RC23856 (W0601-136) January 27, 2006
Computer Science

IBM Research Report

On Dependency Changes in Collaborative Software Development

Cleidson de Souza[1,2], Gloria Mark[2], David Redmiles[2], Li-Te Cheng[3], John Patterson[3], David Millen[3]

[1]Departamento de Informática
Universidade Federal do Pará
Belém - PA - Brasil

[2]School of Information and Computer Science
University of California, Irvine
Irvine, CA, USA

[3]IBM Research Division
One Rogers Street
Cambridge, MA 02142, USA

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

On Dependency Changes in Collaborative Software Development

Cleidson de Souza[1,2]
Li-Te Cheng[3]
[1]*Departamento de Informática*
Universidade Federal do Pará
Belém – PA – Brasil
cbrs@ufpa.br

Gloria Mark[2]
John Patterson[3]
[2]*School of Information and Computer Science*
University of California, Irvine
Irvine – CA - USA
{gmark,redmiles}@ics.uci.edu

David Redmiles[2]
David Millen[3]
[3]*Collaborative User Experience Group*
IBM Watson Research Center
Cambridge – MA - USA
{lite_cheng,john_patterson,david_r_millen}@us.ibm.com

Abstract

In this paper we investigate the phenomena of dependency changes - changes to software systems caused by other changes. Dependency changes are one measure of the level of interdependency in a software module, therefore a good approximation to the study of dependency management in software development. Survey responses from 148 software developers indicate that the frequency of occurrence of dependency changes is negatively correlated with project duration, configuration management tool usage, and software developers' experience with the programming language and with the role they play. The data also shows that those who communicate more often are less likely to consider dependency changes to different files as a problematic situation. Nevertheless, this does not hold for changes to the same file, what suggests that this situation is more difficult to coordinate. This fact indicates the need for software development tools to address this aspect. Additional implications for software development tools are discussed.

Keywords*: Dependency changes, software changes, empirical studies, software maintenance.*

1. Introduction

Any software system undergoes change at all stages of its life cycle. That is, changes are a natural part of any software. There are several reasons for making these changes, including, new and evolving requirements, bug fixes, changes in the environment, refactoring, etc. However, Whitgift points out that *"(…) change is hard to manage because items depend upon each other. An apparently minor change to one element may propagate to items which depend upon it, directly or indirectly, so that consequential changes are needed throughout the system"* [22]. For example, changes to requirements often lead to changes in the source-code to be written, changes in the source-code lead to changes in the test cases to be performed, and so forth.

Software engineering has already identified the need to manage and use software changes. Indeed, one can find several approaches dealing with and using software changes, such as visualization [5], prediction of risk [13], extra-time needed in distributed settings [8, 9], and so on. However, to the best of our knowledge, there is no empirical work studying the propagation of changes in a software development project. By propagation we mean how often changes cause additional changes in the software. That is, we are interested in studying changes in the software (hereafter called *dependency changes*) caused by other different changes. As Whitgift [22] points out, the reason for those dependency changes is the interdependencies that exist among the software components.

Despite the several design techniques (such as information hiding [16]), informal approaches [6] [4], and software development tools (like configuration management systems) proposed to minimize problems with dependencies, this is still an open problem in software engineering. To fill this gap, we conducted a thirty-four item survey in a large software development company during the summer of 2003. Professional software engineers were asked questions about their teams, projects, and individual characteristics, as well as demographics. We wanted to find information about *factors* that influence the occurrence of dependency changes in the everyday work of software engineers. More importantly, we wanted to understand the *consequences* of dependency changes (such as the amount of extra-time and effort necessary do handle them), and how problematic they perceived these changes. Empirical evidence about this phenomena is important since recent research prototypes (e.g., Night Watch [15], and Palantír [20]) are based on the assumption that situations involving dependency changes are not infrequent and therefore do not need computational support. With this study, we plan to col-

lect empirical data about the phenomena of dependency changes, and therefore present empirical evidence either supporting or opposing this assumption.

Our results suggest that the following factors correlate negatively with the frequency of dependency changes: project duration, configuration management usage, software engineer's experience in his current role, and software engineer's experience in the programming language used. On the other hand, we found out that the frequency of communication among software developers correlate negatively with the level of problem they see in these changes. That is, software developers who communicate more often with their colleagues are more likely to perceive dependency changes as non-problematic.

The rest of the paper is organized as follows. The next section defines concepts that will be used throughout the rest of the paper. Then, section 3 presents the research questions that we tried to answer in this paper. The following section discusses the methods that we used in this paper. After that, Section 5 describes the study results. Section 6 presents our discussion about the data that we collected. Finally, conclusions and ideas for future work are presented.

2. Background

The purpose of our web-based survey is to investigate possible factors that influence the occurrence of dependency changes in the software development process as well as the consequences of these changes. This section describes in details the concept of dependency changes and the hypotheses that we developed, with their rationale, to study the dependency changes phenomena.

2.1. Dependency Changes: Naming Convention

As explained before, because of the several interdependencies in any software system, changes in one part of the source-code may propagate to other parts of the source-code that depend upon it, so that additional changes in the source code are necessary [22]. The first type of changes, hereafter called "original changes", describe changes performed by a software engineer necessary for a variety of purposes, such as bug-fixing, enhancements, etc. Meanwhile, the changes required *because of* these original changes are called "dependency changes" or "impact changes". It is important to mention that we are exclusively interested in program dependencies [19], i.e., in dependencies among parts of the source-code.

Note that both the original and the dependency changes occur in files containing the source-code. If both changes occur in the same file, this means that original changes in one file required that dependency changes were performed in the same file. This situation from now on will be called same file situation (condensed SF). Differently, if original changes occur in one file and the dependency changes are necessary in one more different files, then the situation is called different file situation and abbreviated DF.

2.2. Hypotheses

One of the first factors that we wanted to find out about its influence on the occurrence of dependency changes was project duration, i.e., how long the project has been going on. Duration is important because it is long-recognized that the architecture of the software erodes as times passes [18], and when this happens, it is more difficult to enforce the architectural constraints of the software. This is problematic because often architectural styles lead to well-organized and manageable interdependencies. Therefore, when software erodes, dependency changes are more likely to occur. This suggests that:

H_1 *- The duration of the project does influence the occurrence of conflicting changes such that software engineers working in older projects will encounter more dependency changes.*

Whenever one talks about changes in software development, it is impossible not to mention configuration management systems (CM). Indeed, CM is a discipline about "controlling change: assessing the impact of a change before it is made, identifying and managing the multiple versions of items which a change generates, rebuilding derived elements after source elements are changed, and keeping track of all changes that are made to a system" [22]. Furthermore, CM tools are one of the most mature and adopted technologies for software development being used for several years in software development projects. However, the full potential of CM tools is not immediately leveraged because of the several advanced features such as triggers, "*winking in*" techniques to reduce compilation time, labeling and branching strategies. Learning these features is a long and time-consuming process requiring dedicated software engineers. In other words, despite the fact that CM tools are a successful example of software development technology, some time is required to leverage their full potential. Therefore, our next hypothesis is:

H_2 *- Experience using the CM tool does influence the occurrence of dependency changes, so that, projects that adopt CM tools for a longer time are expected to face less dependency changes.*

Distribution of the development team has been proved

to be a factor that delays the software development process because of the fewer opportunities for informal communication that distributed software developers have compared to collocated ones [10]. Previous research [12] has also shown that these spontaneous conversations are essential for the coordination of software development teams as well as other forms of group work. This suggests that:

H_3 *- The frequency of informal communication between the original changes' authors and the dependency changes' authors influence how dependency changes are perceived. Dependency changes are perceived as less problematic when there is more frequent communication among the software engineers.*

Furthermore, other studies have shown that information in distributed groups is not shared equally among remote team members and that remote sites are often excluded from important decisions [7]. Taking this result, we expect distributed software developers to perceive dependency changes as more problematic when compared to collocated developers. In other words:

H_4 *- The distribution of the software development team does influence how dependency changes are perceived. More distributed teams perceive dependency changes as more problematic than collocated teams.*

Furthermore, Grinter and colleagues [7] suggest that software development organizations might choose different models to divide work across distributed sites to minimize coordination needs. The same idea is suggested by Olson and Teasley [14], who found out that over time, the nature of the work being performed by the members of the team changed so that, work at each site became less dependent from work performed at other locations. This means that:

H_5 *- The distribution of the software development team does influence the occurrence of dependency changes. Distributed teams are more likely to face dependency changes than collocated teams.*

As important as the distribution of the software developers involved in the project is the *number* of developers. In this case, we expect to find more dependency changes in larger teams, since the coordination of a larger number of software developers is more difficult. That said, the next hypothesis is:

H_6 *- The number of developers in the project does increase the frequency of dependency changes, so that larger teams are more likely to face them.*

In addition to the number of software developers involved, we argue that a software developer's experience might influence the occurrence of dependency changes. Developers' experience with software development should increase their familiarity with dependency changes and with the problems they bring. Consequently, these software developers are more likely to adopt formal and informal work practices to avoid them. This means that, the next hypothesis is:

H_7 *- The experience that one software engineer has playing a particular role does influence the occurrence of dependency changes. More experienced software engineers will face less dependency changes.*

As mentioned before, software engineering research has been trying to minimize dependency problems. One of the most important principles learned in the 70's to facilitate that is called information hiding [16]. Indeed, several programming languages implement technical constructs to support information hiding. For example, private methods in classes are one way of hiding information from other classes, so that they can not use them. So, one might argue that software engineers with more experience in a particular programming language are more likely to be aware of and use particular constructs that support information hiding, therefore they are less likely to face dependency changes. In other words:

H_8 *- The experience that one software engineer has using the programming language used in the project does influence the occurrence of dependency changes. More experienced software engineers will face less dependency changes.*

The next section describes the methods and the setting where information was collected to test these hypotheses. The results of testing the above set of hypothesis are described in the section 4.

3. Methods

3.1. Target Population and Sampling Methods

Our results draw on survey data collected in a large software development company named HAL (a pseudonym). HAL is one of the largest software development companies with products ranging from operating systems, to software development tools, including e-business and tailored applications. In August 2003, 148 employees located all around the world were invited to complete a web-based questionnaire. Respondents were asked at the

beginning of the survey to answer the questionnaire *only* if they contribute with source-code to a project by either using code-generation tools or by manually writing source code. Therefore, our target population was defined by *excluding* software engineers who do not fit this criterion.

Invitations to fill the survey were sent through broadcast messages using a community-IM toolset described in [11] that enable instant access to communities of interest on a number of levels of engagement. This toolset, which has a large user base in the HAL development organization, allows the creation of open and closed communities about different subjects ranging from software development to British soccer. A user might choose to be a member of different communities at the same time, and, indeed, this is a very typical situation. The communities were the surveys were broadcasted focus on software development. Examples include communities about programming languages, users of specific technologies for software development, software development methods and specific software development projects. Invitations were sent for a total of 41 different communities either once or three times in an interval of fourteen days.

Invitation messages were broadcasted in a community appearing on users' screens for about 10 seconds. Due to limitations of the toolset we were using there is no way of knowing how many users received each invitation. *As a consequence, the answer rate of the survey is unknown.* Respondents were free to fill the survey, therefore implying a self-selection bias. Based on these factors, we make no claim about the representativeness of our results.

3.2. Questionnaire Design

In order to understand the phenomena of dependency changes, we created a web-based survey that consisted of a total of 34 items covering demographics (gender, age, education, employment status, etc), project information (such as duration, phase in the software development process, and configuration management tool usage), team information (e.g., number of members and their locations); and software engineer's individual characteristics regarding their role in the project, how long they have been playing this role, and their overall experience in software development. These questions aimed to identify factors that might influence the occurrence of dependency changes.

The second part of our research questions concerns the *consequences* of the dependency changes in the everyday work of software engineers. To achieve this goal, we included questions in the survey that asked how often they experience this situation, how much time and effort they spend performing these dependency changes, and finally, how problematic they think this situation is. Those questions were only answered by respondents who previously had answered that they had faced dependency changes in the project.

The questionnaire was designed in such a way that the questions about the frequency and the consequences of dependency changes were asked twice: in the first time, we asked about dependency changes in the same file situation, while in the second time, the same set of questions was asked regarding dependency changes in a different files situation. The rationale for adopting this approach is described in the following section.

3.3. Questionnaire Rationale

We divided our questionnaire in two conditions: the same file and different files conditions. Furthermore, because we wanted to collect independent information about each situation. The reason for that is our long-term goal of improving Jazz, a collaborative application development environment being constructed at IBM Research [1, 2]. Among other features, Jazz provides resource-centered awareness, i.e. the ability to indicate what other developers are doing with their local copies of the files (e.g. indicating that a file is currently in focus and being edited at this very moment, or that a file has been locally saved but not checked back into the code repository) [1]. The idea is to provide the developer with the same kind of peripheral awareness of the activities of others on the team as would be available if the team was all working in close proximity.

During the design of Jazz, we conducted 14 interviews with professional software developers at HAL using storyboards with mock screenshots (which we had used to plan our design earlier) to assess potential problems. As a result, interviewees commented that one possible use of resource-centered awareness was to support parallel development (i.e., more than one developer changing the same file) and to monitor files that one developer is dependent on. Therefore, we decided to conduct a survey to collect empirical data about the dependency changes. By dividing the questionnaire between same file and different files conditions, we hoped to be able to identify the most common and problematic situation, and therefore, the one which would drive our efforts in future versions of Jazz. It is important to mention that the phenomena of parallel development has recently received attention by the research community (see [15, 20]), but again empirical data about this phenomena is limited to case studies therefore not adequate to be generalized [17].

4. Empirical Study Results

In this section, we describe the overall results of this survey. Overall, one hundred forty-eight software engi-

neers completed the questionnaire. Of those, 73 software engineers have experienced dependency changes because of changes in the same file, while 80 have experience dependency changes in this project caused by changes in different files. Note that, respondents were not forced to answer the questions; therefore, some of the descriptive statistics presented below might not contain the total answers for each category.

We have organized our results in three major sections. Initially, we describe overall descriptive statistics of the data. After that, we look at factors that influence the occurrence of dependency changes, such as project duration, team size, team distribution, CM usage, etc. Finally, the last section describes information that we collected about the respondents who have already faced dependency changes in the project.

4.1. Descriptive Statistics

Of the respondents, 53 were software architects, 73 were programmers, and 7 were members of the quality assurance team. These developers' experience within the project would vary as follow: 49 developers have been working in this project for more than 2 years, 27 for less than two years but over a year, 31 engineers reported that have been working in this project between 1 year and 6 months, while 21 reported to be working anytime between 6 and months, and finally, 19 software engineers have been working for less than 3 months in the project. Regarding the duration of the project itself, 77 developers reported that the project has been going on for more than 2 years, 25 for less than two years but over a year, 20 reported that the project duration is between 1 year and 6 months, while 19 reported the duration as being anytime between 6 and months, and finally, 7 projects have been going for less than 3 months.

Furthermore, most projects were in the implementation (74) or maintenance (34) phases. Other phases include planning (2), requirements (4) and design (9). A total of 23 software engineers reported that their projects were in a different phase ("Other"). Size of the projects varied a lot across respondents: there were 9 projects with more than 200 software engineers; 5 projects with less than 200 and 100 or more software engineers; 8 projects with less than 100 and 50 or more software engineers; 16 projects with less than 50 and 25 or more software engineers; 34 projects with less than 25 and 10 or more software engineers; 22 projects with less than 10 and 5 or more software engineers; and finally, 54 projects with less than 5 software engineers. Regarding the distribution, 57 projects were classified as highly distributed (members of the team are spread across 3 or more different office locations), 45 projects as partially collocated/distributed (members of the team are spread across 2 different office locations), and a total of 40 projects as completely collocated (members of the team are located in the same office location).

Finally, it is important to collect information about the CM usage. In this case, we found out that 36 projects do not use CM tools while 105 projects use it. Of those 105, 63 have been using it for more than 2 years, 13 for more than 1 year but less than 2 years, 14 for more than 6 months but less than 1 year, 10 for more than 3 but less than 6 months, and, 5 projects use CM tools for less than 3 months.

4.2. Consequences of Dependency Changes

In this section, we describe the descriptive statistics about the second part of the survey, which collected information about the consequences of dependency changes.

Table 1 summarizes our results. One might notice that according to the respondents, dependency changes do not seem to happen very often. Similarly, time and effort spent dealing with the dependency changes did not seem to be very large. Effort was defined as the amount of physical and mental energy and the level of concentration that goes into doing the work for the dependency change.

		Same File	**Different Files**
Frequency	Rarely	91.8 %	83.8 %
	Often	8.2 %	16.2 %
	N	73	80
Time	**Reasonable**	76.7%	73.1 %
	Huge	23.3 %	26.9 %
	N	73	78
Effort	**Reasonable**	81.1 %	81.3 %
	Huge	18.9 %	18.8 %
	N	74	80
Problem	**Non-Problematic**	74.3 %	81.3 %
	Problematic	25.7 %	18.8 %
	N	74	80
Location	**Collocated**	69 %	67.1 %
	Distributed	31 %	32.9 %
	N	71	76
Communication Frequency	**Rarely**	6.8 %	7.4 %
	Often	93.2 %	92.6 %
	N	74	76

Table 1: Descriptive Statistics

A total of 73 (50.3% out of the 148) software engineers have experienced dependency changes because of the changes in the same file, while 80 developers (55.4% out of the 148) have experienced dependency changes because of changes in different files. Combining the data

from these two questions, we found out that about 66.21% of all 148 respondents have already experienced a situation involving dependency changes in their current project. About 22.29% of the respondents have not faced this situation, and finally, 11.48% was not sure about it. Furthermore, we identified that an average of 3.30 (standard deviation of 5.54) other software engineers might be modify the same file over the duration of the project. When this comes to different files, the average number of software engineers involved is 4.92 (standard deviation of 10.48).

In addition to the information about how software engineers deal with dependency changes, we collected information about the location of the original changes' authors and the frequency of informal communication that these software engineers have with the dependency changes' authors. This information was especially useful to test hypothesis H_3, H_4 and H_5.

4.3. Dependency Changes and Factors

In order to identify possible factors that influence the occurrence of dependency changes, we collected three different types of information, loosely classified as follows:

- Project information: duration and configuration management tool usage.
- Team information, including number of software engineers involved, frequency of communication among them, and their locations (distribution); and;
- Individual information, regarding the experience that the software engineer has playing a role, and his / her programming language experience.

The results of our test will be presented below according to these categories.

4.3.1. Project Information

H_1: Project Duration

Regarding the project duration, we found statistically significant evidence that project duration was positively correlated with frequency of dependency changes (same file, r=.21, p<0.05, N=73; different files, r=.22, p<0.05, N=80). This means that older projects usually face less dependency changes than younger projects in both same file and different files conditions. In other words, we found evidence supporting H_1.

H_2: Usage of CM tool

In addition to project duration, our survey included a question about the project usage of configuration management tools. The idea is that experience using configuration management tools would correlate with a reduced occurrence of dependency changes (H_2). We tested that and indeed, we found statistically significant evidence that CM usage experience was positively correlated with frequency of dependency changes (same file, r=.44, p<0.01, N=55; different files, r=.394, p<0.01, N=63). This means that projects that use CM for longer time are more likely to face less dependency changes.

4.3.2. Team Information

We collected one type of information about the teams working in the project, which is the total number of team members. This information was collected for all respondents. In addition to that, we collected more specific information from software engineers who had faced dependency changes in either the same file or different files situations. This information consists of the location of and frequency of informal communication with the other software engineers who caused these dependency changes, i.e., the authors of the original changes in the software.

H_3: Frequency of communication and H_4: Distribution of software developers

As described in section 2, frequency of communication and distribution of team members might affect the opportunities for interaction and the flow of information, which makes coordination more difficult. Therefore H_3 and H_4 suggest that the dependency changes will be perceived as less problematic for developers who communicate often among themselves and are collocated, respectively. After performing correlation tests with the appropriate variables, we found out that, for different files only, there is a positive correlation between how a software developer rates the situation and the frequency of informal communication with the other software developer involved: (same file, r=.14, p<.11, N=74; different files, r=.25, p<.01, N=79). We also found that there is a positive correlation between the rating of the dependency change situation and the location of the developer involved. In this case, there is a trend in the results in the different files situation (r=.16, p<.07, N=78), but no empirical evidence for the same file situation (r=.05, p<.30, N=74).

H_5: Distribution of software developers

Regarding the distribution of the software developers and the frequency of dependency changes (hypothesis H_5), our results (same file, r=.00, p<.49, N=73; different files, r=.01, p<.44, N=78) suggest that there is no correlation between these variables, i.e., distributed software engineers do not face more dependency changes than collocated ones. Thus, our data does not support H_5.

H_6: The number of developers in the project

H_6 was tested by calculating the Spearman's rho correlation coefficient between the total number of software engineers in the project and the frequency of dependency changes, providing the following results: (same file, r=.00, p<.49, N=73; different files, r=-.14, p<.10, N=80). Furthermore, we also calculated the same coefficient between the number of software engineers likely to be modifying the same file or different files, therefore causing dependency changes. The results are the following: (same file, r=.00, p<.47, N=73; different files, r=.13, p<.12, N=80). Surprisingly, the results of both correlation tests provide no evidence supporting H_6.

4.3.3. Individual Information

H_7: Role Experience

Finally, we collected individual information about each respondent regarding its experience in the project playing a particular role. This lead to the hypothesis H_7 that argues that more experienced software engineers will face less dependency changes. In order to test the significance of the correlation between software developers' experience and the frequency of dependency changes, we performed a Spearman's rho correlation test, which provided the following results: (same file, r=.20, p<0.05, N=72; different files, r=.19, p<0.05, N=80). In both cases, the correlation coefficient is positive and there is some evidence (p<0.05) supporting H_7, which means that more experienced software engineers face dependency changes less often.

H_8: Programming Language Experience

We also collected information about software developers' experience with the programming language used in the project in order to test H_8. We tested the significance of the correlation between this experience and the frequency of dependency changes and found out that there is a trend (p<.07) suggesting that more experienced software engineers face dependency changes less often. Our results are the following: (same file, r=.17, p<.07, N=72; different files, r=.20, p<.05, N=80). Thus, our data supports our last hypothesis.

5. Model

In this section, we summarize in a qualitative model the factors that we identified which correlate with the frequency of dependency changes, namely: project duration, experience using the CM tool, software developer's experience in the role, and software developers' experience with the programming language. This model also includes the factor that correlate with the how problematic dependency changes, which is the frequency of communication among software developers. In this case, these factors correlate only in the different files condition.

5.1. *Post-hoc* Analysis

As a *post-hoc* analysis, we tested for multicollinearity among the factors described above. Results are presented below:

Correlation tested	Significance
Project duration and Experience using the CM tool	R=.64,p<.01, N = 105
Project duration and Software developers' experience playing his current role	R=.59,p<.01, N = 147
Project duration and Software developers' experience with the programming language	R=.05,p<.59, N = 141
Frequency of Communication and Location (different file)	R=.37,p<.01, N = 79

Table 2: Multicollinearity Tests

Despite the strong evidence of multicollinearity (p<.01), it is important to note that CM usage is not necessarily linked to project duration, since: (i) project duration might include other phases, that typically do not use CM tools (e.g., planning, requirements, and design); and (ii) even in projects in the implementation and design phase, CM tools might not be adopted during the whole project. Note that an analogous rationale can be applied to project duration and the software developers' experience in a role, i.e., one software developer might start working in a role in a project and change to another one during the course of the project.

Similarly, there is strong evidence of multicollinearity (p<.01) between frequency of communication and location of software developers in the different files condition. In this case, it is definitely reasonable to think that collocated co-workers have a higher frequency of work communication. Therefore, we will eliminate the factor distribution from the model. The resulting model is presented in Figure 1 below.

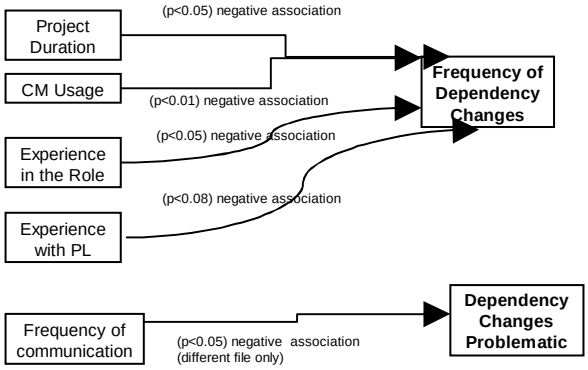

Figure 1: Dependency Changes Model

5.2. Frequency of Dependency Changes

The model above describes the relationships among the several factors that are correlated to the frequency of dependency changes and the problematic rating of these changes. The strength of the correlations is represented by their respective p-values in the model. Regarding the frequency, all factors are negatively correlated with it, which means that increments in the factor correlate with decrements with the frequency. So, for example, the longer the project duration, the lower the frequency of dependency changes. In this particular case, this result provides evidence against H_1. A possible explanation is that as time progresses, software development teams organize their work and the architecture of the software to minimize these situations.

Similarly, the usage of CM tools and its negative correlation with the frequency of dependency changes suggest that, given time, software development teams will use the CM tool in such a way that it will help them to organize their work. Indeed, CM tools are long-known for providing a good isolation, allowing developers to work in parallel without being affected by others' changes [3].

Our model also suggests that one important factor that minimizes the frequency of dependencies changes is the software developer's experience, both with the programming language used and within the role that he plays in the project. We speculate that, given time, software engineers will learn and adopt work practices to avoid problems with dependency changes, similarly to other informal work practices to manage interdependencies [4]. An interesting possibility for research is to identify, document, and encourage these practices.

5.3. Rating Problems with Dependency Changes

Regarding how the situations with dependency changes are rated according to their problematic, our model suggests that the frequency of communication among software developers is one important factor because the more frequencies developers communicate, more they will share useful information that facilitates their coordination. We hypothesize that one type of information is about dependencies changes, therefore it allows developers to prepare for those changes. Because software engineers are expecting these changes, they see this situation as less problematic.

Furthermore, while it is not surprising to find out that the frequency of communication among co-wreks correlate with the rating of the level of problem; it is surprising to find out that this only happens in the situation that involves different files. There is no significant correlation among these variables in the same file situation. This again suggests that dependency changes involving the same file are more problematic than when they involve different files. That is, it does not matter how frequent the communication among the involved developers or how far they are located, those situations are problematic by themselves.

6. Discussion and Implications

Our data provide evidence that dependency changes do not happen very often (maximum of once a month) in any situation: same file or different files. Furthermore, when these changes happen, they do not seem to be very harmful since the time (maximum of a day of work) and effort (a moderate amount of energy) spent to deal with them was rated by most software engineers as reasonable. Therefore, it is not surprising to find out that a most of them rated this overall situation as non-problematic: (SF=74.3%, DF=81.3%). On the other hand, it is interesting to note that dependency changes in the same file happen less often, require less time and effort than dependency changes in different files, but they are still rated as *more* problematic (SF = 25.7%, DF=18.8%). We now speculate that dependency changes in the same file are seen by software engineers as symptoms of larger coordination problems in the software development team. We hypothesize that coordination changes in different files are somewhat expected by software developers, since they are aware of the interdependencies among the several components of the software system [4]. However, coordination changes in the same file represent a situation where not all software changes necessary in a file were performed, meaning that there was ineffectiveness in performing the original changes in the file. Furthermore, these results also suggest that tools that facilitate the coordination of several developers working in the same file, such as Palantir [20] and Night Watch [15] seem to be very promising to facilitate software developers' work.

Finally, while there are several mechanisms that im-

plement the principle of information hiding (to deal with dependency management) in programming languages. These constructs are often more useful in situations involving different files, which means that dependency change management is more challenging

Based on previous work ([7], [21]) it is not surprising to find out that distribution does *not* affect the frequency of dependency changes. This result suggests that the organization of distributed software development teams seems to adopt efficient strategies to avoid the propagation of changes from one site to another, i.e., these strategies seem to be able to insulate software developers at each site so that they are not impacted by their coworkers. The same ides seems to hold for project size, i.e., software development projects seems to do a good design that insulates changes minimizing the number of people affected by changes. Even considering the number of developers likely to be able to change files that cause dependency changes, it is surprising to find out that there is no correlation between frequency of changes and number of software developers involved in the project. One possible explanation is that dependency changes are more or less located in parts of the code

Some limitations of our study are that we only sampled teams from one large software development multinational corporation. It is possible, but unlike, that different software teams in the corporation adopt similar approaches and therefore their data might have skewed our results. We would nevertheless expect our results of the difference between same file and different files conditions to generalize, since those are technical constructs that can not be avoided in any software system. Furthermore, we can not make any major inferences about the data that we collected because we administered the survey over the web. Therefore, this means that a selection bias might have arisen. Similarly, the communities selected for the broadcasting of survey invitations were not randomly selected.

7. Conclusions and Future Work

This paper presented the concept of dependency change as a technical construct that can be used to study interdependencies in software development. This paper also described the 34 questions of the web-based survey used. Professional software engineers were asked questions about their teams, projects, and individual characteristics, as well as demographics. We wanted to find information about *factors* that influence the occurrence of dependency changes in the everyday work of software engineers. We also wanted to understand the *consequences* of dependency changes (such as the amount of extra-time and effort necessary do handle them), and how problematic they perceived these changes. Our results suggest that the following factors correlate negatively with the frequency of dependency changes: project duration, configuration management usage, software engineer's experience in his current role, and software engineer's experience in the programming language used. On the other hand, we found out that the frequency of communication among software developers correlate negatively with the level of problem they see in these changes. That is, software developers who communicate more often with their colleagues are more likely to perceive these dependency changes as non-problematic because the software developer will already know about the dependency changes and its implication before it formally be notified about it.

8. References

[1] Cheng, L.-T., De Souza, C. R. B., et al., "Building Collaboration into IDEs. Edit -> Compile -> Run -> Debug ->Collaborate?," in *ACM Queue*, vol. 1, 2003, pp. 40-50.

[2] Cheng, L.-T., Huppfer, S., et al., "Jazz: a collaborative application development environment," ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications, pp. 102-103, Anaheim, CA, USA, 2003.

[3] Conradi, R. and Westfechtel, B., "Version Models for Software Configuration Management," *ACM Computing Surveys*, vol. 30, pp. 232-282, 1998.

[4] de Souza, C. R. B., Redmiles, D., et al., "Management of Interdependencies in Collaborative Software Development: A Field Study (to appear)," International Symposium on Empirical Software Engineering (ISESE'2003), Rome, Italy, 2003.

[5] Eick, S. G., Graves, T. L., et al., "Visualizing Software Changes," *Software Engineering*, vol. 28, pp. 396-412, 2002.

[6] Grinter, R., "Supporting Articulation Work Using Configuration Management Systems," *Computer Supported Cooperative Work*, vol. 5, pp. 447-465, 1996.

[7] Grinter, R., Herbsleb, J., et al., "The Geography of Coordination: Dealing with Distance in R&D Work," ACM Conference on Supporting Group Work (GROUP '99), Phoenix, AZ, 1999.

[8] Herbsleb, J. D., Mockus, A., et al., "Distance, Dependencies, and Delay in a Global Collaboration," ACM Conference on Computer-Supported Cooperative Work, pp. 319-328, Philadelphia, PA, 2000.

[9] Herbsleb, J. D., Mockus, A., et al., "An Empirical Study of Global Software Development: Distance and Speed," International Conference on Software Engineering, pp. 81-90, Toronto, Canada, 2001.

[10] Herbsleb, J. D. and Moitra, D., "Global software development," *IEEE Software*, vol. V18, pp. 16-20, 2001.

[11] Jania, F., "Broadcast Messaging: Messaging to the Masses," in *ACM Queue*, vol. 1, 2003.

[12] Kraut, R., Egido, C., et al., "Patterns of Contact and

Communication in Scientific Research Collaborations," in *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work*, J. Galegher, C. Egido, and R. Kraut, Eds.: Lawrence Erlbaum, 1990, pp. 149-172.
[13] Mockus, A. and Weiss, D. M., "Predicting Risk of Software Changes," *Bell Labs Technical Journal*, vol. 5, pp. 169-180, 2000.
[14] Olson, J. and Teasley, S., "Groupware in the Wild: Lessons Learned from a Year of Virtual Collocation," Computer Supported Cooperative Work, pp. 419-427, Boston, Massachusetts, USA, 1996.
[15] O'Reilly, C., Morrow, P., et al., "Improving Conflict Detection in Optimistic Concurrency Control Models," 11th International Workshop on Software Configuration Management (SCM-11), Portland, Oregon, 2003.
[16] Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15, pp. 1053-1058, 1972.
[17] Perry, D. E., and, H. P. S., et al., "Parallel Changes in Large Scale Software Development: An Observational Case Study," International Conference on Software Engineering, pp. 251-260, Kyoto, Japan, 1998.
[18] Perry, D. E. and Wolf, A. L., "Foundations for the Study of Software Architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, pp. 40-52, 1992.
[19] Podgurski, A. and Clarke, L. A., "The Implications of Program Dependencies for Software Testing, Debugging, and Maintenance," Symposium on Software Testing, Analysis, and Verification, pp. 168-178, 1989.
[20] Sarma, A., Noroozi, Z., et al., "Palantír: Raising Awareness among Configuration Management Workspaces," Twenty-fifth International Conference on Software Engineering, pp. 444-453, Portland, Oregon, 2003.
[21] Teasley, S., Covi, L., et al., "How Does Radical Collocation Help a Team Succeed?," Conference on Computer Supported Cooperative Work, pp. 339-346, Philadelphia, PA, USA, 2000.
[22] Whitgift, D., *Methods and Tools for Software Configuration Management*. Chichester, UK: Wiley & Sons, 1991.

A. Appendix

Tables A and B below describe how the data presented in Tables 1 and 2 was collapsed.

Frequency of Dependency Changes		**Time spent dealing with Dependency Changes**		**Effort spent dealing with Dependency Changes**	
Rarely	A few times a year Once a month More than once a month	**Reasonable amount of time**	Less than one hour About an hour Several hours Less than a day	**Reasonable Effort**	A limited amount of energy A moderate amount of energy
Often	Once per week More than once per week Once a day More than once a day	**Huge amount of time**	About a day Less than a week About a week Less than a month About a month	**Huge Effort**	Quite a bit of energy A great amount of energy An extraordinary amount of energy

Table A: Description of the data collapsing

How problematic it is to deal with dependency changes?		**Location of Software Developers**		**Frequency of Informal Communication**	
Non-Problematic	Not problematic at all Slightly problematic	**Distributed**	Different country Different city Different building	**Rarely**	Never A few times a year Once a month More than once a month
Problematic	Somewhat problematic Considerably problematic	**Collocated**	Same floor Immediately next to me	**Often**	Once per week More than once per week Once a day More than once a day

Table B: Description of the data collapsing