Analysis of Students' Learning for Efficient Task Assignment in a Distributed Scrum Setting

by

Prashant Chhabra
B.Tech., Maharshi Dayanand University, 2011

Project Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Analysis of Students' Learning for Efficient Task Assignment in a Distributed Scrum Setting

by

Prashant Chhabra
B.Tech., Maharshi Dayanand University, 2011

Supervisory Committee

Dr. Daniela Damian, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Daniela Damian, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

ABSTRACT

The Development of software across multiple sites called global software development [GSD] is the norm of industry. Various factors like monetary benefits, desire to tap into a pool of skilled workers and the proximity to customers, has led to its growth. However, GSD suffers from various challenges, with communication being the biggest. It is difficult to divide complex software into independent modules such that to minimize the communication requirements. Development of these interdependent modules takes place across different sites that are separated by different time zones, cultures and languages. These differences create difficulty in communication and collaboration which is essential for the success of GSD. Scrum, an agile methodology, has shown to mitigate some of these challenges by enforcing structured communication. This report presents a case study of students working in the development of mobile application in distributed scrum setting. The aim of this report is to see if students following scrum methodology learn task assignment to reduce cross team and cross-site dependencies.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I wish to thank my committee members who were more than generous with their expertise and precious time. A special thanks to Dr. Daniela Damian, my supervisor for her countless hours of reflecting, reading, encouraging, and most of all patience throughout the entire process. I thank Dr. Alex Thomo, my graduate committee member.

I would like to express a special word of thanks to my friends and family who tirelessly listened to my ideas and offered encouragement when it was most needed. I would like to thank Guy Evans, my fellow lab member.

Finally, I would like to thank Dr. Kelly Blinoce, Dr. Casper Lassenius, Jyoti Sheoran, Aminah Yussuf and Francis Harrison who assisted me with this project.

DEDICATION

I dedicate my report to my loving parents.

Chapter 1

Introduction

The Global Software Development [GSD] is the development of software across multiple sites. Various factors like: the cost benefit of software development in low-income countries, the proximity to customer, the desire to tap pool of skilled workers and the development of high-speed communication links has contributed to the growth of GSD [1]. Previous studies have shown the importance of effective communication in GSD [3][5]. However, effective communication is also one of the biggest challenges in GSD [12]. Different time zones and cultural differences alleviate the problem of ineffective communication. Paasivaara et al. [15] found that scrum can mitigate some of the challenges of GSD. Scrum is an agile approach for developing innovative products and services [17]. In scrum, development takes place in small intervals called iterations. It enforces specific workflow in agile methodology and promotes communication. Studies of teaching scrum in GSD have received positive feedback from students [19]-[16].

Software development takes place by dividing work into small portions called work items. Dependency arises when one work item is dependent on another work item. Dependencies in work items increase the need for communication and coordination [3]. Hence, less dependency of work items across teams and across sites is desirable as communication is a big challenge across teams and across sites [12]. This report describes a study of scrum as a methodology that enables students to learn effective task division and dependency management in globally distributed software development projects. This report presents the analysis from a subset of a bigger research project that describes the experience of teaching global software development using scrum practices [14].

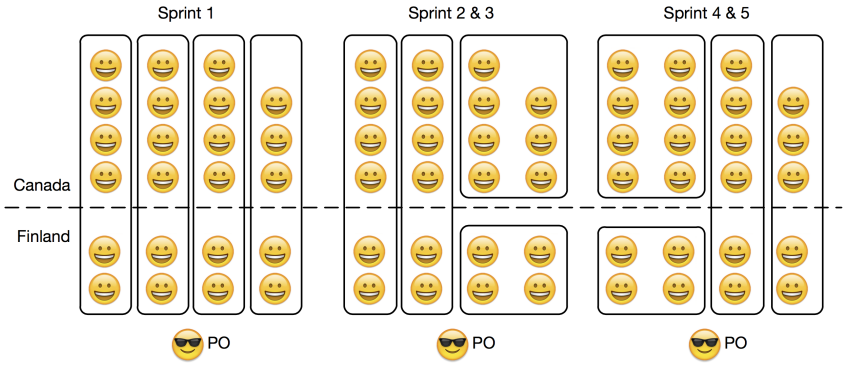

Figure 1.1: Study Design. This figure has been borrowed from Passivaara et al. paper [14].

1.1 Study Setup

The analysis presented here was conducted with data collected during a course on Global Software Development offered in the spring term of 2014. The course was jointly setup between the University of Victoria (UVIC), Canada and the Aalto University, Finland. Students of the class worked on developing a mobile client[1] for Agilefant[2] (a backlog management software). The course at UVIC included lectures, class discussions and blogging on research papers whereas the course at Aalto University was a capstone project. There were 15 Canadians and 8 Finnish students. The Finnish students started working in September 2013 and the Canadians joined in January 2014.

The Finnish professor along with the product owner and two students travelled to Canada to help the Canadian students become accustomed to the codebase, practices and tools. Hence, the first couple of sprints were used for training and were not included in the analysis. There were four sprints, each approximately two weeks long. Every sprint had 4 scrum teams with 4-6 members in each team. To give the experience of working in both co-located and distributed teams, the students were divided into two distributed teams, one local Canadian team and one local Finnish team in the first two sprints. For the next two sprints, local teams were split to make two distributed teams and the distributed teams were split to make one local team at each site. Each team had a scrum master who was responsible for enforcing scrum rules, principles and practices. The scrum master was changed every sprint. The

[1] https://play.google.com/store/apps/details?id=com.agilefant.mobile&hl=en
[2] http://agilefant.com/

product owner was situated in Finland.

Each sprint began with sprint planning in which the product owner presented a prioritized list of work items, called stories. Story assignment to teams was done on a voluntary basis. After gathering stories, the teams were given different rooms to perform sprint planning within the team. Each team breaks the stories into smaller work items called tasks, estimate story points and the time required to complete the stories and assign the stories to different team members. The product owner joined different teams using Google Hangouts[3] for clarification of the stories with the teams.

During a sprint, teams had two 15 minute weekly meetings. These meetings were similar to a daily standup of scrum in which every team member has to talk about 3 things:

1 What they did,

2 What they plan on doing next and

3 If they are stuck or blocked by someone's work.

Instead of daily meeting prescribed by scrum, two weekly meetings were conducted as the students were expected to work only 10-12 hours a week. While working on stories, the students were encouraged to contact the product owner for any clarifications. All of the code was hosted on GitHub (a web-based project hosting service for distributed revision control Git)[4]. The students followed the Git workflow[5], which included the submission of pull requests[6] on GitHub. Any student could review and merge the pull request. The students were allowed to defer any story to the next sprint if they are unable to complete it in the current sprint. Demos were conducted at the end of each sprint to show the work done by each team. This was followed by retrospectives to have a look at what worked, what didn't work and if any modifications are needed in the process. The whole process of demo, retrospectives and sprint planning took 2.5 hours. The most commonly used tools during the course were Google Hangouts (VoIP), GitHub (version-control), Flowdock[7] (IM) and Agilefant (backlog management).

[3]https://plus.google.com/hangouts
[4]https://github.com/
[5]https://help.github.com/articles/what-is-a-good-git-workflow/
[6]https://help.github.com/articles/creating-a-pull-request/
[7]https://www.flowdock.com/

1.2 Methodology

The aim of this project was to answer the research question: ***"Do students following the Scrum methodology learn to self assign stories in time to reduce cross team and cross site dependencies in a distributed software project?"*** The methodology being employed was a case study of students' work in the development of a mobile application called Mobilefant. Quantitative analysis was conducted on the data collected from GitHub and Agilefant. The data was used to calculate logical dependency between files, which was then mapped to users and stories to contrast dependencies across teams and across sites. If the dependencies across teams and across sites reduced over time, we can say that students learned to self-assign work items.

1.3 Contribution

The contribution of this project is the empirical evidence that shows that the students following scrum methodology learned efficient task assignment in a distributed setting. The results show effective task allocation in time to keep cross-team and cross-site dependencies in check which can be attributed to scrum. Hence, Scrum methodology in distributed setting is an effective way to teach GSD.

1.4 Report Outline

This chapter briefly introduced goal, motivation, methodology and course structure of case study. The remaining sections are organized as follows:

Chapter 2 mentions reasons for the growth of GSD, challenges of GSD and introduces scrum methodology. It also talks about the previous studies conducted on teaching GSD and ways of calculating dependencies in codebase.

Chapter 3 mentions in detail the methodology being employed, various levels of dependency and the procedure being employed for their mapping.

Chapter 4 presents results and contrasts co-located vs. distributed dependencies and inter-team vs. intra-team dependencies.

Chapter 5 discusses results to draw conclusions and mentions limitations of this study.

Chapter 2

Literature Review

It is quite common to see software development being distributed across multiple geographic locations [1], referred to as Global Software Engineering. Various reasons have led to the growth of GSD [1]: -

- *Difference in cost* - There is a huge disparity between wages of a software engineer from USA and a software engineer from Asia or South America with equivalent skills. Development of high-speed communication links has made it easier for companies to outsource software development to lower-cost economies.
- *Tapping the pool of skilled workers* - GSD makes it possible for companies to tap highly skilled developers irrespective of their location.
- *Proximity to Customer* - Sometimes companies wants to develop software closer to the location of their customers. This helps them to better understand customers and markets needs.

2.1 Challenges of Global Software Development

Despite many advantages that come with GSD, it also suffers from various challenges: such as the lack of overlapping work hours, different languages, different cultures, the lack of trust, the lack of frequent contact and the lack of coordination and communication [12]. Communication across different sites is considered to be one of the biggest challenges of GSD [12]. Previous studies have found that ineffective or lack of communication leads to software failures and broken builds [3] [5].

The dependency of work items require greater communication for the success of a project [3]. Modularization seems to be the solution to reduce dependencies [3]. However, it is difficult to modularize components in software development especially because requirements are known over time and keep on changing [3].

Different cultural backgrounds on different sites alleviate the problem of communication. Employees on different sites could have different training, education, native language and style. These differences make it difficult to initiate communication on different sites. Differences in language and style could make people appear vague or rude [12].

Lack of unplanned contact also hampers communication across sites [12]. If employees are on same site, they are more likely to engage in unplanned contact in hallway, lunch, coffee, etc. From previous study [10], we know that unplanned or informal communication is vital as it could help to solve various problems, which could otherwise turn out to be quite costly [10]. Different time zones also alleviate the problem of communication. Nonexistent overlap in working hours leads to the reliance on asynchronous communication like email [12].

Different cultural backgrounds, lack of unplanned contact and different time zones contribute to the lack of communication between sites, which can lead to lack of awareness across sites. Awareness is defined as an understanding of activities of others, which provides a context for one's own activities [5]. All these factors contribute to difficulty in clarification of simple questions and lack of knowledge about who is the expert for a specific module of software. [12].

2.2 Scrum

Scrum *is an agile approach for developing innovative products and services* [17]. According to scrum, work is done in small intervals of time called iteration. Each iteration (also called as sprint) starts with a product backlog, which is a list of items sorted according to the priority. Hence, any uncompleted work at end of sprint has lower priority than completed work. Scrum also enforces a specific workflow.

2.2.1 Scrum Roles

There are three roles in scrum - Product owner (PO), Scrum master and development team. The product owner has a leadership role. The PO is responsible for creating a

list of work items for every sprint and also to sort work items according to the priority. There should be a clear communication between the PO, the scrum master and the developers to convey the product vision and answer any clarification questions about work items. The Scrum master is responsible for enforcing scrum rules, principles and practices. The Scrum master also resolves team issues and makes improvements to the scrum. The Developers work on work items from the backlog. They can interact with the scrum master or the PO for clarifications.

2.2.2 Scrum Workflow

Each iteration starts with a product backlog, which is a list of work items sorted by their priority. The PO creates this list. In Sprint planning, the development team estimates the amount of time for every work item and decides which items can be completed in that sprint. Items are being worked upon in sprint execution. During sprint execution, daily standups are conducted to give an update about what every team member has completed since the last standup, what they plan to work next, and if they have any obstacles in completing their work. Next, Sprint demos are conducted to demo completed work items to the product owner. A sprint review is conducted to adapt the product according to feedback from team members. Retrospectives are conducted just before sprint completion to have a look at what worked, what didn't work and if any modifications are needed in the process. Scrum of scrums meeting is used to improve inter-team coordination especially when several teams are working on same project [17].

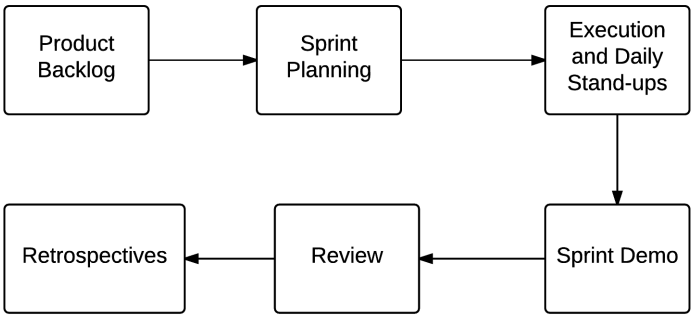

Figure 2.1: Workflow in Scrum Iterations

2.2.3 Scrum mitigates some problems of GSD

Agile development methods and especially scrum is gaining popularity [11][8]. Literature review of case studies show successful agile GSD for distributed teams [8]. Scrum has been shown to mitigate some of the challenges in GSD by improving communication, trust, motivation and quality [15]. Paasivaara et al. [15] reports the increase in quality and frequency of communication along with improvement in collaboration attributed to scrum.

The frequent communication introduced by scrum has a positive effect on the level of trust between different development sites [15]. Improved communication and trust also has a motivating effect on team members [15]. A working demo at the end of each sprint helps maintain the quality of the product closer to release quality [15]. As a process, scrum has reflection build in through retrospectives, which can help team members to learn from their mistakes.

2.2.4 Teaching Global Software Engineering

It is common to see development of software being distributed across multiple sites [1]. Training and education in GSD has become quite important due to the growth of GSD [13]. Teaching GSD to students should involve practical component through which students can learn by doing things [13]. Systematic mapping of 19 GSD courses found that only one of the GSD course was purely theoretical [6].

There are various challenges in teaching GSD [13], [2]. GSD is a broad field and it is difficult to cover all problems of GSD in one course [13]. Students studying at the university have different schedules that alleviate the problem of scheduling for the course [13]. Students suffer from lack of motivation [13] and most of the projects end with the end of the course [2]. Instructors are also faced with the challenge of figuring out architecture and design of software that could work in a distributed setting [2]. Another challenge is to make communication effective among different sites [2]. Some of the traditional models like the waterfall are not designed for distributed settings [2].

Using scrum in a distributed setting has received positive experience from students [19]-[16]. Daily scrums have been found to be successful at detecting issues [19]. Students have mentioned about the importance of scrum as it gave them a structured approach and objectivity on the work they could achieve [18]. Scrum has shown to mitigate some of the typical challenges of GSD like effective distributed

communication, collaboration, teamwork, trust, cultural difference, etc [16].

2.3 Dependencies

It has been mentioned in section 2.1 that communication is the biggest challenge in distributed software development. High dependency between work items increases communication requirement for the success of the project [3]. A study by Sosa et al. [21] shows a strong co-relation between inter-dependence of work items and communication frequency. For successful distributed software development, module independency is essential [9]. Also work assignment should be done on the basis of independency of modules [9]. Hence, it becomes quite logical to assign independent tasks across different sites.

There are two ways of computing dependencies in codebase: logical and syntactical. Syntactical dependencies are computed on the basis of relationships like #include directives and functions calls [20][5] from the codebase which can be huge [7]. Whereas logical dependencies are computed from change release history, which is usually smaller than full codebase [7]. Syntactic dependencies do not uncover hidden dependencies like dynamic dependencies [7] and have shown to be less reliable [7][4]. A case study by Cataldo et al. [4] found that software failure is impacted by high logical dependency rather than high syntactical dependency and there is very little overlap between the two.

Chapter 3

Methodology

This chapter describes the methodology used to answer research question: ***Do students following the Scrum methodology learn to self assign stories in time to reduce cross team and cross site dependencies in a distributed software project?*** A case study was done on students working in the development of a mobile application called Mobilefant. Data wascollected from the source-control software (GitHub) and backlog management software (Agilefant) was analyzed to calculate dependencies across-teams and across-sites. Figure 3.1 gives a brief overview of methodology being employed.

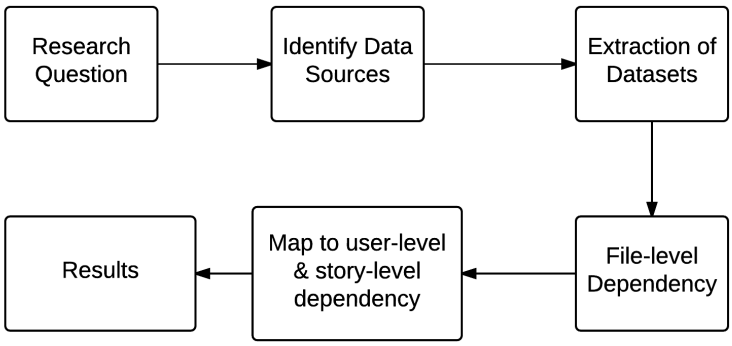

Figure 3.1: Research Methodology

3.1 Data Sources

GitHub GitHub is a web-based project hosting service for Git repositories (a distributed version control system)[13]. Apart from web hosting, it provides several

features like bugging tracking, task assignments, etc. It also promotes social coding by providing features like following other users activities, starring and watching Git repositories, etc. GitHub is a web app and it also has desktop and mobile clients.

The Mobilefant project used GitHub for source code management. Students used Git branches and GitHub's pull-request feature to merge their local code to the master. GitHub's data for Mobilefant project was required to calculate the technical dependencies on the basis of source code.

Agilefant[1] - It is an open source backlog management software for agile development. It provides a desktop website. Agilefant was used for backlog management throughout the project. It contains various information about the project such as iterations, teams, users, stories, tasks, effort spent, state of story (started, deferred, in progress), team burndown, story points, responsibles and timestamp. The Agilefant dataset provided information on sprints, teams and stories to compute technical dependencies between users and stories.

3.1.1 Extraction of Datasets

GitHub

GitHub provides a public API[2] to fetch data from a GitHub repository. GitHub-Mirror[3] tool was used to fetch the Mobilefant repository data from GitHub. It is a ruby gem that fetches data of a given repository and loads it into MongoDB[4]. A subset of the data is also stored in MySQL database. The MySQL database contained tables for pull requests, commits, users, pull-request commits, etc. A YAML file was configured for using this tool. YAML is shorthand for YAML Aint Markup Language [5]. It is a human- readable data serialization format. The credentials of the GitHub account, the MySQL database and the MongoDB database are mentioned in this YAML file. The YAML file can also contain the number of historical pages for API calls and number of requests per hour. A schema of database is given in Appendix A.1.

The MySQL database, as created with the GitHub-mirror tool, contained limited

[1] http://agilefant.com
[2] https://developer.github.com/v3/
[3] https://github.com/gousiosg/github-mirror
[4] https://www.mongodb.org
[5] http://yaml.org

Field	Type	Null	Default
sha	varchar(100)	YES	NULL
filename	varchar(100)	YES	NULL
additions	int(5)	YES	NULL
deletions	int(5)	YES	NULL

Table 3.1: Schema of commit_files table

Field	Type	Null	Default
sha	varchar(100)	YES	NULL
additions	int(5)	YES	NULL
deletions	int(5)	YES	NULL
message	varchar(100)	YES	NULL
commit_id	int(11)	YES	NULL

Table 3.2: Schema of commit_msg table

information about commits (commit table). To find technical dependencies, more information about a commit was needed, such as the name of files that were committed in the version control system and associated commit messages. As MongoDB database contained all information about a commit, a PHP script was used to extract the needed information from MongoDB database and load it into the MySQL database (code snippet to be added appendix). MySQL database was preferred because of its familiarity and ability to run set-based queries. NoSQL database like MongoDB would have resulted in additional learning curve.

The PHP script created two new tables - *commit_files* and *commit_msg*. Schema of these two tables are described in Table 3.1 and Table 3.2.

Schema of commit_files :
The field *sha* represents hash of a commit. The field *filename* specifies the name of file including the path. The fields *addition* and *deletion* provide the number of lines added or deleted.

Schema of commit_msg:
The *sha* field is hash of the commit as mentioned in the table 3.2. The commit message of each commit is mentioned in the *message* field. The field *commit_id* is the identifier for the commits. The fields, *additions* and *deletions* contain the number of lines added or deleted in a specific commit.

Agilefant
The Agilefant database was requested by the Software Business and Engineering In-

stitute (SoberIT) of Helsinki University of Technology. The dataset for the Mobilefant project was provided as a MySQL dump. The database dump was easily imported into the MySQL client without any processing. A schema of the database is given in Appendix A.2.

3.2 Calculation of Dependencies

To see if teams became intelligent in picking stories to reduce cross-team and cross-site dependencies, a comparison of intra-team vs. inter-team dependencies and co-located vs. distributed intra-team dependencies was made.

3.2.1 Intra-team vs. Inter-team dependencies

If dependencies exist between user stories, the teams responsible for those stories must coordinate their work, introducing additional coordination overhead. If there are dependent stories, then those stories should go to the same team. Hence, a comparison between the dependencies within a team (intra-team) and across the teams (intra-team) was made to see if the students learned to assign dependent stories to the same team and reduced cross-team dependencies. For this comparison, the dependency between the stories was needed. Firstly, file-level dependency was calculated from the source code and then mapped to the stories to get dependency at user-level and at story-level.

File-level dependency - Two methods for computing dependencies using source code were used: syntactic dependency and logical dependency, as described below.

Syntactic Dependency - Syntactic dependency identifies dependencies only through relationships like #include directives, function calls, etc. It does not uncover other dependencies like dynamic relationships [7].

Example of Syntactic dependency The sample code of a C++ file has #include directive. This header files allows to make interface. Hence, this file is syntactically dependent on myclass.h header file.

```
#include "myclass.h"

int main()
```

```
{
  MyClass a;
 return 0;
}
```

The sample of code of Java shows two java class files *Foo.java* and *Bar.java*. We can see from line bar.display(); that main method of *Foo* class depends on display method of *Bar* class. Hence, *Foo* is syntactically dependent on *Bar* class.

```
Foo.java
public class Foo {
  public static void main(String args[])
  {
      Bar bar = new Bar();
      bar.display();
  }
}
```

```
Bar.java
public class Bar {
  int data = 1;
  public void display()
  {
      System.out.println("Value of data is "+data+"\n");
  }
}
```

It is difficult to compute syntactic dependency between files of different languages and files containing code in dynamic languages like JavaScript. Tools identified for syntactic analysis were able to find dependency of only JavaScript files. The Mobilefant codebase consisted of HTML, CSS, XML and JavaScript files. Hence, syntactic dependencies were not considered for the analysis.

Logical Dependencies - Logical dependency keeps track of the files that have been checked-in or committed together to identify semantic relationships. These were computed by analyzing change history of codebase [7].

For example - If *file1.js* and *file2.js* are edited in the same commit, they are logically dependent on each other. For this analysis, files present in hooks and plugins directory were not considered as developers of this project did not edit these files. Hash tables were used to store name of files, name of files dependent on it and the count of number of times those two files are committed together.

Story Level Dependency Computation of logical dependencies revealed dependency between files. Files were then mapped to stories to compute dependency between stories. For this mapping, a bottom up approach was followed as depicted in Figure 3.4. The code related to a story was merged in the master source branch using pull-requests. A pull-request may contain one or more commits. A commit consists of one or more file changes. Hence, files were mapped to commits, commits were mapped to pull requests and pull requests were mapped to stories.

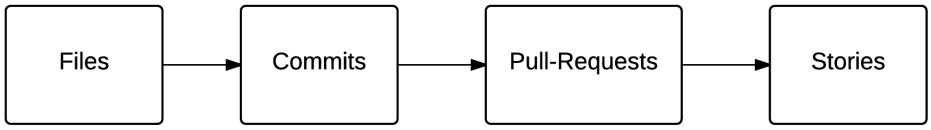

Figure 3.2: Process of mapping of dependencies from Source Files to User Stories

Once the dependency between files is calculated, the dependency between commits was calculated by finding which files were changed in a commit. This information was available from the *commit_msg* table. The relation between pull-request and commits was also available from the *pull_request_commits* table.

The relation between pull-requests and stories was not apparent. Only one team used the protocol to mention story ID in pull requests and pull request ID in story. Hence mapping of pull-requests to stories was done manually by reading pull-request and its associated commit messages and relating them with the story description. The time-stamp when a pull-request was created and merged was readily available from the SQL database. This timestamp was used to find the sprint in which the pull-request was created. This helped in reducing the number of stories, as pull requests were matched with stories of only relevant sprint. A member from each team was also contacted to confirm mapping of commits and pull-requests to the stories.

3.2.2 Co-located vs. distributed Intrateam dependencies

As mentioned in the introduction section, there were two co-located and two distributed teams in every sprint. The course instructor provided the information about the location of users separately. The teams were responsible for assignment of stories between team members. It has been mentioned in the literature review section that dependencies between work items increases the need for communication and coordination, which is difficult across sites. There was a time difference of 10 hours between the two locations, which can further dampen communication. Hence, lower distributed dependencies as compared to co-located dependencies within team were desired for the success of the project. Userlevel dependency is calculated for comparison of co-located vs. distributed dependencies.

User Level Dependency - Two users are dependent on each other if they worked on same story or if they worked on stories that are dependent on each other. List of users responsible for a story was available directly from backlog management database. Users were considered responsible for a story if they have made a commit for its completion even if they are not the assignee of the story in Agilefant. After computing dependent stories, stories were mapped to the users to compute user level dependencies. Every pair of dependent user is counted as one dependency.

3.2.3 Consideration of deferred stories

If a story cannot be completed in one sprint, developers deferred it to the subsequent sprint. For story dependency analysis, three different approaches can be used -

- Counting stories in all the sprints from creation to completion.
- Counting stories only in the sprint they were created.
- Counting stories only in the sprint they were completed.

In sprint planning sessions, the students had the opportunity to pick stories when they are created and presented by product owner. Hence, it made sense to count stories only in the sprint they are created to reflect the learning of students in task assignment.

Chapter 4

Analysis and Results

This chapter presents the quantitative analysis and results to examine if students became intelligent in work assignment to reduce cross-team and cross-site dependencies in the globally distributed project.

4.1 Inter-team vs. Intra-team dependencies

As mentioned in methodology section, for computing inter-team vs. intra-team dependencies, file level dependency was computed first and then mapped to stories to compare dependencies across teams and dependencies within teams.

4.1.1 Logical Dependency of Files

To find logical dependencies in source-code files, code was written in Java. It connects to the MySQL database to retrieve commits and related source files and then computes dependencies in the source files.

For computing logical dependency of files, a threshold of 4 commits was used. Hence, files were considered to be logically dependent on each other if they have been committed together at least 4 times. This threshold was used to identify pairs of files that have strong interdependency. Also, without any threshold almost every file was dependent on each other. This can be attributed to the fact that most of the developers were using Git for the first time and might have made small changes in files unintentionally. Every pair of dependent files was counted as one dependency. Number of files grew in subsequent sprints and reached 187 by the last sprint. Hence there can be at most n!/((n-2)!*2!) dependencies, where n is number of files).

Sprint Number	Total Stories	Dependent Stories	Percentage of Dependent Stories
2	27	11	40.74%
3	40	15	37.5%
4	33	12	36.33%
5	56	19	33.92%

Table 4.1: Dependent Stories in each Sprint

4.1.2 Logical Dependency of Stories

Every pair of dependent story was counted as one dependency. There were 156 unique stores in 4 sprints. So, in total 12090 dependencies are possible as 156!/(154!*2!) = 12090.

Stories which took less than 240 minutes for completion were not included in the analysis as we assume that coordination requirement for these stories is very low. For computing logical dependency of stories, files were mapped to commits, commits were mapped to pull requests and pull requests were mapped to stories as mentioned in methodology section.

Table 4.1 shows that the number of dependent stories increased with increase in total stories and decreased with decrease in total stories. The number of stories in each sprint varies. Because of this reason, simply looking at the number of dependencies does not give any conclusive results. The percentage of dependent stories is more informative. We observe that the percentage of dependent stories decreased in subsequent sprints. As the PO was responsible for creation of stories, hence it can be concluded that the PO became smart in creating a higher fraction of loosely coupled stories.

4.1.3 Inter-team vs. Intra-team dependency at story level

Table 4.1 already shows that even though the PO created a higher fraction of loosely coupled stories the number of dependent stories decreased across sprints. As the students had the power to self-assign stories, the students assigned dependent stories in the same team to reduce the need for cross-team coordination.

Table 4.3 shows the total number of dependencies as well as the number of intra-team and inter-team dependencies in each sprint. The table also presents the percentage of intra-team and inter-team dependencies in each sprint. It can be observed from

Sprint Number	Total Dependencies	Intra-team Dependencies	Percentage of Intra-team Dependencies	Inter-team Dependencies	Percentage of Inter-team Dependencies
2	43	6	14%	37	86%
3	106	24	23%	82	77%
4	44	12	27%	32	73%
5	142	74	52%	68	48%

Table 4.2: Inter-team and Intra-team Dependencies

Sprint Number	Number of co-located dependencies	Number of distributed dependencies	Percentage of distributed dependencies
2	9	3	25%
3	16	6	27%
4	8	6	43%
5	8	2	20%

Table 4.3: Co-located and Distributed intra-team dependencies

the table that the percentage of inter-team dependencies decreased and the percentage of intra-team dependencies increased across sprints. This suggests that students learned to pick stories to minimize their need for cross-team coordination.

4.2 Co-located vs. distributed intra-team user-level dependencies

As mentioned in the methodology section, a comparison between co-located and distributed intra-team dependencies was made by calculating user-level dependencies. Dependent stories were mapped to users to find user-level dependencies.

Table 4.3 does not show any pattern in the number of intrateam user-level distributed dependencies. An increase in the percentage of intrateam distributed dependency after sprint 2 can be attributed to the splitting and joining of teams. Interviews with the students revealed that there was some confusion in the assignment of new stories that were similar to deferred stories as their original assignees now belonged to different teams. Table 4.3 also shows that intra-team distributed dependencies are always less than half of total intra-team dependencies. Hence, it we can be concluded that teams intelligently assigned tasks within teams to keep intra-team cross-site

dependencies in check.

Chapter 5

Discussion

This chapter discusses how the results of the analysis reported here relate to the existing literature, how they answer the research question and what are the possible threats to validity of these results.

Work mentioned in this report is a subset of a bigger project [14], which describes experience of teaching GSD using scrum practices. It has already been mentioned in the literature review section that communication is the biggest challenge of GSD. Modularization is difficult to achieve and dependency of work items increase the need for coordination. Hence, assignment of dependent stories to the same team and on same site within the team is more desirable. Table 4.2 shows better task allocation over time to reduce the percentage of inter-team dependencies and table 4.3 shows good task allocation within team to keep a check on intrateam cross-site dependencies. Hence, it can be concluded that students learnt to self-assign stories in time to reduce cross-team dependencies and kept a check on intrateam cross-site dependencies.

Students' learning can be attributed to the usage of scrum. It has been mentioned in the literature review section that scrum enforces structured communication, which can mitigate some of the typical challenges of GSD [15]. Some of the results mentioned in Paasivaara et al. [14] also support this claim: -

- High level of satisfaction among students in using scrum practices for both co-located and distributed members.
- High level of satisfaction in using communication practices for both co-located and distributed members.
- Increase in teamwork quality in subsequent sprints.

- Better fulfillment of communication requirement in subsequent sprints.

Scrum helped the students' to learn better task assignment. This claim is strengthened by interview results mentioned in the case study by Paasivaara et al. [14]. The students had positive opinion about usage of scrum for distributed teams as it increased project engagement and kept everybody on track.

5.1 Threats to Validity

The research presented in this report has some limitations. One of the biggest threats to validity of this research is the small dataset. There were only 4 sprints in the course. Clearly, more sprints would have strengthened the argument that students learned to self-assign stories in time to reduce cross-team dependencies and kept a check on intra-team cross-site dependencies.

There were also some confounding variables. The number of students across different sites was not same (15 Canadian and 8 Finnish students) which undermines comparison of co-located vs. distributed intra-team dependencies. The duration of every sprint was not same and the number of stories in every sprint was also varying. Also, the PO created a higher fraction of independent stories, which could have helped students in efficient story assignment. Another confounding variable is the split of teams after sprint number 3. It was found that this split lead to confusion of assignment of deferred stories and newly created stories that were similar to deferred stories. Some of these confounding variables can be argued to be responsible for lack of any pattern in co-located vs. distributed user level dependencies. The course at the University of Victoria had lectures and in-class discussions where as at the Aalto University, it was a capstone project-based course without lectures. Also, the Finish students started course in September 2013 whereas the Canadians joined in January 2014. These factors are bound to have some effect on learning of the students.

It cannot be claimed that logical dependency calculation is perfect and it uncovers all dependencies. Although manual mapping of stories to commits was performed carefully and members of different teams were also contacted for verification, a small human error cannot be discarded.

Appendix A

Additional Information

A.1 Schema of MySQL database of Github dataset

A.2 Schema of MySQL database of Agilefant dataset

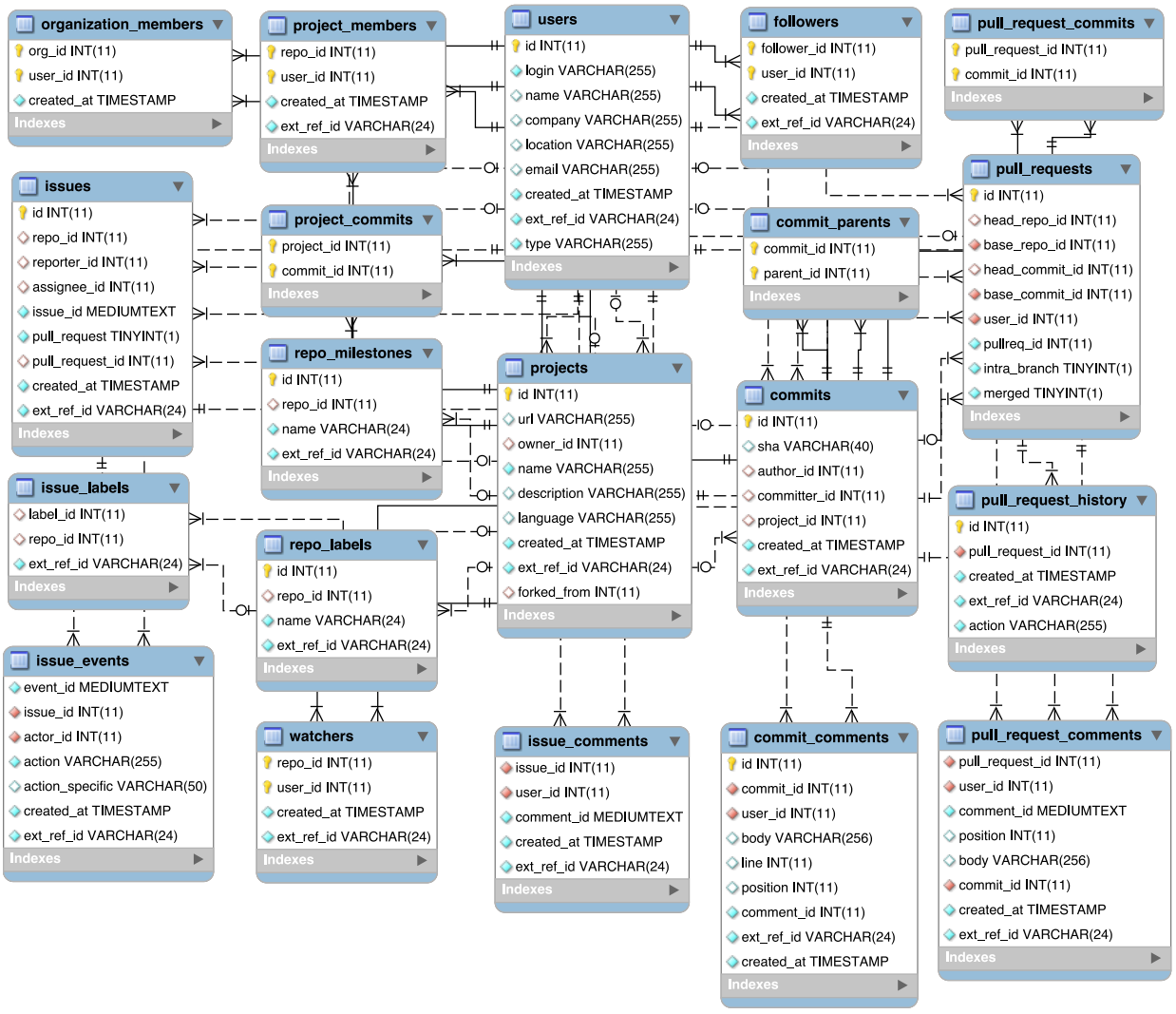

Figure A.1: Schema of GitHub-Mirror MySQL dataset. Borrowed from ghtorrent.org

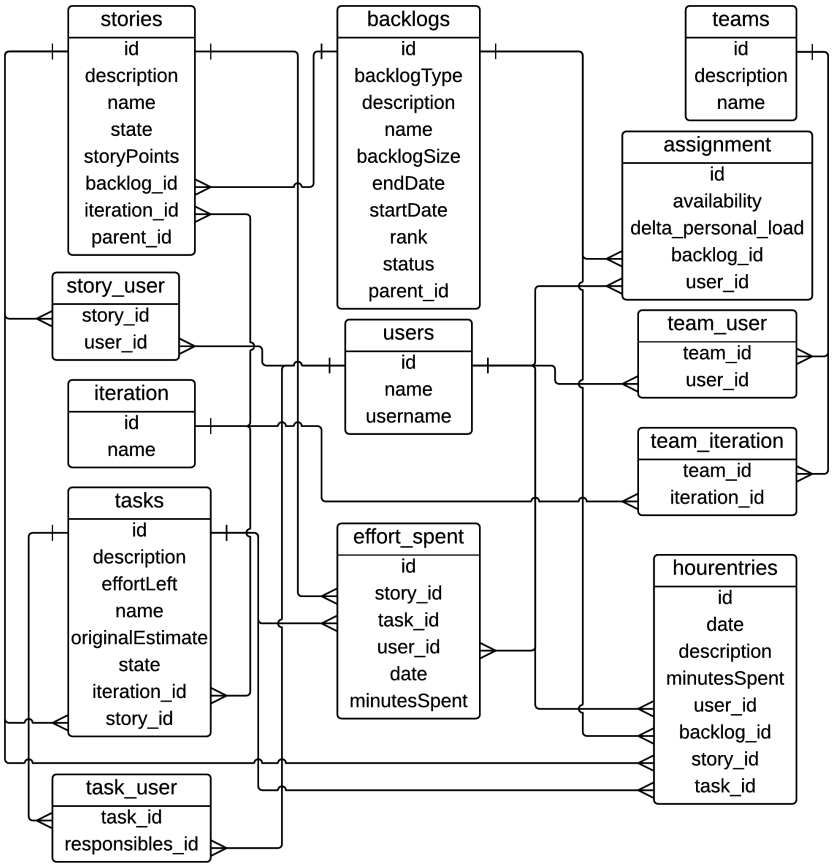

Figure A.2: Schema of Agilefant MySQL dataset

Bibliography

[1] Par J. Agerfalk, Brian Fitzgerald, Helena Holmstrm Olsson, and Eoino Conchuir. Benefits of global software development: The known and unknown. In *Proceedings of the Software Process, 2008 International Conference on Making Globally Distributed Software Development a Success Story*, ICSP'08, pages 1–9, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] Yuanfang Cai and Warren Baelen. On the development of pedagogical materials for globally distributed software engineering. In *Proc of Collaborative Teaching of Globally Distributed Software Development-Community Building Workshop*, 2012.

[3] Marcelo Cataldo and James D. Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Trans. Softw. Eng.*, 39(3):343–360, March 2013.

[4] Marcelo Cataldo, Audris Mockus, Jeffrey A Roberts, and James D Herbsleb. Software dependencies, work dependencies, and their impact on failures. *Software Engineering, IEEE Transactions on*, 35(6):864–878, 2009.

[5] Daniela Damian, Luis Izquierdo, Janice Singer, and Irwin Kwan. Awareness in the wild: Why communication breakdowns occur. In *Proceedings of the International Conference on Global Software Engineering*, ICGSE '07, pages 81–90, Washington, DC, USA, 2007. IEEE Computer Society.

[6] Luiz Leandro Fortaleza, Tayana Conte, Sabrina Marczak, and Rafael Prikladnicki. Towards a gse international teaching network: Mapping global software engineering courses. In *Proceedings of the Second International Workshop on Collaborative Teaching of Globally Distributed Software Development*, pages 1–5. IEEE Press, 2012.

[7] Harald Gall, Karin Hajek, and Mehdi Jazayeri. Detection of logical coupling based on product release history. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 190–198. IEEE, 1998.

[8] Geir Kjetil Hanssen, Darja Smite, and Nils Brede Moe. Signs of agile trends in global software engineering research: A tertiary study. In *Global Software Engineering Workshop (ICGSEW), 2011 Sixth IEEE International Conference on*, pages 17–23. IEEE, 2011.

[9] James D Herbsleb and Rebecca E Grinter. Architectures, coordination, and distance: Conway's law and beyond. *IEEE software*, 16(5):63–70, 1999.

[10] James D Herbsleb and Rebecca E Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st international conference on Software engineering*, pages 85–95. ACM, 1999.

[11] Emam Hossain, Muhammad Ali Babar, and Hye-young Paik. Using scrum in global software development: a systematic literature review. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pages 175–184. Ieee, 2009.

[12] A. Mockus and J. Herbsleb. Challenges of global software development. In *Proceedings of the 7th International Symposium on Software Metrics*, METRICS '01, pages 182–, Washington, DC, USA, 2001. IEEE Computer Society.

[13] Miguel J Monasor, Aurora Vizcaino, Mario Piattini, and Ismael Caballero. Preparing students and engineers for global software development: a systematic review. In *Global Software Engineering (ICGSE), 2010 5th IEEE International Conference on*, pages 177–186. IEEE, 2010.

[14] Maria Paasivaara, Kelly Blincoe, Casper Lassenius, Daniela Damian, Jyoti Sheoran, Francis Harrison, Prashant Chhabra, Aminah Yussuf, and Veikko Isotalo. Learning global agile software engineering using same-site and cross-site teams. In *Proceedings of the 2015 International Conference on Software Engineering*, ICSE '15. IEEE Press, 2015.

[15] Maria Paasivaara, Sandra Durasiewicz, and Casper Lassenius. Using scrum in a globally distributed project: A case study. *Softw. Process*, 13(6):527–544, November 2008.

[16] Maria Paasivaara, Casper Lassenius, Daniela Damian, Petteri Räty, and Adrian Schröter. Teaching students global software engineering skills using distributed scrum. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1128–1137, Piscataway, NJ, USA, 2013. IEEE Press.

[17] Kenneth S. Rubin. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley Professional, 1st edition, 2012.

[18] Christelle Scharff. Guiding global software development projects using scrum and agile with quality assurance. In *Proceedings of the 2011 24th IEEE-CS Conference on Software Engineering Education and Training*, CSEET '11, pages 274–283, Washington, DC, USA, 2011. IEEE Computer Society.

[19] Christelle Scharff, Olly Gotel, and Vidya Kulkarni. Transitioning to distributed development in students' global software development projects: The role of agile methodologies and end-to-end tooling. In *Proceedings of the 2010 Fifth International Conference on Software Engineering Advances*, ICSEA '10, pages 388–394, Washington, DC, USA, 2010. IEEE Computer Society.

[20] Christelle Scharff, Samedi Heng, and Vidya Kulkarni. On the difficulties for students to adhere to scrum on global software development projects: preliminary results. In *Collaborative Teaching of Globally Distributed Software Development Workshop (CTGDSD), 2012*, pages 25–29. IEEE, 2012.

[21] Manuel Emilio Sosa, Steven D Eppinger, Michael Pich, David G McKendrick, and Suzanne K Stout. Factors that influence technical communication in distributed product development: an empirical study in the telecommunications industry. *Engineering Management, IEEE Transactions on*, 49(1):45–58, 2002.