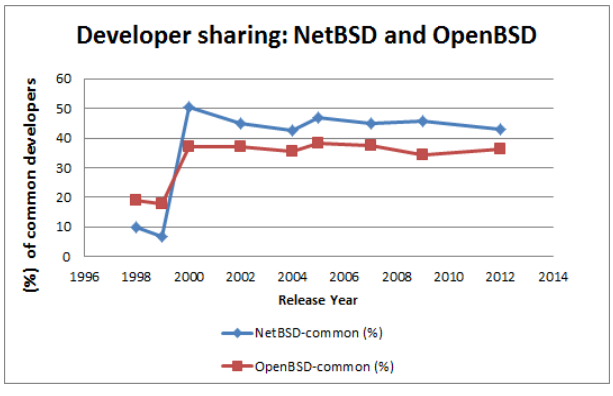


Figure 15. Community concurring pattern between NetBSD and OpenBSD projects

in the community may actually be due to the coordination needs as identified by the architectural dependencies.

On the other hand, the approximation level of the congruence to that of the Explicit Coordination Network reveals a similar pattern for the three projects. Fig. 18

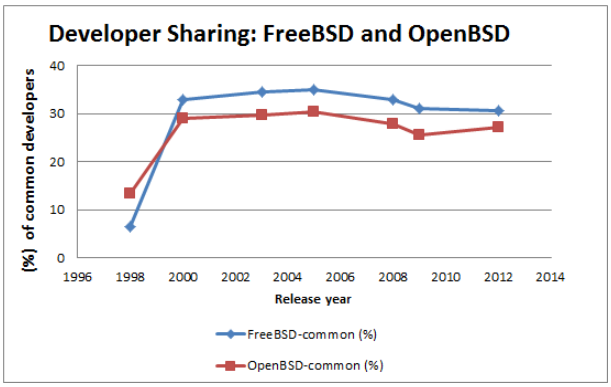


Figure 16. Community concurring pattern between FreeBSD and OpenBSD projects

shows the evolution of approximation against each stable release of the projects.

For FreeBSD (the blue line in Fig. 18), the approximation of the congruence remained between 70,63% and 87,31% from the fourth stable release onwards. A few drifts in congruence in the early three releases were noticed, which can be justified with the same reasoning as before. Yet, there was a decreasing trend of congruence noticed for the last two stable releases.

In the case of OpenBSD (the green line in Fig. 18), the approximation of the congruence to that of Explicit Coordination Network started with 80%, and remained stable between the value 73,35% and 87,77% during the entire evolution of the project. Only for the last release the congruence value went down to 39,58%, which is mainly due to missing data.

For NetBSD (the maroon line in Fig. 18) the congruence approximation started with a high value of 98,87% and remained stable between 8139% and 98,87% as the project progressed. Only for the tenth release (May 2005 in the chart) the congruence has gone as bellow as

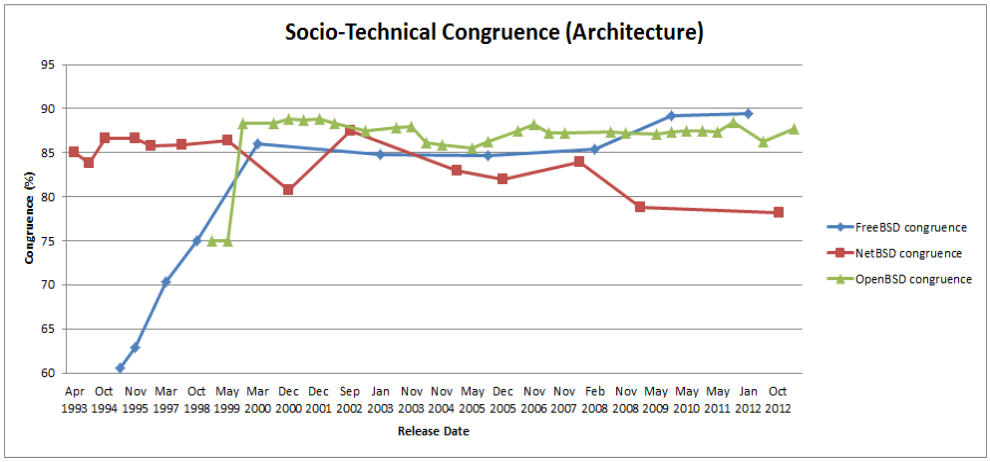


Figure 17. Evolution of Congruence at Architectural Level of the BSD Projects

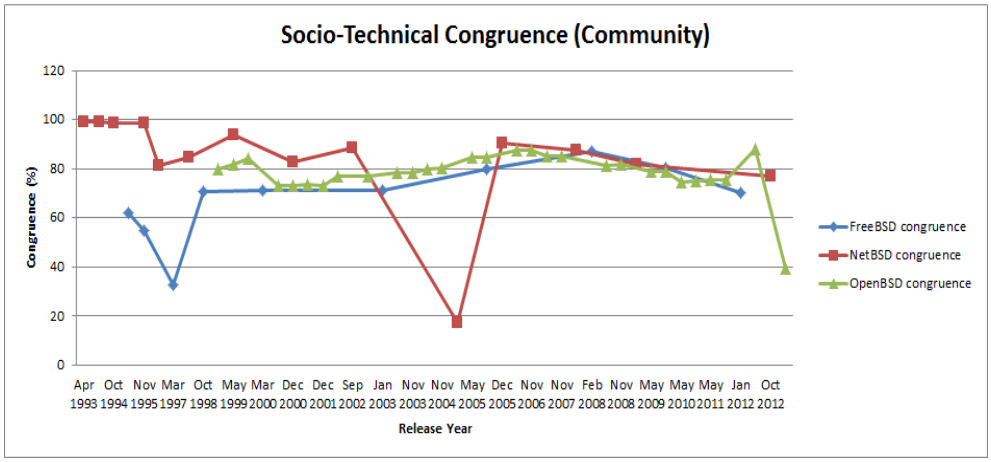


Figure 18. Evolution of Congruence at Community Level of the BSD Projects

17,23%. But it can be treated as an outlier due to missing data. Yet there was a slight decrease noticed for the last three stable releases.

To summarize these results, it can be conceived that the congruence approximation to that of Explicit Coordination Network is considerably high for the three projects. That is, the communication pattern of the developer community derived from the architectural dependency of the components effectively resembles the actual communication pattern. Thus, the communication pattern of contributing developer community can be used to simulate the underlying architectural dependency of the software to a great extent.

VI. Discussion

In this section we hereby summarize the findings of this study and possible implications in relation to prior works.

A. Research Questions Revisited

The evidence presented provides a strong indication that each forked project in the BSD family enjoys a high level of Socio-technical congruence throughout their evolution history. Thus, it can be affirmed that to a considerable extent the communication of the contributing developers in the BSD communities might be due to the coordination needs as identified by the technical dependency, and vice-versa. This observation is in-line with the prior work that reported congruence as a desired property and a natural phenomenon of collaborative development works [16] [36].

Alongside these observations, communities of the forked BSD projects have maintained a certain level of collaboration throughout the project history. Our reported model of collaboration shows that a portion of the community is mutual for both the projects. In literature, this group of community members are termed as the bridge between the projects [37], and a means of information flow and collaboration [37] [38].

Moreover, the architectural design at higher abstraction level has remained homogeneous among the forked projects. This might have supported the developer community with better understanding of the overall system designs and have created a common ground for collaboration and contribution. However contrary to this, at detail architectural level these projects are progressively getting liberated. This could be explained by the fact that the

developer community of each fork has adopted their own implementation strategies when it comes to fine grained design decisions.

Finally, it was noticed that the pattern of community and architectural evolution for all the three forking relationships (e.g., siblings, parent-child and lineages) have followed similar patterns. This observation highlights the point that forked projects that have originated from the same root project would ideally share a common architectural design and a healthy inter-project collaboration.

B. Implications

It can be argued here that Socio-technical congruence plays a pivotal role in forming cohesive and organized community driven projects, which eventually leads to their successful evolution with high quality. This argument is also affirmed in earlier literature conducted on in-house projects: Higher congruence influences project success [3] [5] [16], with improved productivity [39] [6], maintainability [40], and quality [7].

This measure of socio-technical congruence would better serve the purpose of software development process and organization. Because it provides a quick index of how well the organization is actually aligned with the current and planned sub-division of responsibility in the project [41]. Additionally, the Implicit Architecture can be used as a complementary to the traditional reverse engineering process [42] [43] to derive and validate the recovery of the Explicit Architecture of legacy systems.

The identified pattern of collaboration among the three projects could be one way to explain the sustainability of the forked projects [44], particularly during their early formation stages. Additionally, further study could be initiated to verify the impact of such collaboration on cross project porting and code cloning [45] [46].

Overall, based on our study results, we claim that the traditional perception of forking in OSS projects, which is thought to have negative stimuli for sustainable evolution of the projects [8], can be effectively remedied though (a) maintaining a consistent and cohesive abstract architectural design to form a common ground of collaboration among the forked projects, (b) adopt a collaboration model in which members of a project could participate in other forks, and (c) maintain a consistent and high socio-technical congruence within the project.

None-the-less, this study puts a step forward in reasoning about the successful evolution of forked OSS projects, as this perspective has rarely been studied in current literature on OSS evolution analysis [8] [47].

VII. On the Missing data and Replication of the Study

Data collection process for this study sufferers from some missing data. The missing data constitutes the general communication emails stored in the email archives. Missing email conversations are encountered for NetBSD and OpenBSD projects. To be specific, email conversations during the period of April, 2005 to May, 2005 can not be extracted fully for NetBSD project. Whereas for OpenBSD project, missing emails are noticed during the period of September and October, 2012. In case of NetBSD it is mainly due to broken links to the archives, and for OpenBSD it is probably due to unavailability of the data during that time period.

However, the volume of such missing data is not massive, and thus, have little impact on the overall results. Only at the two points of congruence measure (as discussed in Section V-C), such missing data injected drifts, which however, do not hamper the overall trend of the congruence.

Replication of the study depends on addressing several issues, which includes, (a) data collection from the relevant sources, (b) cleaning and representation of the data and finally, (c) carrying out the analysis. In what follows, a guideline to accomplish these tasks.

Data is collected from two sources, SVN/CVS repositories and email archives. A detail discussion on downloading and extracting data from these sources are presented in Sections IV-B and IV-C. However, to ease this process of data collection for interested researchers, we make available the extracted data in the link given bellow[1]. Further instructions on how to interpret and use the data in replicating this study is discussed in the given link.

Finally, generating the architectures and networks, and carrying out the congruence measure are done thorough the implementation of scripts. There scripts are directly derived from the definitions and analysis methods discussed in Sections II and IV-D, respectively. Tools and packages listed in Section IV-E are used for script implementation. All the packages are open source and are available online for free downloading. However, the scripts used in this study are not made available in the given link. If researchers require assistance in implementing the scripts, we could provide adequate guidelines and the scripts upon request[2].

VIII. Threats to Validity

The following aspects have been identified which could lead to threats to validity of this study.

External validity (how results can be generalized): As case study subject, projects from the BSD family were selected, which are FreeBSD, NetBSD and OpenBSD. All these projects belong to the operating system domain, have large developer and user communities, and have over twenty years of evolution history. Additionally, OSS evolution studies often used these projects as case study. Thus it might be possible to stress the results reported in this article to the population of OSS projects having similar properties, e.g., domain, project size, evolution history. Yet, we cannot claim complete external validity of the results.

[1] http://msyeed.weebly.com/replication-package.html

[2] Contact: $rajit.cit@gmail.com$

Internal validity (confounding factors can influence the findings): Missing historical data - the study has been able to make use only of available data. It is possible, for instance, that there are commit records and developer chat entries other than that recorded in the emails. Additionally, we encountered several broken URL links for emails that could not be retrieved. Thus, we make no claim on the completeness of the email entries with relevance to this study target.

Construct validity (relationship between theory and observation): There exist a few issues that concern the construct validity of the study. First, part of the email entries were categorized to a specific stable release according to their date of post. The reasoning here is that developers commit and discuss on release planning before the product is officially released. Yet, we do not claim the perfection of this approach. Second, the data extraction programs written for this study provided an accuracy of 97%, which was measured with random sample of the collected data. This may affect the construct validity.

IX. Conclusions

The current study provides empirical evidence that successful OSS forked projects that are lineages of an ancestor project may follow similar evolution patterns in terms of (a) technical and social dependencies and (b) achieving a high level of congruence that sustains throughout their evolution. Though from a technical perspective the forked projects get more and more independent by time, they may enjoy a sustainable level of cross project collaboration. Keeping in line with prior evidence [9], we can argue that congruence is an implicit characteristic of successful forked OSS projects, and combining it with inter project collaboration would portray the reason behind the success of such projects. This claim however needs further empirical evidence. As an alternative to the qualitative argumentation approach taken in our study, one could frame our research questions as hypotheses and perform statistical analysis to evaluate them. This constitutes our future work.

References

[1] A. Mockus, R. Fielding, and J. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *Journal of TOSEM*, vol. 11, no. 3, pp. 309–346, 2002.

[2] G. Valetto, S. Chulani, and C. Williams, "Balancing the value and risk of socio-technical congruence," *Workshop on Sociotechnical Congruence*, 2008.

[3] I. Kwan, A. Schrter, and D. Damian, "Does socio-technical congruence have an effect on software build success? a study of coordination in a software project," in *IEEE Trans. Software Eng.*, vol. 37, no. 3, 2011, pp. 307–324.

[4] M. E. Conway, "How do committees invent?" *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[5] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley, "Identification of coordination requirements: Implications for the design of collaboration and awareness tools," in *ACM CSCW*, 2006, pp. 353–362.

[6] L. Colfer and C. Baldwin, "The mirroring hypothesis: Theory, evidence and exceptions," in *working paper, Harvard Business School*, 2010.

[7] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: an empirical case study," in *ICSE '08 Proceedings of the 30th international conference on Software engineering*, 2008, pp. 521–530.

[8] G. Robles and J. Gonzalez-Barahona, "A comprehensive study of software forks: Dates, reasons and outcomes," in *OSS, IFIP AICT 378*, 2012, pp. 1–14.

[9] M. Syeed and I. Hammouda, "Socio-technical congruence in oss projects: Exploring conways law in freebsd oss evolution," in *Proceedings of 9th International Conference of Open Source Systems (OSS), Springer*, 2013.

[10] M. Fischer, J. Oberleitner, J. Ratzinger, and H. Gall, "Mining evolution data of a product family," *ACM SIGSOFT Software Engineering Notes*, vol. 4, no. 30, pp. 1–5, 2005.

[11] J. Niels, "Putting it all in the trunk: incremental software development in the freebsd open source project," *Information Systems Journal*, vol. 11, no. 4, pp. 321–336, 2001.

[12] T. Dinh-Trong and J. Bieman, "The freebsd project: A replication case study of open source development," *Software Engineering, IEEE Transactions on*, vol. 31, no. 6, pp. 481–494, 2005.

[13] J. Han, C. wu, and B. Lee, "Extracting development organization from open source software," in *16th Asia-Pacific Software Engineering Conference, IEEE.*, 2009, pp. 441–448.

[14] E. S. Raymond, "The new hacker's dictionary (3rd ed.)," in *Cambridge, MA, USA: MIT Press*, 1996.

[15] L. M. Nyman and T. Mikkonen, "To fork or not to fork: Fork motivations in sourceforge projects," in *Source Systems: Grounding Research : IFIP Advances in Information and Communication Technology*, 2011, pp. 259–268.

[16] T. Browning, "Applying the design structure matrix to system decomposition and integration problems: a review and new directions," in *Engineering Management, IEEE Transactions on*, vol. 48, no. 3, 2001, pp. 292–306.

[17] M. E. Sosa, S. D. Eppinger, and C. M. Rowles, "The misalignment of product architecture and organizational structure in complex product development," in *Management Science*, vol. 50, no. 12, 2004, pp. 1674–1689.

[18] I. Herraiz, J. Gonzalez-Barahona, G. Robles, and D. German, "On the prediction of the evolution of libre software projects," in *ICSM*, oct. 2007, pp. 405 –414.

[19] FreeBSD, "http://www.freebsd.org/," 2013.

[20] NetBSD, "http://www.netbsd.org/about/," 2013.

[21] OpenBSD, "http://www.openbsd.org/," 2013.

[22] J. Wu, R. Holt, and A. Hassan, "Empirical evidence for soc dynamics in software evolution," in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, oct. 2007, pp. 244 –254.

[23] I. Herraiz, "A statistical examination of the evolution and properties of libre software," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, sept. 2009, pp. 439 –442.

[24] J. C. JE, L. V. LG, and A. Wolf, "Cost-effective analysis of in-place software processes," in *IEEE Transactions on Software Engineering*, vol. 24, no. 8, 1998, pp. 650–663.

[25] D. Atkins, T. Ball, T. Graves, and A. Mockus, "Using version control data to evaluate the impact of software tools," in *Proceedings 21st International Conference on Software Engineering*, vol. 24, no. 8, 1999, pp. 324–333.

[26] I. Kwan, M. Cataldo, and D. Damian, "Conway's law revisited: The evidence for a task-based perspective," *IEEE Software*, vol. 29, no. 1, pp. 90–93, 2012.

[27] M. Goeminne and T. Mens, "A framework for analysing and visualising open source software ecosystems," in *Proceeding IWPSE-EVOL '10*, 2010, pp. 42–47.

[28] D. M. German, "Using software trails to reconstruct the evolution of software," in *JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE*, vol. 16, 2004, pp. 367–384.
[29] Y. Wang, D. Guo, and H. Shi, "Measuring the evolution of open source software systems with their communities," in *ACM SIGSOFT Software Engineering Notes*, vol. 32, no. 6, 2007.
[30] W. Zhang, Y. Yang, and Q. Wang, "Network analysis of oss evolution: An empirical study on argouml project," in *IWPSE-EVOL11*, 2011.
[31] jsoup: Java HTML Parser, "http://jsoup.org/," 2013.
[32] U. S. C. Analysis and Metrics, "http://www.scitools.com/," 2013.
[33] D. Darcy, S. Daniel, and K. Stewart, "Exploring complexity in open source software: Evolutionary patterns, antecedents, and outcomes," in *Proceedings of the 43rd Hawaii International Conference on System Sciences*, 2010, pp. 1–11.
[34] M. Simmons, P. Vercellone-Smith, and P. Laplante, "Understanding open source software through software archaeology: The case of nethack," in *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*, 2006, pp. 47–58.
[35] A. P.-J. A. for Microsoft Documents, "http://poi.apache.org/," 2013.
[36] J. Herbsleb and R. Grinter, "Architectures, coordination, and distance: Conway's law and beyond," in *Journal IEEE Software*, vol. 16, no. 5, 1999, pp. 63–70.
[37] M. Weiss, G. Moroiu, and P. Zhao, "Evolution of open source communities," in *IFIP International Federation for Information Processing, Volume 203, Open Source Systems*, 2006, pp. 21–32.
[38] J. Gonzalez-Barahona, L. Lopez, and G. Robles, "Community structure of modules in the apache project," in *Workshop on Open Source Software Engineering*, 2004.
[39] C. Baldwin and K. Clark, "Design rules: The power of modularity," in *MIT Press*, 2000.
[40] F. P. Brooks, "The mythical man-month," in *Anniversary Edition: Addison-Wesley Publishing Company*, 1995.
[41] G. Valetto, M. Helander, K. Ehrlich, S. Chulani, M. Wegman, and C. Williams, "Using software repositories to investigate socio-technical congruence in development projects," in *ICSE Workshops MSR*, 2007, pp. 25–25.
[42] H. Dayani-Fard, Y. Yu, J. Mylopoulos, and A. Periklis, "Improving the build architecture of legacy c/c++ software systems," in *8th FASE*, 2005.
[43] R. Kazman and S. Carrire, "Playing detective: Reconstructing software architecture from available evidence," in *Technical Report CMU/SEI-97-TR-010, Carnegie Mellon University*, 1997.
[44] J. Gamalielsson and B. Lundell, "Sustainability of open source software communities beyond a fork: How and why has the libreoffice project evolved?" *Journal of Systems and Software*, vol. 89, pp. 128–145, 2014.
[45] B. Ray and M. Kim, "A case study of cross-system porting in forked projects," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 53.
[46] D. German, M. D. Penta, Y.-G. G. éhéneuc, and G. Antoniol, "Code siblings: Technical and legal implications of copying code between applications," in *MSR'09*. IEEE, 2009, pp. 81–90.
[47] M. Syeed, I. Hammouda, and T. Systa, "The evolution of open source software projects: a systematic literature review," *Journal of Software*, vol. 8, no. 11, pp. 2815–2829, 2013.

M.M. Mahbubul Syeed received his B.Sc degree in Computer Science and Information Technology from Islamic University of Technology, Bangladesh in September, 2002 and his M.Sc degree in Information Technology from Tampere University of Technology, Finland in April, 2010. He is currently working towards his Ph.D. degree and working as a researcher in the same university. His current research interest includes study of Open Source Software ecosystem, ecosystem enabling architecture, project evolution, experimental software development, and big data mining and knowledge extraction.

Dr. Imed Hammouda joined University of Gothenburg in September 2013. Before that, he was Associate Professor of software engineering at Tampere University of Technology (TUT), Finland. At TUT, he was heading the international masters programme at the Department of Pervasive Computing. He got his Ph.D. in software engineering from TUT in 2005. Dr. Hammouda's research interests include open source software, software architecture, software development methods and tools, and variability management. He was a founding member and leader of TUTOpen - TUT research group on open source software. He has been the principal investigator of several research projects on various open initiatives. Dr. Hammouda's publication record includes over fifty journal and conference papers.