ISSN 2278-3091
Volume 9, No.3, May - June 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse182932020.pdf
https://doi.org/10.30534/ijatcse/2020/182932020

Modelling Lower-level Technical Dependencies to Improve Coordination in Software Engineering Projects : A Conceptualization

N.M.Nor[1], S.S.M.Fauzi[1*], R.Ahmad[2], M.M.Rosli[3]
[1]Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Perlis Branch, Arau Campus, 02600, Arau, Perlis, Malaysia, shukorsanim@uitm.edu.my
[2]Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia, rodina@um.edu.my
[3]Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, 40450 Shah Alam, Selangor Darul Ehsan, Malaysia, marshima@tmsk.uitm.edu.my

ABSTRACT

Socio-Technical Congruence (STC) is one of the most significant current discussions in software engineering community and have been used for measuring developer's coordination. Over the past few years, increasing interest been observed in improvement of measure for STC, and its relationship with task performance, software quality and software productivity. However, a major limitation with this kind of measure is the inability to represent lower level of technical dependencies and only conceptualize a limited range of actual coordination. This study therefore provides a conceptualization to integrate lower-level technical dependencies and actual coordination, and examine the connection of STC on task complexity in distributed software development. Thus, we outline the literature review, research methodology, potential contributions and the expected findings of the study.

Key words : Coordination, Dependencies, Software Development, Software Engineering Projects, Technical Dependencies.

1. INTRODUCTION

In the past decade, a large and growing body of literature suggests that social and technical dependencies, are essential in attaining effective developer coordination and collaboration [1]. For instance, in software application, current dependencies might be modified, and social organization specifically team membership will evolve when the new code dependencies pop up [2]. Literature evidence has discussed how a project can matched its coordination activities to fulfill the technical requirements so that optimal coordination activities can occur in projects [2]. Over the past decade, there have been several approaches to improvising coordination in software engineering projects. For instance, a study suggested the information hiding and modularization theory that intends to alleviate software development complexity. By reducing dependencies at the artefact level, it is possible to reduce dependencies among developers [3]. However, these approaches have limitations that can impact developer coordination. For example, as technical interdependencies inside system module decrease, coordination needs among teammates also decrease. Less communication causes interpretations and misconceptions [4]. Therefore, there exists Socio-Technical Congruence (STC) method, which measures the match among coordination needs and actual coordination activities [5].

Despite the recognized positive effect of STC, some of the STC's shortcomings and fundamental issues still need to be addressed. Firstly, a recent study on STC suggested the limitation that the conceptualizations of STC do not represent lower-level technical dependencies and only conceptualize a limited range of actual coordination among developers [6]. Secondly, little evidence established the relationship of STC on task complexity, and prior literature on STC only focused on a limited range of development settings [6]. Thus, STC to be further explored in many aspects in order to emphasize it to be classified among software development established theories.

Accordingly, this study attempts to enhance the current STC model by constructing an integrated model of socio-technical congruence that conceptualizes lower-level technical dependencies and to examine the socio-technical congruence correlation on task complexity in distributed software development.

This study has been organized in the following way. The first section of this study details the literature review of STC. The second section is concerned with the methodology used for this study. Third section begins by laying out the expected potential contributions of this study. The fourth section presents the expected findings of this study. Finally, the conclusion gives a brief summary of this study.

2. LITERATURE REVIEW

A study found that complex and extensive software engineering projects have many technical dependencies [7]. Throughout this paper, the term technical dependency will refer to "a type of dependency between two technical entities" [8]. Various types of technical dependencies exist in software engineering projects, for instance modules that rely on other modules [9]. The presence of technical dependencies in software projects will not only increase the complexity of the project but will also impact the coordination among software developers [10].

Hence, Socio-Technical Congruence (STC) was introduced as a conceptualization used to quantitatively study social and technical aspects in software engineering projects [10]. STC is specified as "the match among the coordination needs, established by the dependencies between tasks, and the actual coordination activities performed by developers" [10]. STC conceptualization was first carried out by Melvin Conway (1968), who observe that the system structure resembles the organization structure whose constructed it [11]. Alignment between developers formed when the developers collaborate on the same tasks to keep abreast of the changes taking place in the system. In Conway's Law arguments, products should be break down into components with restricted technical dependencies in order to reduce communication overhead [11]. However, due to the growth of the worldwide distributed project, coordination also has become more challenging. Then, information hiding and modularization theory that intends to alleviate software development complexity was suggested. This theory highlights the possibility to reduce dependencies among developers, by reducing dependencies at the artefact level [12]. However, these approaches have limitations that can impact developer coordination and could leads to coordination problems [13]. Thus, STC has been introduced since the coordination required for the more powerful technique to technical coordination [5].

In recent years, there has been an increasing amount of literature on STC. Several studies thus far have paraphrased the definition of socio-technical congruence [3],[10]. The properties of STC have been explored in a study [14]. A significant number of studies on the effectiveness of socio-technical congruence has been reported. It has conclusively showed that higher level of congruence could reduce the time for resolution of a modification request, decrease errors, boost productivity and reducing cost [3],[10], [15]. Besides, in a study which set out to investigate the measures for calculating STC, a model that involves weighted edges has been proposed [15]. It has been demonstrated that STC has a positive impact on software builds in software engineering projects [8]. A prior empirical study created an ISTC metric to collect STC measure for a person. The research study wanted to discuss potential large-scale project problems created by involvement in a variety of projects that could affect the coordination needs of the project [16]. On the other hand, another study broadened the STC measure by adding extra congruence scales for resource-dependent and knowledge-dependent congruence. The study also examined the relationship between STC on team performance and found that a higher degree of congruence could have obtained when there is a greater interaction between project team [17].

To manage STC at Global Software Development (GSD), a researcher proposed a multi-agent architecture. In the study, the researcher expanded the range of the congruence metric to represent all project stages [18]. Derived from social network approaches, a quantitative measure of social interaction has been proposed [19]. Another study reported that low congruence and having several coordination problems could substantially raise the project failures number [13]. On the other hand, other studies investigate the Conway's Law and Reverse Conway's Law throughout the open-source development projects environment by introducing a different measurement method. This study later explores the significance of STC as the project matures [20]. The researchers focused their research on FreeBSD project. They found out when the FreeBSD project had such a steady background of development over the past seven releases, the congruence measure is substantially high [20]. The same authors then investigate the correlations among the coordination activities of the project team and the dependency structure of project in the Ruby ecosystem [21]. The study conclusively reported that the teamwork pattern between the Ruby ecosystem's developers is not inevitably demonstrated the communication requirements showed by the dependencies between projects in the ecosystem.

Prior study outlines result from experimental research that lead to a deeper understanding of STC by analyzing benefits and disadvantages of congruent and incongruent plots. They observed that, amendments in the communication structure alone cause amendments in the software products structure [22]. Further, they establish a rubber band effect hypothesis and counsel a replication research that outgrowing the fresh insights of Conway's law by examining the progress of STC over the years [22]. In an analysis of STC in the ambience of open-source component, a researcher recognizes the necessity for a new model of STC for dependencies among projects. The study proposed a need to revise the function of STC in open-source component use since less interaction intervenes as an open-source component is utilized more frequently [23]. In a study which designed to determine the impact of STC on software development, an author investigated STC and task performance’s connection in software development life cycle incremental model. The research investigated the insight of the incremental model. In the same vein with prior research, they agreed that a technique of breakdown task helps lessens dependencies which deliver good performance [24]. The researchers then constructed a model of the STC and project performance’s relationship in the various types of software development lifecycle [25]. Another research was performed to study the effect of STC on software quality. By using defects as a measurement of software quality, this research expects that higher congruence level in a project team will lower the defects quantity and increase quality [26].

A recent study has suggested an approach to calculate STC build-level for the utilization in prolonged fault prediction. This finding indicated that STC build-level had a great impact on ongoing integration build failures, and useful in continuous flaw prediction [27]. Later, the same authors explore the connection among STC file-level and bug proneness in Open Source Software (OSS) projects. Some required modifications were implemented to the technique of calculating the coordination needs and coordination activities at file-level compared to the original study [27].

Despite the recognized positive effect of STC, STC need to be further explored in many aspects in order to emphasize it to be classified among software development established theories.

3. METHODOLOGY

Figure 1 illustrate the proposed Socio Technical Congruence (STC) model.

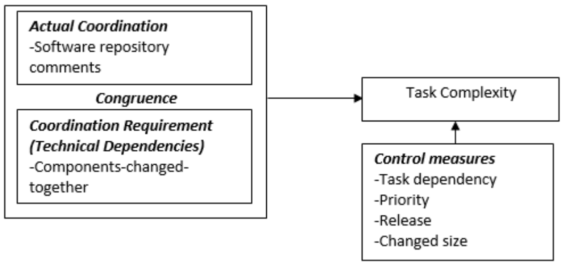
Actual Coordination
-Software repository comments
Congruence
Coordination Requirement (Technical Dependencies)
-Components-changed-together
Task Complexity
Control measures
-Task dependency
-Priority
-Release
-Changed size

Figure 1: Proposed Socio-Technical Congruence (STC) model

A brief description about each variable is discussed below:

Coordination Requirement is a link that suggests that two developers must be coordinated depending on their assignment of tasks [13]. Data for coordination requirement will be extracted from various sources. As for instance, modification request report, version control system and also source code itself. In this study, the coordination requirement is derived from task dependencies occurring in components-changed -together. Each task (Modification Request) typically has components that need to be changed by the developer(s). Hence, the coordination requirement of an MR task is treated as a group. The study presumed a coordination requirement among developers occurs when, at least, two developers modify the similar component in the similar MR task.

Actual coordination is the interaction between developers that occurs in the project [13]. Actual coordination will be calculated based on communications that occurred in the software repository comment. Actual coordination is calculated by developing table matrices that indicate the communication that occurred among person *i* and person *j*.

Congruence is described as the fit among coordination requirements and actual coordination. It is measured by matching the coordination requirement matrix with actual coordination matrix [14]. The value of congruence is from 0 to 1. Congruence will be computed by comparing table matrices between coordination requirement and actual coordination.

Task Complexity is an indicator of the resources a system expends while dealing with a software program to accomplish a task. It is measured on the basis of program code which disregards comments and stylistic features such as indentation and naming conventions. Usually, the measures rely on the size of the program, the control structure, or the nature of the module interfaces [28].

Due to a more limited range of variables available in the open-source project datasets, four of these control variables were able to be applied in this study, namely, task dependency, priority, release and change size. First is task dependency, which is a measure of the affected MRs which need to be referred to in order to accomplish the task. It is measured as a number of MRs that are depended on in performing the task (#of MRs). Second is priority, which is the order in which the task should be fixed. It will be measured as a value assigned to MRs to represent the level of criticality of the MRs. Third is release, which is the ordinal number of the software version contributed to be the. It is measured as the number of releases of the product with which the MR is associated. Lastly change size, is an indicator for the total number of development work performed. The change size is measured as the sum of files and components that are altered as part of the MR change.

Open source project will be selected to conduct the research. This research will fully utilize R script to perform data extraction, cleaning, and statistical regression. Thus, script needs to be developed and tested before being implemented.

This study will use archived data from a Modification Request (MR) bugs repository and software repository comments. We will extract data for this study by using Mining Software Repository (MSR) method. The MR bugs repository consists of a bundle of information related to changes in each project. This information can be manipulated and analyzed to understand the coordination requirement among the developers. Coordination requirement is derived from the technical dependencies occurring in components-changed-together.

The study will use linear regression to identify the association among socio-technical congruence and task complexity in distributed software development projects.

4. POTENTIAL CONTRIBUTIONS

This research will potentially provide the following contributions: 1) an improved STC model, and 2) an additional piece of empirical evidence on the relationship between the improved STC model on task complexity in distributed software development.

This study also extends the coordination requirements by using 'components-change-together' which differ from previous studies. Besides, this study also provides a valuable opportunity to advance the understanding of how social factors affect the working mechanism, and the used of technical systems. This interpretation offers some crucial insights into the software industry because it can lead to the improvement of organizational structures and technical frameworks and therefore, can contribute to the creation of more suitable systems to end-users.

5. EXPECTED RESULT

In this study, we expect to find higher levels of congruence between lower-level technical dependencies is associated with task complexity.

6. CONCLUSION

The intention of this research is to integrate lower-level technical dependencies and actual coordination and examine the connection of STC on task complexity in distributed software development. In this study, an outline of the literature review, research methodology, potential contributions and expected outcomes of this study are presented. Therefore, this study should make an essential contribution to the field of software development by representing lower-level technical dependencies in the conceptualization of socio-technical congruence. Understanding how socio-technical congruence will manifest itself in the new ways of working will also help understanding how future software should be created and maintained.

ACKNOWLEDGEMENT

This research was supported by Ministry of Higher Education Malaysia under Fundamental Research Grant Scheme (FRGS), grant (600-IRMI/FRGS 5/3/ (092/2019)).

REFERENCES

1. R. E. Grinter. ***Understanding dependencies: A study of the coordination challenges in software development***, Ph.D. dissertation, University of California, Irvine, 1996.
2. M. Cataldo. **Dependencies in geographically distributed software development: overcoming the limits of modularity**, *Dissertation Abstracts International*, vol. 68, no. 12, December 2007. https://doi.org/10.1145/1414004.1414008
3. M. Cataldo, J. D. Herbsleb, and K. M. Carley, **Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity**, in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, New York, 2008, pp.2-11.
4. R. Hassani, and Y. E. B. El Idrissi. **A framework to succeed planning of IT projects through the Machine Learning**. *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 9, no. 2, March-April 2020 https://doi.org/10.30534/ijatcse/2020/220922020
5. J. Herbsleb, M. Cataldo, D. Damian, P. Devenbu, Easterbrook, S., and A. Mockus. **Socio-technical congruence (STC 2008),** in *Companion of the 30th international conference on Software engineering*. 2008, pp. 1027-1028.
6. J. M. Sierra, A. Vizcaíno, M. Genero, and M. Piattini. **A systematic mapping study about socio-technical congruence**, *Information and Software Technology*, vol. 94, pp. 111–129, 2018.
7. N. Sekitoleko, F. Evbota, E. Knauss, A. Sandberg, M. Chaudron, and H. H. Olsson. **Technical dependency challenges in large-scale agile software development,** in *International Conference on Agile Software Development*, Springer, Cham, 2014, pp. 46-61. https://doi.org/10.1007/978-3-319-06862-6_4
8. I. Kwan, A. Schroter, and D. Damian. **Does socio-technical congruence have an effect on software build success? a study of coordination in a software project**. *IEEE Transactions on Software Engineering*, vol.37, no.3, pp.307-324, 2011.
9. P. Rani, and Katari. **An Application for the Development of Smart Library as an Academic Initiative**, *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, no. 1.3, pp. 55-58, 2019.
10. 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 *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, 2006, pp. 353–362.

11. M. E. Conway. **How do committees invent**. *Datamation*, vol. 14, no.4, pp.28-31, 1968.
12. D. L. Parnas. **On the criteria to be used in decomposing systems into modules**, *Communications of ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972. https://doi.org/10.1145/361598.361623
13. M. Cataldo, and J. D. Herbsleb. **Coordination breakdowns and their impact on development productivity and software failures**, *IEEE Transactions on Software Engineering*, vol.*39*, no.3, pp.343-360, 2012.
14. A. Sarma, J. Herbsleb, and A. Van Der Hoek. **Challenges in measuring, understanding, and achieving social-technical congruence**, in *Proceedings of Socio-Technical Congruence Workshop, In Conjuction With the International Conference on Software Engineering*, 2008.
15. F. Bolici, J. Howison, and K. Crowston. **Coordination without discussion? Socio-technical congruence and Stigmergy in Free and Open Source Software projects**, in *Socio-Technical Congruence Workshop in conj International Conference on Software Engineering*, 2009.
16. P. Wagstrom, J. D. Herbsleb, and K. M. Carley. **Communication, Team Performance, and the Individual: Bridging Technical Dependencies**, in *Academy of Management Proceedings*, 2010, pp. 1–7.
17. L. Jiang, K. M. Carley, and A. Eberlein. **Assessing team performance from a socio-technical congruence perspective**, in *2012 International Conference on Software and System Process (ICSSP)*, 2012, pp. 160–169. https://doi.org/10.1109/ICSSP.2012.6225961
18. J. P. Rodríguez. **An Agent Architecture with which to Improve Coordination and Communication in Global Software Engineering**, Ph.D. dissertation, University of Castilla-La Mancha, Ciudad Real, Spain, 2013.
19. 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 *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*, 2007, pp. 25–25.
20. M. M. Syeed, and I. Hammouda. **Socio-Technical Dependencies in Forked OSS Projects: Evidence from the BSD Family**, *JSW*, vol.9, no.11, pp.2895-2909, 2014. https://doi.org/10.4304/jsw.9.11.2895-2909
21. M. M. Syeed, K. M. Hansen, I. Hammouda, and K. Manikas. **Socio-technical congruence in the ruby ecosystem**, in *Proceedings of The International Symposium on Open Collaboration*, 2014, pp. 2.
22. S. Betz, S. Fricker, A. Moss, W. Afzal, M. Svahnberg, C. Wohlin, and T. Gorschek. **An evolutionary perspective on socio-technical congruence: The rubber band effect**, in *2013 3rd International Workshop on Replication in Empirical Software Engineering Research*, 2013, pp. 15-24. https://doi.org/10.1109/RESER.2013.8
23. M. Palyart, G. C. Murphy, and V. Masrani. **A study of social interactions in open source component use**, *IEEE Transactions on Software Engineering*, vol.44, no.12, pp.1132-1145, 2017.
24. W. A. W. M. Sobri, S. S. M. Fauzi, M. H. N. Nasir, R. Ahmad, and A. J. Suali. **A mechanism to assess the relationship between socio-technical congruence and project performance in incremental model**. *Journal of Fundamental and Applied Sciences*, vol.9, no.5S, pp.58-74, 2017. https://doi.org/10.4314/jfas.v9i5s.6
25. W. A. W. M. Sobri, S. S. M. Fauzi, M. H. N. M. Nasir, R. Ahmad, and A. J. Suali. **Measuring The Impact of Socio-Technical Congruence in a Different Types of Software Life Cycle**, *Sains Humanika*, vol.9, no.1-3, 2017.
26. A. J. Suali, S. S. M. Fauzi, M. H. N. M. Nasir, and W. A. W. M. Sobri, **Assessing the Impact of Socio-Technical Dependencies on Software Defect**. *Sains Humanika*, vol.9, no.1-3, 2017.
27. W. Zhang, S. C. Cheung, Z. Chen, Y. Zhou., and B. Luo. **File-level socio-technical congruence and its relationship with bug proneness in OSS projects**. *Journal of Systems and Software*,2019. https://doi.org/10.1016/j.jss.2019.05.030
28. M. Vieira and D. Richardson. **Analyzing dependencies in large component-based systems**, in *Proceedings 17th IEEE International Conference on Automated Software Engineering*, 2002, pp. 241–244.