Socio-technical Congruence in OSS Projects: Exploring Conway's Law in FreeBSD

M.M. Mahbubul Syeed and Imed Hammouda

Tampere University of Technology, Finland
{mm.syeed,imed.hammouda}@tut.fi

Abstract. 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 projects where development is often more dispersed and distributed. In this paper, we study the match between the coordination needs established by the technical domain (i.e. source code) and the actual coordination activities carried out by the development team, such hypothetical match is also known as socio-technical congruence. We carry out our study by empirically examining Conway's law in FreeBSD project. Our study shows that the congruence measure is significantly high in FreeBSD and that the congruence value remains stable as the project matured.

1 Introduction

Investigating socio-technical congruence in software development projects has become an active research area in the last decade [10]. 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 development team) [6]. 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 [6]. Socio-technical congruence not only has been used as a measure for a number of project properties such as software build success [8] but also as a means for software engineering tasks like architecture recovery [23].

In most of research on socio-technical congruence, Conway's law [4] is often presented as a guide and a basis for the underlying study. Conway's Law in its purest form states that "the organizations that design systems are constrained to produce systems which are copies of the communication structures of these organizations" [1]. In other words, the product architecture often reflects the organizational structure of the development team [1][6]. In [7], Conway's law is considered homomorphic and thus claimed to be true in reverse as well. This means the communication pattern within developer community should reflect the architectural dependency in the software. Thus, the notion of socio-technical congruence is actually a conceptualization of Conway's law.

E. Petrinja et al. (Eds.): OSS 2013, IFIP AICT 404, pp. 109–126, 2013.

Studying socio-technical congruence in open source software has its own questions and hypotheses, for those products are peer-produced, crowdsourced systems where centralized planning and control of architecture is difficult. Furthermore, the evolution of such systems, and their underlying architecture, is not bound to any pre-defined plans [5]. Surprisingly, this research area has not been given much attention among open source researchers [34].

In this paper, we study Conway's Law and Reverse Conway's Law, as a lens for socio-technical congruence, in the context of open source development projects by proposing a novel evaluation and measurement technique. We further explore the significance of socio-technical congruence as the project matures. To investigate our research problem, we focused on a popular open source project, FreeBSD, which is an advanced operating systems for computing platforms. We carried out our study empirically by analyzing the software repository of the project in a semi-automatic way.

The remaining of the paper is organized as follows. Section 2 introduces a number of key concepts that this study uses. Motivation and related work is presented in Section 3. We then introduce the research questions explored in this paper and our study design in Section 4 and 5 respectively. Results are reported and discussed in Section 6. Possible limitations and threats to validity are highlighted in Section 7. Finally, Section 8 concludes the papers and shed light on future research.

2 Definitions

In this paper we interpreted Conway's law (and reverse Conway's law) as a measure to verify the extent to which the communication pattern of the contributing members is due to the communication needs established by the concrete architecture and vice versa. In examining this the following concepts are defined.

2.1 Developer Contribution

Developer contribution to the software can be defined as the code contribution or any form of commit made to the code base.

2.2 Concrete Architecture

Concrete 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. In this work, header file inclusions dependency at code file level were used to derive the concrete architecture. In this architecture two files, and their corresponding packages, were linked if the two files have an inclusion dependency.

2.3 Concrete Coordination Network

Concrete Coordination Network is a social network in which two developers have a relationship if they have communication history as seen by the mailing archives representing the social and technical interactions among the developers.

2.4 Derived Architecture

Derived architecture defines an architecture of the software where any two components (e.g., packages or code files) are related if there are developer(s) who have either (a) contributed to both the components, or (b) have communication at organizational level (e.g., through email). 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 has communication at organizational level as shown in Fig. 1(a). Thus according to the definition, packages P1, P2 and P3 are linked to each others in the Derived Architecture (Fig. 1(b)).

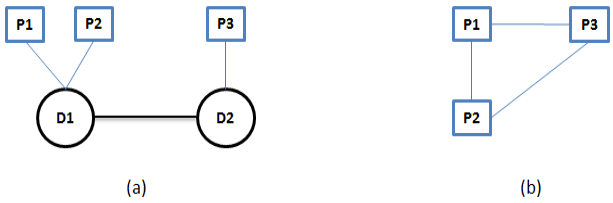


Fig. 1. (a) Concrete Coordination Network with contribution to code base (b) Corresponding Derived Architecture

2.5 Derived Coordination Network

Derived 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 relationships in the concrete architecture. For instance consider that developer D1 and D2 has contributed to package P1 and developer D3 has contributed to package P2. Also consider that P1 and P2 have an inclusion dependency as shown in Fig.2(a). Then, according to the definition developers D1, D2 and D3 will have relationships in Derived Coordination Network as shown in Fig. 2(b).

3 Motivation and Related Work

In this section we discuss the significance of Conway's law and socio-technical congruence in the field of software engineering citing related works from existing literature.

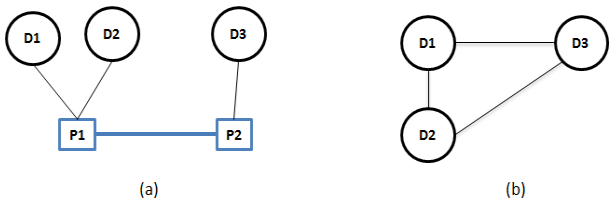


Fig. 2. (a) Concrete architecture with contributing developers (b) Corresponding Derived Coordination Network

3.1 Need for Conway's Law and Socio-technical Congruence

Conway's law was stated 30 years ago by Melvin Conway as the means to emphasize the need of coordination in software development [1]. Until then researchers have long argued that the mirroring effect of the software architecture and the communication pattern of the developer community plays a pivotal role in coordinating the development work [9]. A central question arises "Does Conway's law matter in the modern era of software development?". As reported in [10], Conway's law still matters in the domain of software development and the product quality is strongly affected by the organizational structure [11]. Also with the advent of global software engineering where development teams are distributed across the world, the impact of organizational structure on Conways law and its implications on quality is significant [12].

In the domain of software engineering, socio-technical congruence as a conceptualization of Conway's law was first coined in [13]. In this paper, socio-technical congruence was employed as a fine-grained measure of coordination that can be used to diagnose coordination problems in the development team. Also, evidence from studies [13] shows that higher congruence leads to faster completion of modification requests. This concept has been identified as an important element for product design in the field of engineering [14] and management science [15] as well. Researchers also argued that such congruence is a natural consequence and a desired property for collaborative development activities [14], such as software engineering.

3.2 Socio-technical Congruence and OSS Development

In the epoch of OSS projects, there exists a significant number of successful software systems whose volume and complexity are as compound as that of their proprietary counterparts. Such large and successful OSS projects often consist of hundreds and even thousands of developers contributing to the development of the project. Developers working in such projects are not strictly bound to any organizational rules, regulation and structure. Rather they voluntarily join and contribute to the project. OSS developers belong to discrete geographical locations of significant background, timezone, language and cultural distances. Often an OSS project is developed and evolved through the collaboration among the developers using simple communication media like email, wiki, chat [31] as

well as revision tracking systems (such as CVS, SVN) to store and access the software code they produce [19].

This unconventional organizational structure and practices of OSS projects combing with the inherent complexity of the software development (as discussed above) brings forth the question, "why such projects succeed and can socio-technical congruence be conceived as an implicit driving force for such success?". Surprisingly, socio-technical congruence as a research area has not been given much attention among open source researchers. In this paper, we infer that socio-technical congruence is an endogenous driver for successful OSS projects, and examines its existence in OSS project though empirical evidences.

4 Research Questions

Our research objective in this paper is to study the applicability of Conway's law as a means of verifying the socio-technical congruence on large, distributed and evolving OSS projects. In great part because of the specific characteristics of the OSS projects and yet having revolutionary success, it is most likely that the OSS development process implicitly encompasses the notion of socio-technical congruence. Thus, we targeted the following research questions.

(a) How does the communication patterns of OSS developer community resemble the architecture of the software?
Conway's law can be interpreted as the first explicit recognition that the communication pattern of the organization has an inevitable impact on the product [20]. Thus there must exist a correspondence between the community structure and the software structure. Our point of interest here is that if such homomorphic force between the software model and its development organization exists in OSS projects, then it could be effectively used to conceptualize the architecture of legacy systems. It can also be used as a metric to assess the quality and maintainability of the system as well.

(b) How does the architecture of the software resemble the collaboration patterns of OSS developer community?
Conway's law pointed out [1] that only the organizational arrangements can be optimized with respect to the system concept. This observation enforces the need for a stable and modular design of the system in order to facilitate effective communication, coordination among the developers. With geographically distributed development setup, as in OSS projects, co-ordination becomes more challenging due to location, time, and cultural differences, and the need for a stable design becomes even more evident [9]. For this kind of organizational setup, a clearly separable and stable architectural design would be the basis for assigning task to the developers [9] and a key to the success of the project. In other words, the collaboration pattern of the developer community should mirror the architecture of the software which should remain stable during the evolution of the project.

(c) Does the socio-technical congruence evolve as the project matures?
Our intension is to examine how the socio-technical congruence evolves as the project matures. We are particularly interested in deriving any visible pattern of such congruence during the evolution. But relating such trends in congruence in OSS projects with its success parameters are out of the scope of this study.

5 Study Design

5.1 Case and Subject Selection

To investigate our research questions, we focused on a popular open source project, FreeBSD, which is an advanced operating system for modern server, desktop, and embedded computer platforms [24]. It is derived from BSD, the version of UNIX developed at the University of California, Berkeley. Selection of this project as a case study was influenced by the facts that FreeBSD's code base has undergone continuous development, improvement, and optimization for thirty years [24]. The project has been developed and maintained by a large team of individuals. Also, FreeBSD has gained considerable attention in earlier research on the evolution of OSS projects [16] [17] [18].

5.2 Data Sources

In literature [10] a great emphasis was given to leveraging software repositories along with the communication data for deriving technical dependencies as well as developers coordination patterns. In OSS projects, repository data is stored and maintained through different data management systems, e.g., source code repository (SVN or CVS), change logs, mailing archive, bug reporting systems, and communication channels. These data sources are highly accepted and utilized medium for empirical studies on OSS projects [25][26][27]. In FreeBSD project, the development and communication history is maintained through SVN repositories and mailing archives. Following is the description of these sources and the data extracted for this study.

1. Source code repository: FreeBSD has two release branches, stable and production releases. In this study, SVN repositories of the stable releases were collected. Fig. 3 provides detail of these stable releases and the data collected from each release.
2. Mailing list archive: In OSS projects, email archives provide a useful trace of the task-oriented communication and co-ordination activities of the developers during the evolution of the project [28]. In the FreeBSD project, email archives are categorized according to their purpose. For instance, there are email for project development (e.g., commits), stable release planning, chat, user emails, bug reports. These archives contain all mailing lists since 1994 and are updated every week [24]. Fig. 4 provides detail on the FreeBSD email archives that were extracted for this study. These email archives contain the

detail of the commit records made for each stable release and are only used by the developer community. These archives thus give more accurate history of contributions than the bug reporting system which is often used by the the passive users for different purposes.

Stable Releases	Release Date	Size (# of code files)	Size (# of packages)	Data Extracted for each code file	Study Purpose
stable-2.0.5	June, 1995	5038	15	1. code or header file name. 2. File directory path. 3. File package name. 4. List of included header files. 5. Copyright information (name of the copyright holder, year, email address). 6. Number of code files, number of other files, number of packages	1. Identifying developers for each stable release. 2. Identifying developer contribution to each stable release. 3. Deriving concrete architecture, and Coordination Architecture
stable-2.1	November, 1995	6831	16		
stable-2.2	March, 1997	8424	19		
stable-3	October, 1998	6241	14		
stable-4	March, 2000	15038	19		
stable-5	January, 2003	17025	19		
stable-6	November 2005	18695	19		
stable-7	February 2008	21088	20		
stable-8	November 2009	22849	20		
stable-9	January 2012	25583	20		
Programming language used:		C and C++			
Download source:		http://svn.freebsd.org/base/stable			

Fig. 3. Stable Releases of FreeBSD

Email Collection period:		january, 1994 - january, 2012		
Download Source:		http://docs.freebsd.org/mail/		
Mail type	Year	Mail Archive	Study Purpose	Data Extracted from each email
CVS/SVN commit	1994-1998	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-	1. Identifying developers for each stable release. 2. Identifying developer contribution to each stable release. 3. Deriving Coordination Network, and Derived Coordination Network.	1. Email subject 2. Sender name 3. Sender email 4. Receiver name, email 5. Date and time Posted (i.e., year, month, day, and gmt)
	1999-2007	cvs-all		
	2008-2012	cvs-all, svn-src-stable-6, svn-src-stable-7, svn-src-stable-8, svn-src-stable-9, svn-src-stable-other		
Discussion	1994-2012	freebsd-chat, freebsd-stable	1. Identifying developers for each stable release. 2. Deriving Coordination Network, Derived Coordination Network.	

Fig. 4. FreeBSD email archives used for this study

5.3 Data Collection Procedure

Data Collection from Source Code Repositories: Each stable release of the FreeBSD project was downloaded from the SVN repository to the local directory. Fig. 3 lists the stable releases and associated detail of each release. To extract data from each of these releases, a parser was written in Java. This parser searched through each directory of a stable release, read through the files in a directory and parsed relevant data. The data that were parsed from each file were listed in Fig. 3. As FreeBSD was written in C and C++, included header files were identified from the #include directive in each code file. The parser excluded all the library header files and kept only the user defined header files. Also the copyright directive in a file contains information of the developer name,

email and the copyright year. The parser extracted each of these information from the copyright directive. The developers that were found in this process were considered as the contributors to that file for that stable release.

The parsed data for each stable release was stored in excel files. To read/write excel files Apache POI [29] was used. The number of code files and packages read from each stable release was shown in Fig. 3.

Data Collection from Email Archives: The email archives that concern CVS/SVN commits and general discussions (e.g., on stable releases and the chat entries) were extracted from FreeBSD email archive as shown in Fig. 4. 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 (link is provided in Fig. 4). Thus each email was read as an HTML page and the data was extracted using Jsoup html parser [30]. Data that was extracted from each email entry is listed in Fig. 4. This data was then stored in excel files according to the archive name and year. Then the email data was sorted according to each stable release as follows: (a) emails 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) and (b) other emails (e.g., freeBSD-stable, freeBSD-chat and most of the CVS commit emails) for which the release numbers were not mentioned, the posting dates were checked. In this case, for instance, an email was categorized to stable release 3 if its posting date fall 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.

After categorizing emails to each stable release, the subject of a CVS/SVN commit email was parsed. This subject mentioned the path to the repository to which the commit was made. From this subject, the directory path, package name, and if provided, the name of the modified code file and the stable release number were identified. Sender name for each of these CVS/SVN commit email was considered as a contributor to the code base for that release. Contributors found in this process were combined with the contributors found from the code base to get list of developers who contributed to each stable release. Fig. 5 lists release wise distribution of the number of developers and email entries identified through this process.

Releases	Stable-2.0.5	Stable-2.1	Stable-2.2	Stable-3	Stable-4	Stable-5	Stable-6	Stable-7	Stable-8	Stable-9
# of contributors / release	168	196	325	367	603	693	827	983	1081	1128
# of emails / release	7014	6343	17874	14906	22833	115536	22290	21599	15473	33882

Fig. 5. Number of contributors and emails identified for stable releases

Data preprocessing: Data that was extracted and parsed following the above process contained anomalies data 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 parsed data inappropriately in some cases. For example, copyright text "All 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.

5.4 Analysis Procedure

This section discussed the methods used to construct the communication networks, architectures and to measure socio-technical congruence (defined in section 2) utilizing the data collected from the FreeBSD project.

Developer Contribution: Release-wise developer contribution was measured 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. 6(a) shows a sample contribution made by developer *John Birrell* in stable release 3.

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/uthread/	uthread_attr_getstacksize.c	jb@cimlogic.com.au	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

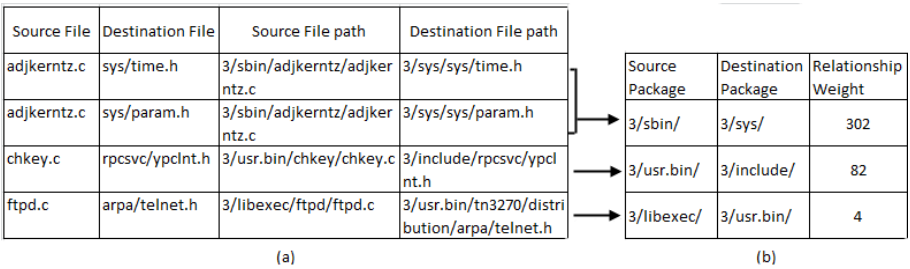
(b)

Fig. 6. (a) Sample contributions made by developer John Birrell (b) Sample relationships in Concrete Coordination Network

Concrete Architecture: The concrete architecture of a stable release was constructed based on header file inclusion dependency as defined in section 2.2. This inclusion dependency relation was used in earlier works [32] [35] to construct architecture of legacy C/C++ software systems.

Then higher level abstraction of this architecture was built to get the package level concrete architecture. In this architecture, package p1 and p2 had a relationship if file f1 in package p1 had an inclusion dependency with a file f2 in package p2 or vice-versa. Relationship between packages were weighted which was the total number of inclusion relationships that exists between the files in two packages. An example of file level concrete architecture and corresponding

package level architecture of stable release 3 is shown in Fig. 7(a) and (b) respectively. Package level concrete architecture was constructed for each stable release.



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)

Fig. 7. (a) Code file level Concrete Architecture (b) Package level Concrete Architecture

Concrete Coordination Network: Common email conversation in FreeBSD can appear in any of the email archives listed in Fig. 4. This relationship was weighted, meaning that for each new instance of email conversation between the same developers, the relationship weight would be increased by one. For each stable release of FreeBSD, one such Concrete Coordination Network was constructed. Fig. 6(b) shows example relationships in the Concrete Coordination Network of stable release 3. Weight column in this figure shows the number of emails common between the two developers.

Derived Architecture: Derived Architecture was generated based on the definition in Section 2.4. Each package relationship in this architecture was weighted and would increase if either of the two criteria hold for other developers. Both package level and code file leave derived architectures were constructed for each stable release. Fig. 8(a) shows the package level Derived Architecture for stable release 3.

Derived Coordination Network: Derived Coordination Network was generated based on the package level Concrete Architecture and for each stable release following the definition presented in Section 2.5. Each relationship was weighted. Fig. 8(b) shows an example of this network for stable release 3.

Thus this network shows the actual communication need among the developers which is based on the design of the software (i.e., the concrete architecture). This network is essential due to the fact that if two subsystems are to communicate, it is likely to have communication between the developers of the two subsystems [1].

Socio-technical Congruence: To measure socio-technical congruence the following method was applied: common relationships are identified that exists between (a) Concrete Architecture and Derived Architecture, and between (b)

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)

Fig. 8. (a) Derived Architecture (b) Derived Coordination Network

Concrete Coordination Network and Derived Coordination Network. The two sets of relationships identified in this process is termed as congruence.

The former congruence illustrates the match between the architectural dependency and the architecture produced due to the communication structure of the community. And the latter congruence depicts the match between the actual coordination activities in the community and the coordination need established by the architectural dependency of the software. To be precise, these congruences verify Conway's law and the reverse Conway's law, respectively. Both form of congruences were determined for each stable release. A partial snapshot of the congruence between Concrete and Derived Architectures of stable release 3 is shown in Fig. 9.

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 (in Derived architecture)	Weight (in Concrete architecture)
3/usr.bin/	3/include/	365	82
3/sbin/	3/sys/	806	302

(c)

Fig. 9. (a) Concrete Architecture (b) Derived Architecture (c) Congruence

To construct the networks and architectures and to measure socio-technical congruence and Conway's law, Java programs were written.

Result Interpretation: To measure the extent to which Conway's law holds for each stable release of the FreeBSD project, percentile value was calculated for (a) the congruence value and the corresponding Concrete Architecture, and (b) the congruence value and corresponding Coordination Network. These percentile values were plotted in a graph against the stable releases to conceptualize the evolution of the Conway's law and socio-technical congruence in the FreeBSD project.

6 Results

6.1 Resemblance of Communication Pattern to Software Architecture

Our study target in this work was to verify the extent to which the communication patterns of the members of the developer community resembles the actual architectural dependencies. To achieve this, we collected historical data from the FreeBSD project and measured such resemblance for each stable release. The resemblance process consists of determining the concrete architecture and derived architecture for each stable release. Then, the congruence between the two architectures and corresponding percentile measure of Conway's law for each release was measured. The result of this process is reported in Fig. 10. Column 2 and 3 in this figure show the number of relations between packages identified by the respective architectures. The congruence relationships and the percentile value for each release indicate the extent to which the two architectures of the software overlap with each other. Here the percentile value is measured between the congruence and the concrete architecture, because the intension is to measure the extent to which the derived architecture approximates the concrete architecture.

Stable releases	No of Relationships among packages			
	Derived Architecture	Concrete Architecture	Congruence	(%) Conway's law
stable-2.0.5	104	33	22	66,67
stable-2.1	91	38	26	68,43
stable-2.2	211	47	34	72,35
stable-3	153	27	22	81,49
stable-4	153	49	40	81,64
stable-5	136	52	41	78,85
stable-6	136	48	38	79,17
stable-7	190	53	40	75,48
stable-8	193	55	44	80
stable-9	178	56	45	80,36

Fig. 10. Resemblance of communication pattern to software architecture

The percentile values suggest that around 76% to 82% overlap between the two architecutres are found from stable release 3 onward. Whereas for the earlier three releases it is between 66% to 73%. It is worth to mention here that we noted considerable drift in the congruence value for the first three releases of the FreeBSD project. Thus we excluded them from our observation, considering this period as a restructuring and reformation period of FreeBSD after being forked. However, this observation is a positive sign towards the socio-technical congrucence, and can be an implicit charactersitcs of OSS development process. From this result, we can safely say that to a considerable extent the communication of the contributing developers in the community is actually due to the coordination need as identified by the architectural dependency.

Now the question is, does the derived architecture actually resembles the concrete architecture, and can it be used to recover the architecture of the legacy software? By examining Fig. 10 it is evident that the derived architecture is overestimating the concrete architecture for all the releases. For instance, in stable release 4, the derived architecture identifies 153 relations among the 19 packages, whereas concrete architecture shows only 49. Thus, it is not possible to resemble the concrete architecture from the derived architecture. Yet, the following observation can be offered.

To a great part, the communication pattern of the contributing members within the community is due to the communication needs established by the concrete architecture.

The interdependency among the packages in the concrete architecture influences their contributors to communicate at community level. This supports the existence of Conway's law within the FreeBSD development process. None-theless, both the architectures overlap considerably. Thus the derived architecture can be used in authenticating the architecture recovered by traditional reverse engineering process.

This over estimation by the derived architecture can be justified in a way that this architecture was derived from the communication record of their contributing developers. And the developers in the community can communicate and collaborate on issues that might be outside of the scope of the actual code implementation and commits. For instance, FreeBSD-chat archive collects only the email threads related to the general discussion. These discussions generate additional relationships among the contributing developers which were not due to only common contribution. In the derived architecture these relationships create additional links between the packages. Yet it can also be possible that some of the links between packages in the derived architecture reflect valid relationships for the release, as the concrete architecture might not identify all the links that actually exists for that release.

6.2 Resemblance of Software Architecture to Communication Pattern

Conventional wisdom supports that the socio-technical congruence measured in reverse (i.e., reverse Conway's law) should hold as well. That is the communication pattern identified by the concrete architecture should simulate the actual communication pattern of the contributing community members. To measure this we constructed the coordination network and the derived coordination network. These networks show the communication among the contributing developers identified by the actual communication through email and by the concrete architecture, respectively. Then the congruence between these two networks and corresponding percentile value was calculated for each stable release. The percentile value indicates the extent to which both networks overlap. Relationships identified by each of these networks are presented in Fig.11.

Stable releases	No of Relationships among contributing developers			
	Derived Coordination Network	Concrete Coordination Network	Congruence	(%) Reverse Conway's law
stable-2.0.5	11017	925	574	62,06
stable-2.1	14884	805	443	55,04
stable-2.2	39059	6614	2165	32,74
stable-3	53152	5635	3980	70,63
stable-4	138228	4195	2983	71,11
stable-5	198602	11590	8253	71,21
stable-6	286244	17113	13634	79,68
stable-7	413188	10828	9453	87,31
stable-8	504458	7167	5768	80,48
stable-9	550427	4073	2857	70,15

Fig. 11. Resemblance of software architecture to communication pattern

It can be noted that the architecture of FreeBSD remains quite stable in terms of number of packages (Fig.3) and their interdependency as identified by the concrete architecture (column 3 of Fig.10). Thus it can be ascertained that

The architectural design of the software remains stable during its evolution.

As can be seen from Fig. 11 the congruence values remain in-between 71% to 88% from stable release 3 to 9 (ignoring the first three releases as discussed in section 6.1). This result is closely tied with the results reported in Fig. 10. This implies that socio-technical congruence in reverse also holds for FreeBSD. And the communication pattern of contributing developer community simulates the underlying architectural dependency of the software to a gereat extent. Yet the derived coordination network is not able to estimate the coordination network, the former network overestimates the later one, similar to the derived architecture. Instead the following observatioin can be offered.

The communication need defined by the architectural design and interdependency among the modules of the software is effectively embedded within the communication pattern of the developer community of the FreeBSD project.

Again the over estimation by the derived coordination network can be vindicated as follows, this network was derived from package level concrete architecture. Thus developers contributing to a package or to the related packages were considered to have communication among them. This lead to a number of relationships among developers in the derived coordination architecture who might not have contributed to the same code files, but to the same packages. Yet these relationships among contributing developers should be taken as a suggestion to improve the coordination further in the community.

6.3 Evolution of Socio-technical Congruence in FreeBSD Project

To conceptualize the evolution of socio-technical congruence in FreeBSD we plotted the congruence values in graph against the stable releases. As shown in

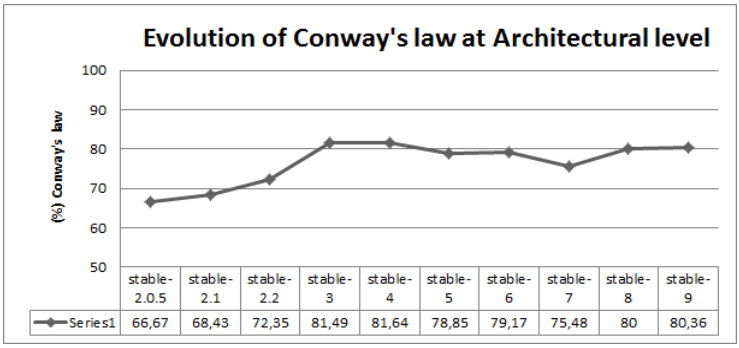


Fig. 12. Evolution of Conway's law at architectural level

Fig. 12 the congruence values remains stable around 76% to 82% starting from stable release 3.

Similar trend can be notified in Fig. 13 where the congruence value remains stable within the range 71% to 88% starting from stable release 3. Based on these observations it can be stated that with the maturation of the project, the communication pattern among developers get more structured following the need of communication based on their tasks. This socio-technical congruence pattern of communication among the community memebrs becomes a traditional practice as the project evolves. Thus it can be affirmed that

The socio-technical congruence in FreeBSD project remains stable during its evolution.

7 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, FreeBSD project was selected, which has been used popularly in the OSS evolution studies. Also, FreeBSD is a large and well established OSS project with over thirty years of evolution history. Yet the findings may not generalize to other OSS projects. This threat can only be countered by doing additional case studies with projects from different domains, which is a part of our future work.

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. Also other email archives may contain

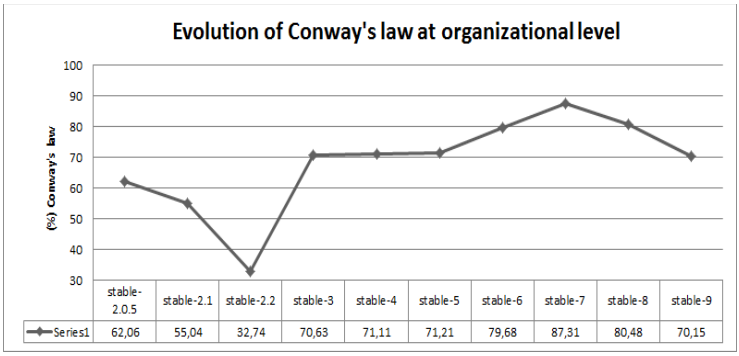


Fig. 13. Evolution of Conway's law at organizational level

relevant data. 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): In this study, part of the email entries were categorized to 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.

8 Conclusions

In this paper we evaluated the concept of Conway's law as a means of verifying the socio-technical congruence within the context of FreeBSD project. According to our judgment the congruence measure is significantly high in FreeBSD project which has a stable evolution history for the last seven releases of the project. In other words, the communication pattern of the contributing members within FreeBSD community are due to the communication need established by the concrete architecture. Having such high congruence between the code and the community would resolve many contemporary questions that cannot be answered otherwise. For instance, in identifying exact community member(s) to contract for an specific issue [36], it is required to trace which community members collaborates and exchange knowledge based on their task level responsibility.

This congruence is a desired property for collaborative development activities [14] and the reported result conforms the same for collaborative open source community. This result also creates a favorable basis to explore its implications on the OSS projects concerning the quality, success, and sustainability, which were already examined and confirmed in closed source projects.

Also, the results reported in this work hold for a specific OSS project, hence the study suffers from lack of generalizability. Thus a part of our future work is to extend and examine the findings in other OSS project, preferably to the BSD group and operating system domain.

References

1. Conway, M.E.: How Do Committees Invent? Datamation 14(4), 28–31 (1968)
2. Kwan, I., Schröter, A., Damian, D.: Does Socio-Technical Congruence Have an Effect on Software Build Success? A Study of Coordination in a Software Project. TSE 37(3), 307–324 (2011)
3. Ovaska, P., Rossi, M., Marttiin, P.: Architecture as a coordination tool in multi-site software development. Soft. Process: Improvement & Prac., 233–247 (2003)
4. de Souza, C.R.B., Quirk, S., Trainer, E., Redmiles, D.F.: Supporting collaborative software development through the visualization of socio-technical dependencies. In: International ACM Conference on Supporting Group Work, pp. 147–156 (2007)
5. Scacchi, W.: Understanding the requirements for developing open source software systems. IEE Proceedings Software 149(1), 24–39 (2002)
6. Bendifallah, S., Scacchi, W.: Work Structures and Shifts: An Empirical Analysis of Software Specification Teamwork. In: 11th ICSE, pp. 260–270 (1989)
7. Jongdae, H., Chisu, W., Byungjeong, L.: Extracting Development Organization from Open Source Software. In: 16th APSEC, pp. 441–448. IEEE (2009)
8. Raymond, E.S.: The new hacker's dictionary, 3rd edn. MIT Press, Cambridge (1996)
9. Nagappan, N., Murphy, B., Basili, V.R.: Architectures, Coordination, and Distance: Conway's Law and Beyond. Journal IEEE Software 16(5), 63–70 (1999)
10. Kwan, I., Cataldo, M., Damian, D.: Conway's Law Revisited: The Evidence for a Task-Based Perspective. IEEE Software 29(1), 90–93 (2012)
11. Brooks, F.P.: The Mythical Man-Month, Anniversary Edition. Addison-Wesley Publishing Company (1995)
12. Nagappan, N., Murphy, B., Basili, V.R.: The influence of organizational structure on software quality: an empirical case study. In: ICSE 2008, pp. 521–530 (2008)
13. Cataldo, M., Wagstrom, P.A., Herbsleb, J.D., Carley, K.M.: Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In: CSCW, Banff, Canada (2006)
14. Browning, T.: Applying the design structure matrix to system decomposition and integration problems: a review and new directions. IEEE Transactions on Engineering Management 48(3), 292–306 (2001)
15. Sosa, M.E., Eppinger, S.D., Rowles, C.M.: The misalignment of product architecture and organizational structure in complex product development. Management Science 50(12), 1674–1689 (2004)
16. Jingwei, W., Holt, R.C., Hassan, A.E.: Empirical Evidence for SOC Dynamics in Software Evolution. In: ICSM, pp. 244–254 (2007)
17. Herraiz, I.: A statistical examination of the evolution and properties of libre software. In: ICSM, pp. 439–442 (2009)
18. Herraiz, I., Gonzalez-Barahona, J.M., Robles, G., German, D.M.: On the prediction of the evolution of libre software projects. In: ICSM, pp. 405–414 (2007)
19. Fogel, K., Bar, M.: Open Source Development with CVS: Learn How to Work With Open Source Software. The Coriolis Group (1999)
20. Herbsleb, J.D., Grinter, R.E.: Splitting the Organization and Integrating the Code: Conway's Law Revisited. In: ICSE, pp. 85–95. ACM Press, New York (1999)
21. Baldwin, C.Y., Clark, K.B.: Design Rules: The Power of Modularity. MIT (2000)
22. Colfer, L., Baldwin, C.Y.: The Mirroring Hypothesis: Theory, Evidence and Exceptions, Working paper. Harvard Business School (2010)

23. Bowman, I.T., Holt, R.C.: Software Architecture Recovery Using Conway's Law. In: CASCON 1998, pp. 123–133 (1998)
24. FreeBSD (2013), http://www.freebsd.org/
25. Mathieu, G., Mens, T.: A framework for analyzing and visualizing open source software ecosystems. In: IWPSE-EVOL, pp. 42–47 (2010)
26. Daniel, M.G.: Using software trails to reconstruct the evolution of software. Journal of Software Maintenance and Evolution 16(6), 367–384 (2004)
27. Wang, Y., Guo, D., Shi, H.: Measuring the Evolution of Open Source Software Systems with their Communities. ACM SIGSOFT Notes 32(6) (2007)
28. Zhang, W., Yang, Y., Wang, Q.: Network Analysis of OSS Evolution: An Empirical Study on ArgoUML Project. In: IWPSE-EVOL, pp. 71–80 (2011)
29. Apache POI-Java API for Microsoft Documents (2013), http://poi.apache.org/
30. jsoup: Java HTML Parser (2013), http://jsoup.org/
31. Yamauchi, Y., Yokozawa, M., Shinohara, T., Ishida, T.: Collaboration with Lean Media: How Open-source Software Succeeds. In: CSCW, pp. 329–338 (2000)
32. Dayani-Fard, H., Yu, Y., Mylopoulos, J., Andritsos, P.: Improving the build architecture of legacy C/C++ software systems. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 96–110. Springer, Heidelberg (2005)
33. Mahbubul Syeed, M.M.: Binoculars: Comprehending Open Source Projects through graphs. In: Hammouda, I., Lundell, B., Mikkonen, T., Scacchi, W. (eds.) OSS 2012. IFIP AICT, vol. 378, pp. 350–355. Springer, Heidelberg (2012)
34. Bolici, F., Howison, J., Crowston, K.: Coordination without discussion? Socio-technical congruence and Stigmergy in Free and Open Source Software projects. In: 2nd STC, ICSE (2009)
35. Kazman, R., Carrière, S.J.: Playing Detective: Reconstructing Software Architecture from Available Evidence, Technical Re- p ort CMU/SEI-97-TR-010, Carnegie Mellon University (1997)
36. Syeed, M.M., Altonen, T., Hammouda, I., Systä, T.: Tool Assisted Analysis of Open Source Projects: A Multi-facet Challenge. IJOSSP 3(2) (2011)