Decaying Socio-Technical Congruence as a Method to Account for Architectural Changes

Patrick Wagstrom, James Herbsleb, and Kathleen Carley
Institute for Software Research
School of Computer Science
Carnegie Mellon University
{pwagstro,jdh,carley}@cs.cmu.edu

Abstract

The socio-technical metric (STC) [1,2] relates the coordination activities on a team to the coordination requirements generated by the interaction of dependencies between tasks and assignment of tasks. In this paper we extend STC to include a decay factor to account for changes in network structure over time. We evaluate the changes in a large Open Source community and find that small amounts of decay increase the overall explanatory power of the metric across a broad spectrum of projects and larger amounts of decay may be beneficial for projects with large changes in requirements and communication patterns.

1. Introduction

Functionally, many teams serve as information processing entities that manage an incoming stream of information, parsing and delivering the information to relevant team members and then doing work based on the combination of new and existing information [3,4]. As the team evolves, ad-hoc communication channels emerge, allowing information to flow more easily and easing the coordination difficulties that the team may face [5,6]. These communication channels often arise as a response to technical constraints inherent in the task and the task's network of dependencies. As these technical constraints linking components of the task change, the team must alter their communication patterns to ensure they can continue to complete their task efficiently. However, even a small change in the technical constraints of the system can have potentially widespread changes on the coordination requirements of the team, rendering old communication channels obsolete and forcing the team to develop new channels for information[7].

While the above statements may be true for almost any team, they are particularly relevant for Software Engineering teams, which frequently face additional challenges. Software engineering projects often are poorly specified, long lived, and face unforeseen difficulties.

One important way that software engineering differentiates itself from traditional engineering is the long life of the projects and the need to support the project on changing platforms over long time periods. While many complex engineering projects, such as the NASA space shuttle receive incremental updates over the course of their lifetime, their domain is generally fixed. A space shuttle is designed to take into account the forces of earth and space to safely provide transit for a crew of astronauts. While electronics upgrades and small advancements in materials may make their way into refits of the shuttle, it will not be refit to work on another planet, to move astronauts around the globe, or to land on the surface of the moon[8].

Software, on the other hand, frequently faces changing requirements. The operating systems that the software runs on are constantly being upgraded, introducing small changes to APIs, program timing, and the rights allowed to the program. Devices on the platform also change, sometimes requiring difficult work-arounds when upgraded versions of the software are released. New programming languages, libraries, and even programming paradigms are constantly being developed that can dramatically improve the program should project developers and maintainers choose to update and re-write their code to utilize these libraries. However, rather than re-writing the program, software engineers are expected, to a degree, to maintain compatibility with older legacy systems[9].

While there exists a handful of projects that developers hand-off after release, the majority of projects utilize the same team for multiple releases over a period of years. This paper examines the effects

of STC on long lived software engineering projects to understand how architectural changes affect the STC metric. We propose an adaptation to STC calculation that implements a decay factor in communication and coordination requirements. We evaluate the metric against several projects from a single community and then discuss the larger implications of allowing communication and networks to decay over time.

2. Socio-Technical Congruence

The Socio-Technical Congruence metric as proposed by Cataldo et. al. assess the fit of an organization's actual coordination to the set of coordination requirements found in artifacts of the project [1,2]. In a nutshell, the metric works as follows:

For a given organization three different networks are collected in matrix form. The first is a task assignment network, $\boldsymbol{T_A}$, that maps individuals to tasks in the organization. Within a software development organization these tasks may be files, individual requirements, bug reports, or any other notion of a work item. Next, a task dependency network, $\boldsymbol{T_D}$, is collected that maps dependencies between different tasks. For example if task A requires task B to be completed before proceeding, an edge would appear in the network between A and B. Using these networks matrix multiplications is used to get the coordination requirements network, $\boldsymbol{C_R}$, between individuals in the network. This combination of networks in matrix form is largely based off the model combining relationships proposed by Carley and Krackhardt in their PCANS model of interaction [10].

$$\boldsymbol{C_R = T_A \times T_D \times T'_A}$$

This multiplication is done to get at the sometimes hidden dependencies between individuals within an organization. Often times individuals are aware of the other people working on a component, or those individuals who have worked on the component in the past, but they are not aware the way that their work impacts individuals that work on tasks which are dependent on their tasks.

The final network collected is a network representing actual coordination within the organization, $\boldsymbol{C_A}$. Examples of this network may be co-attendance at meetings, shared geographic locations, or archives of email messages. The socio-technical congruence for the organization is then the proportion of edges present in $\boldsymbol{C_R}$ that are also present in $\boldsymbol{C_A}$. As the metric does not take into account edge weights, it is possible to dichotomize all networks such that edges with positive weights are set to 1 and all other edges have weights of 0. In such a case, the following formula can be used to calculate overall STC.

$$STC = \frac{\sum(\boldsymbol{C_A} \wedge \boldsymbol{C_R})}{\sum \boldsymbol{C_R}}$$

For a single point in time, this formulation is quite useful, but it loses some usefulness when aggregating data over a longer period. This is primarily because over the life of a long-lived project changes in architecture, team structure, and task dependencies will begin to accumulate[9]. For example, while Windows Vista inherits portions of the code from Windows 95 and before, there is little doubt that the dependencies and the team structure have varied dramatically since that point. The work necessary to update the code for changes in architecture is frequently referred to as refactoring and often causes dramatic changes in the structure of the code and the social structure of the team developing the code[11,12].

To account for this, we propose adding a decay factor to the collected networks $\boldsymbol{C_A}$, $\boldsymbol{T_A}$, and $\boldsymbol{T_D}$. This factor, δ, is scaled between 0 and 1, where 0 indicates full decay, relying only the current data, and 1 indicates no decay. Thus, at a time period i the networks may be formulated as follows, where $\boldsymbol{c_A}$, $\boldsymbol{t_A}$, and $\boldsymbol{t_D}$ represent the contribution to the networks only from time period j.

$$\boldsymbol{C_{A_i}} = \sum_{j=0}^{i} \delta^{i-j} \boldsymbol{c_{A_j}}$$

$$\boldsymbol{T_{A_i}} = \sum_{j=0}^{i} \delta^{i-j} \boldsymbol{t_{A_j}}$$

$$\boldsymbol{T_{D_i}} = \sum_{j=0}^{i} \delta^{i-j} \boldsymbol{t_{D_j}}$$

This addition of a decay factor allows older task dependencies, task assignments and observed communications to be slowly removed from the network over time. By applying the decay factor to all three networks, it is possible that the value of socio-technical congruence at each time period can vary either positively or negatively.

3. Description of Data

To test the viability of STC within a real world project, it was necessary to locate a large community with significant history and ample available data. This was found in the GNOME project, a large Open Source project that seeks to create a completely free (as in money cost and liberty) desktop environment for Linux and Unix-like operating systems[13,14]. One aspect of GNOME that makes it interesting is that although the

community is working to build a coherent product, the GNOME desktop environment, functionally it is comprised of numerous smaller projects that are each given the freedom to operate relatively independently from one another[15].

The community is comprised of independent volunteers, students, and professional software developers. It has a moderate amount of commercial support, however commercial firms do not directly chart the direction of the community. Each project in the community is allowed to largely govern itself. There is a small board that coordinates major events in the community, such as software releases and the annual conferences.

For this paper, the complete version control history of the community was collected, spanning from the start of the project until the end of 2007. In addition a complete copy of the community's bug tracker database was obtained (many thanks to the system administrators of the GNOME project for their work in providing this). Finally, archives of project mailing lists, which are freely available via the World Wide Web were also downloaded.

One of the largest difficulties with this project is that there was no uniform sense of identity across projects in the community. Many developers had multiple accounts they used for accessing project source code, different email addresses used for mailing lists, and still other addresses used to register with the project bug tracker. As the lead author is active in the community, this allowed the identities of the 1200 developers who had contributed code directly to the project source code repository to be resolved, primarily by hand. Where uncertainty about developer identity occurred, individuals were validated by spot checking identities with active members of the community. This was a very time consuming process that largely was a result of the historical nature of GNOME. When the project was founded many of the tools that provide integrated environments for Open Source project management were not yet common, so each component of project management tools was obtained from a different source and managed using different tools[5,13,16]. Modern environments such as Launchpad from Canonical should make this form of data collection much easier for future Open Source projects[17].

Because the community operates in a relatively open manner, any individual with sufficient credentials is allowed to create a new project with little or no oversight. This has led to numerous small projects that are similar to toys or small spaces in which a developer experiments on new concepts. A filter was applied to the projects in the community to eliminate such low activity projects. Projects with more than 1 year of activity, contributions from 10 or more developers, a publicly accessible bug tracker, and publicly accessible mailing lists were selected for analysis.

Each of the networks for the project was generated as follows. Within the community, the concept of a task was mapped to individual files in the project version control system repository. The task assignment matrix, $\boldsymbol{T_A}$, was generated by examining the version control system archive for each file adding an edge between an individual and the file if the individual had made changes to that file and committed those changes back to the repository. Task dependencies between files, $\boldsymbol{T_D}$, were generated through an examination of logical coupling. If two files were committed together to the version control system then an edge was added between those files[18]. This method that has previously shown to be useful in the generation of networks for STC analysis [1,2]. Finally, the actual coordination network, $\boldsymbol{C_A}$, was generated through an examination of project mailing lists and bug trackers. Project mailing lists were broken up by thread and all individuals posting messages in a thread were connected to one another. Likewise, all individuals commenting on a bug were connected to one another, creating cliques for each conversation. The networks obtained from mailing lists and bug trackers were then aggregated to obtain $\boldsymbol{C_A}$.

Small amounts of additional filtering were done on the data. Commits to the version control system that touched more than 20 files were discarded as these are typically housekeeping and maintenance tasks, such as updating licenses or changing copyrights. Norms within the community help to ensure that commits typically touch far fewer files. In addition, all files that were not source code were removed and not considered, this includes project documentation, build scripts, and graphics.

Time slices in the community were set to one month intervals. The community makes two major releases a year, in March and September, however the release data of individual software packages does not necessarily correspond to these wider community release dates.

4. Preliminary Validation of STC

Before examining trends in STC by introducing a decay metric, it was first necessary to validate that sufficient data was collected to properly calculate STC for the community and that the same positive effect on team performance observed by Cataldo et. al. was once again observed[1]. The dependent variable in these observations was the time to resolve software defects

Table 1: Summary statistics of regression variables

Variable	Min	Max	Mean	Skew	Kurtosis
$\log_2(ResolutionTime)$	0.00	10.00	3.496	0.226	1.529
$numDevs$	1.00	5.00	1.303	1.855	6.243
$comments$	1.00	39.00	4.601	3.320	18.666
$deltaPeople$	1.00	9.00	1.695	2.093	9.385
STC	0.00	0.750	0.103	-0.067	1.287

on the key projects as measured by the $\log_2$ of the number of days the bug was open. Only actual software defects were analyzed, filtering out requests for new features, miscategorized issues, and duplicate bug reports.

For each bug report, several control variables were collected. Within large scale engineering projects, frequently adding additional developers to a project makes the project slower. To address this issue we tracked the number of developers, $numDevs$, who contributed code to the project who also were active on the bug report. The second control variable collected for each bug was the number of comments made on the bug. In the model this variable is called $comments$. Bugs that attract many comments are likely to be difficult or contentious issues within the organization and may lead to longer resolution times. The final control variable collected for each project was the number of people who changed the status of the bug, $deltaPeople$. A bug's status is its current state in the bug triaging and fixing process. All bugs begin as NEW, and then may progress to other states such as ASSIGNED, RESOLVED, NOTABUG, and NEEDINFO. A large number of people changing the status of a bug is a sign that a bug may be complicated or not properly fixed. A variety of other control variables were explored based on properties of the data set, but they were consistent too highly correlated with the other predictor valuable to prove valuable in the analysis.

There were 2859 bugs associated with projects for which STC could be calculated. This data was then used to build a regression model. The results of this analysis can be seen in Table 2. The number of developers and number of comments made on a bug have no statistical impact on the amount of time it takes to resolve a bug. This is largely explained by the lack of variation in these common metrics within the data set.

Table 2: Preliminary results of STC analysis within the GNOME community

Variable	Estimate	Std Error	P-Value
Intercept	1.7274	0.1461	<.0001
$numDevs$	0.0990	0.1049	0.346
$comments$	-0.0153	0.0155	0.323
$deltaPeople$	1.1812	0.0679	<.0001
STC	-4.0831	0.6042	<.0001
R^2=0.1577, DF=2844, p < 0.0001			

The number of people making status changes to the bug has a great impact on the overall time to resolve defects. For each additional person who changes the status of the bug, the expected time to resolve the defect is expected to slightly more than double. The final predictor variable, STC, also has a negative coefficient, indicating that higher levels of the team's overall socio-technical congruence at the time of the software defect are associated with shorter times to resolve software defects. In this data the effect is quite large, indicating that a team with an STC of 0.5 could expect to resolve defects in approximately 1/4th the time of a team with an STC of 0.

Unfortunately, the overall model explains only a very small amount of the overall variance in the time to resolve software defects in the community. It was, however, expected that the overall variance would be greater and explanatory power less than analysis on commercial products because of the heavily volunteer nature of GNOME. Common control variables for the productivity of software engineers, such as tenure and education, were not available for the community, nor are they likely to be easily accessible without a survey of the developers, which previously have found only mixed amounts of success in Open Source communities[19].

4. Evaluating Decay of Networks in Socio-Technical Congruence

After validating that there was sufficient data present from the projects to assess socio-technical congruence, the decay metric was introduced to the project networks. For each project the decay level was varied between 0.80 (high decay) and 1.00 (no decay) in steps of 0.05. The STC of each project in the community was calculated at each of the time periods, producing one curve for each value of STC.

An example of such a curve can be seen in Figure 1. This curve shows the results for the "Beagle" project, a desktop search engine similar to Google Desktop Search written in the object oriented language C#. The project has consistently enjoyed commercial support and during the 24 months of observation had a stable core project development team. Examinations of project mailing list archives shows that while new versions of the software were released frequently, there was never an effort to significantly re-architect the project.

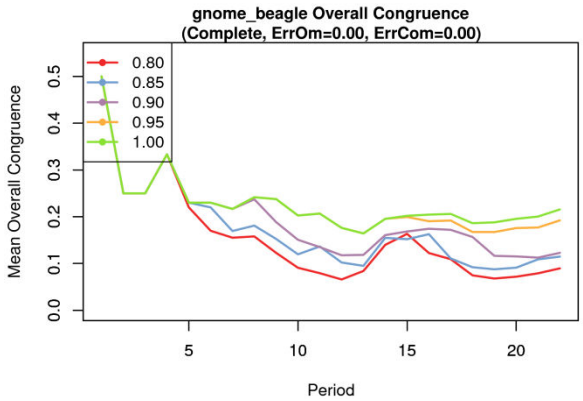

Figure 1: STC curves of the "Beagle" project. As the decay is increased, the overall STC of the project decreases.

This result is in marked contrast to the curves shown in Figure 2, which depict the STC of the Rhythmbox project, a music and media player for GNOME. In contrast to Beagle, Rhythmbox has much less commercial support. While there is some financial and development support from RedHat, and developers from other commercial firms have contributed code to the project, the project remains largely a volunteer project. Furthermore, the leadership of the project has changed hands, this change lead to a redesign of many of the internal components of the software partially as a result of changes in the underlying platform RhythmBox was built on and partly because of lost knowledge in the project leadership transition.

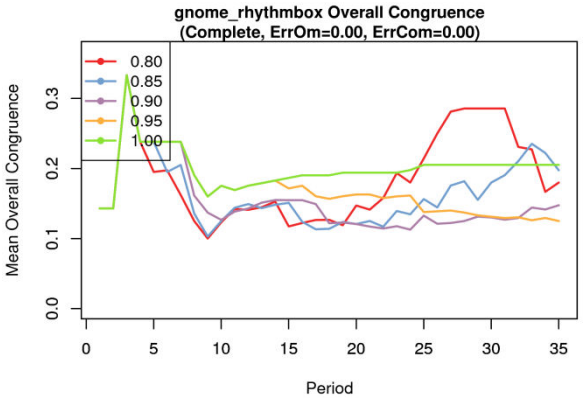

Figure 2: STC curves of the "RhythmBox" project. As decay is increased, the resulting change in STC is much more unpredictable.

Within Beagle, an increase in the decay of the project led to decreased overall socio-technical congruence. This is possible when the decay in actual coordination is faster than the decay in coordination requirements. For such a drop to occur the dependencies between tasks in the network (files in the project) must have been renewed and updated more frequently than the communication within the project was updated. From a software development perspective this means that these dependencies persisted throughout the project, while the communication to address those dependencies occurred earlier in the project and was not renewed later in the project.

Such a situation is the hallmark of a fairly static team, once individuals understand many of the dependencies, there is less of need to continually discuss the dependencies between files in the community.

Rhythmbox, on the other hand, exhibits largely decreasing values of socio-technical congruence with higher levels of decay until about half way through the observations, at which point the highest decay curve

Table 3: Summary statistics of STC at different decay levels

Variable	Min	Max	Mean	Skew	Kurtosis
1.00 (no decay)	0.00	0.750	0.1028	-0.067	1.287
0.95	0.00	0.750	0.0935	-0.048	1.406
0.90	0.00	0.750	0.0822	0.059	1.629
0.85	0.00	0.750	0.0792	0.310	1.864
0.80	0.00	0.750	0.0739	0.413	1.995

begins to increase. Each of the other decay curves then increases at a point slightly later and to a slightly lesser degree as the decay becomes less. This is to be expected as the higher decay models put less emphasis on more current communication and more emphasis on communication that occurred long in the past.

5. Evaluation of different decay levels

To examine the overall effect of increasing decay on communications and requirements in STC, aggregate statistics were collected for the different levels of STC across all projects. The results can be seen in Table 3. Of note is that the maximum is consistent across all projects because it occurs at the first time period for a project. The mean gradually declines indicating that more projects behave like Beagle than like Rhythmbox, and that in general applying a decay factor will result in decreased levels of STC. The data gradually moves from a negative skew to a positive skew indicating that increasing the decay within STC pushes the metric lower, however, high scores may still exist. The kurtosis is also monotonically increasing, indicating that as decay increases the scores become slightly more uniform and the variance comes from extreme points.

The same regression model was again used changing the STC predictor with each of the different decay levels. The results can be seen in Table 4 through Table 7.

Table 4: Results of STC regression with 0.95 decay

Variable	Estimate	Std Error	P-Value
Intercept	1.8613	0.1461	<.0001
numDevs	0.1073	0.1043	0.304
comments	-0.0136	0.0154	0.374
deltaPeople	1.1722	0.0675	<.0001
STCDecay0.95	-5.9549	0.6648	<.0001
R^2=0.1676, DF=2844, p < 0.0001			

Table 5: Results of STC regression with 0.90 decay

Variable	Estimate	Std Error	P-Value
Intercept	1.7709	0.1467	<.0001
numDevs	0.0943	0.1048	0.368
comments	-0.0158	0.0154	0.308
deltaPeople	1.1780	0.0678	<.0001
STCDecay0.90	-5.4628	0.7511	<.0001
R^2=0.1598, DF=2844, p < 0.0001			

Table 6: Results of STC regression with 0.85 decay

Variable	Estimate	Std Error	P-Value
Intercept	1.6095	0.1466	<.0001
numDevs	0.0785	0.1053	0.456
comments	-0.0172	0.0155	0.269
deltaPeople	1.8265	0.0682	<.0001
STCDecay0.85	-3.3945	0.7538	<.0001
R^2=0.1502, DF=2844, p < 0.0001			

Table 7: Results of STC regression with 0.80 decay

Variable	Estimate	Std Error	P-Value
Intercept	1.6116	0.1470	<.0001
numDevs	0.0738	0.1053	0.484
comments	-0.0173	0.0155	0.267
deltaPeople	1.1825	0.0682	<.0001
STCDecay0.80	-3.5682	0.7992	<.0001
R^2=0.1501, DF=2844, p < 0.0001			

When measured by explanatory power of the model, utilizing a decay of 0.95 or 0.90 both provide slightly better results than no decay. As the decay increased beyond 0.90 the overall explanatory power of the model decreased as the variance in STC decreased and the variance was increasingly a result of a few outliers. This indicates that introducing a small decay factor to STC may prove beneficial to tools and implies that

with a 0.95 decay factor, the strength of older ties is only 54% as strong as with no decay factor.

The selection of such a decay factor, is not entirely straightforward. While for the entire community a decay factor 0.95 produces the best results, this is not constant across all projects. For the Rhythmbox project, which experienced a period of leadership change and re-architecting of the project code, increasing the amount of decay continues to increase explanatory power of the model up to and including a decay factor of 0.80. The Beagle project, on the other hand, shows the best results when there is no decay factor applied.

6. Discussion

This paper has proposed the incorporation of a decay factor into the socio-technical congruence metric and evaluated the modification on projects within a large Open Source community. It was found that across the community the incorporation of such a decay metric increases the overall explanatory power of the model to a small degree. However, as a whole increasing the decay factor beyond 0.95 decreases the power.

We note that the optimal decay level varies across projects. Projects that have a stable architecture and rarely change will likely not need to apply a decay metric, while projects that undergo a large scale architectural change will benefit from a much higher amount of decay. The optimal solution for the evaluation of STC after an architectural change may utilize decay as one of many ways to alter the metric. Additional changes may include giving heavier weight to periods immediately after the new architecture is begun or removing all influence from periods before the rearchitecture process was started.

This finding of implementing a decay factor is most useful when designing tools that assess the degree to which team are communicating and fulfilling their coordination dependencies, such as Tesseract by Sarma et. al. [20] In particular, the application of decay seems particularly well suited for monitoring of Open Source projects or other projects where the monitoring and analyzing party may not be aware of all of the decisions being made with regards to project structure.

7. Acknowledgements

This work was supported in part by the National Science Foundation (IIS-0414698), the IGERT training program in CASOS (NSF,DGE-9972762), the Office of Naval Research under Dynamic Network Analysis program (N00014-02-1-0973, the Air Force Office of Sponsored Research (MURI: Cultural Modeling of the Adversary, 600322) for research in the area of dynamic network analysis. Additional support was provided by CASOS - the center for Computational Analysis of Social and Organizational Systems at Carnegie Mellon University. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation, the Office of Naval Research, The Army Research Lab (CTA: 20002504), or the Army Research Institute (W91WAW07C0063).

8. References

[1] M. Cataldo, P. Wagstrom, J. Herbsleb, and K. Carley, "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools," *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, Banff, Alberta, Canada: ACM Press, 2006, pp. 353-362.

[2] 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," *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, Kaiserslautern, Germany: ACM, 2008, pp. 2-11.

[3] J. March and H. Simon, *Organizations*, New York, NY: Wiley, 1958.

[4] J.R. Galbraith, *Designing Complex Organizations*, Addison Wesley, 1973.

[5] Y. Yamauchi, M. Yokozawa, T. Shinohara, and T. Ishida, "Collaboration with Lean Media: how open-source software succeeds," *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, Philadelphia, Pennsylvania, United States: ACM, 2000, pp. 329-338.

[6] R.M. Henderson and K.B. Clark, "Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms ," *Administrative Science Quarterly*, vol. 35, Mar. 1990, pp. 9-30.

[7] H. Mintzberg, *The Structuring of Organizations*, Prentice Hall, 1979.

[8] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri, "Challenges in software evolution," *Principles of Software Evolution, Eighth International Workshop on*, 2005, pp. 13-22.

[9] R.C. Seacord, D. Plakosh, and G.A. Lewis, *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*, Addison-Wesley Professional, 2003.

[10] K. Carley and D. Krackhardt, "A PCANS model of structure in organization," *1998 International Symposium on Command and Control Research*, 1998.

[11] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional, 1999.

[12] T. Mens and T. Tourwe, "A survey of software refactoring," *Software Engineering, IEEE Transactions on*, vol. 30, 2004, pp. 126-139.

[13] D. German, "Software Engineering Practices in the GNOME Project," *Perspectives on Free and Open Source Software*, J. Feller, B. Fitzgerald, S.A. Hissam, K.R. Lakhani, and M. Cusumano, eds., MIT Press, 2005, pp. 211-226.

[14] S. Koch and G. Schneider, "Effort, co-operation and co-ordination in an open source software project: GNOME," *Information Systems Journal*, vol. 12, Jan. 2002, pp. 27-42.

[15] D. German, "The GNOME project: a case study of open source, global software development," *Software Process: Improvement and Practice*, vol. 8, Sep. 2004, pp. 201-215.

[16] T. Halloran and W. Scherlis, "High Quality and Open Source Practices," *2nd Workshop on Open Source Software Engineering*, Orlando, Florida: 2002.

[17] Canonical, Ltd, "Launchpad."

[18] H. Gall, K. Hajek, and M. Jazayeri, "Detection of Logical Coupling Based on Product Release History," *14th IEEE International Conference on Software Maintenance*, IEEE Press, 1998.

[19] R.A. Ghosh, R. Glott, B. Krieger, and G. Robles, *Free/Libre and Open Source Software: Survey and Study*, International Institute of Infonomics University of Maastricht, The Netherlands, 2002.

[20] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development," *Proceedings of the 2009 International Conference on Software Engineering*, Vancouver, BC: 2009.