12-2019

Understanding gaps between established Software Engineering Process knowledge and its actual implementation

Sayantika Bhattacharya
sxb2606@rit.edu

Understanding gaps between established Software Engineering Process knowledge and its actual implementation

APPROVED BY

SUPERVISING COMMITTEE:

Dr. Scott Hawker, Supervisor

Erika S. Mesh

Understanding gaps between established Software Engineering Process knowledge and its actual implementation

by

Sayantika Bhattacharya

THESIS

Presented to the Faculty of the Department of Software Engineering

Golisano College of Computer and Information Sciences

Rochester Institute of Technology

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Software Engineering

Rochester Institute of Technology

December 2019

Abstract

Understanding gaps between established Software Engineering Process knowledge and its actual implementation

Sayantika Bhattacharya, M.S.
Rochester Institute of Technology, 2019

Supervisor: Dr. Scott Hawker

A part of Software Engineering (SE) Process Improvement is to identify and bridge the gaps between what we learn about established SE Processes and what we actually execute. Students majoring in SE degrees learn about various established SE processes in class and also try to execute them in their academic projects. In our research, we analyze student SE projects and interview these project teams to identify the gaps between what students learn in class about SE processes, what they think they execute, along with understanding the cause behind these gaps.

Table of Contents

List of Figures

Chapter 1

Introduction

Be it small start-ups or multi-national corporations, software products are now being created by small and large teams of software engineers (SE). These SEs may perform out the various roles of product managers, designers, architects, requirement engineers, developers, testers, etc. They come from varied backgrounds of cultures, work experience, knowledge, personalities and so on. And they have to work together to achieve the goal of delivering a good quality software product within a certain time frame. This is no easy feat to achieve without having some rules and guidelines that provide every team member the same environment to work in. And thus, when planning a software project, it is crucial to keep in mind a software engineering process that is appropriate for the product being developed and to which the whole team adheres.

To deliver good software, we require proper planning, incorporation of design principles and patterns, clear unambiguous communication between the team members about objectives and goals, understanding precise user requirements, and so much more. Software products are now being engineered under different types of SE processes based on project and product requirements. It

has now become possible to identify a lot of the SE practices that have been a part of successful projects and also some of them which have been common in the not so successful ones [15]. Laplante defines software engineering as "a systematic approach to the analysis, design, assessment, implementation, test, maintenance and re-engineering of software." [21]. The process of developing a software product is not just good coding practices. It is about using the right approach, understanding the requirements and goals of the product, having a well designed plan, implementation, testing and so much more. Kroll and MacIsaac have written [20] about how to identify which SE practices are best suitable for your project and how to go about incorporating them. The software community is trying to incorporate these in the development of software products.

It may be a reasonable assumption by the software development community that a vast number of students majoring in SE degrees have been taught the importance of choosing the appropriate SE processes for their software products and that their coursework should have required them to have discussions on identifying different SE processes and appreciating their importance.

We wanted to find similarities and differences between what the students thought they did and what they actually did. Using this information, we wanted to figure out the gaps between the intended process, and the actual execution of them when it comes to working on projects.

Chapter 2

Background

For our research, we wanted to narrow down our study by focusing on an agile project management methodology called Scrum [28] because this was what the students we worked with for our research were using as it was a part of their course requirement. We wanted to further analyze the gaps between learned knowledge and actual implementation of this process by SE students.

2.1 Our Research

There is a lot of rich data available in software repositories which can provide us with a large amount of information regarding the code, the people, the processes, the bug tracking system, issue tracking system, etc. We have mined four repositories of student projects all done for the same graduate level course, to gather data that provides insight on the SE process being used in the projects. We have also conducted interviews on teams of students who have worked on these, and gathered information about the projects and their opinions and views about the SE processes they had to follow for these projects. The transcripts for these interviews are accessible in the appendix. We have used the mined data from repositories to compare with the interview data.

This helped us find similarities and differences between what the students thought they did and what they actually did. We try to analyze some of the decisions behind the implementation of these processes and challenges faced when it comes to using appropriate SE processes in real world projects.

Another field of software development that we wanted to reflect upon, is the relation between the supporting artifacts and the real process execution. Artifacts prove to be some of the most useful aspects of a software product development. They provide a wealth of information about the software product, the software process, the design of the product and its process, the goals, etc. Hence, it is important for software engineers to have a well documented project. We wanted to hear the opinions of the SE students on this, and compare their opinions with the artifacts they created for their projects.

2.2 Research Questions

Our research questions are stated as below:

1 Do SE students adhere to established software engineering processes in their projects?

2 What are the similarities and differences between SE students' opinions on the software process phases they believe they have used, and the actual implementation of them in the projects?

3 What are their views and opinions about their SE classroom knowledge

versus the feasibility of applying said knowledge when working on real projects?

Chapter 3

Research Methodology

3.1 Data Sample

For our research we worked on 4 projects done by 4 teams of students from the SE department at Rochester Institute of Technology. These teams of 4 members each, had worked on the online Web Checkers project [13] as part of their course curriculum of Foundations of Software Engineering (SWEN-610) [10] close to two years prior to us working with them on this research. They had used software processes and tools provided by their department. WebCheckers is a web based game available 24/7, which must allow players to play checkers with other signed-in players according to the American rules [12]. The game user interface (UI) will support drag-and-drop browser capabilities for making moves.

Project Scope: An Agile style development approach was used to implement functions in a team of 4 students in 4 sprints. Communication amongst team members is primarily to be done on a Slack workspace [8]. Project work tracking will be done using Trello [9].

We chose these teams because their projects' main focus was to execute the SE process and since we wanted to identify the differences between what

students learn in class about established SE processes, and their actual execution in projects, we felt this was the best demographic to start this research with.

3.2 Overview

Our research methodology can be divided into three parts:

1 Software Process Mining

2 SE student team interviews

3 Word Frequency and relevancy of project artifacts

We extracted certain information about each of the projects by mining their repositories. But this information remained incomplete without bringing in the human aspect behind some of the actions and decisions. Hence, we decided to conduct interviews with each of the project teams. Along with the reasons behind some of their actions, we also questioned them on their opinions on various aspects of SE processes. The interview questions (see figure 3.1) were related to the SE processes, the different phases, students' opinions on them, etc. that these teams worked on and their opinions on some of the aspects of these SE processes. Our goals were

1. To understand the outputs that we got from mining repositories better by bringing in the human aspects of decision making and challenges faced.

2. To get opinions of the students on the SE process they used and it's phases.

1. What Software engineering process did you use for your project?
2. What are the SE practices/activities that you followed?
3. What kind of testing did you perform?
4. Did you document your testing on GitHub?
5. What SE quality practices did you incorporate in your project?
6. How much importance do you place on commenting your code?
7. What kinds of comments does your project have?
8. How much importance does documentation have in your project?
9. What kind of documentation does your project have?
10. How has the work been divided between team members?
11. What do you feel is the contribution of each person when it comes to
    i. Design
    ii. Documentation
    iii. Requirement Gathering
    iv. User Stories/Use cases
    v. Feature Listing
    vi. Coding
    vii. Testing
    viii. Issue Management
12. How did you collaborate with each other?
13. What are the collaboration issues that you have encountered?
14. How did you work on resolving collaboration issues?
15. What type of test artifacts are there on GitHub?
16. How do you define quality?
17. What are you doing to maintain SE process quality?
18. How did you handle your releases?
19. Did you document issues on GitHub?
20. Did each one of you commit/merge your code?
21. Did any of the tools help or hinder process adherence?
22. What types of coding issues did you encounter? How did you work on resolving them?
23. What programming languages did you choose for your project?
24. What were the main reasons for choosing the language you chose?
25. Does one language have any specific benefits over another when it comes to incorporating SE practices.
26. Code reviews?

Figure 3.1: List of Questions for interview

The interview transcripts for each of the teams are available in Appendix A.1. The teams had worked on these projects almost two years prior to when we conducted the interviews, so we did get a few instances where the students had not remembered the answer to a question. We recorded

the interviews of each student and then the transcripts were made from those recordings.

From the output we got from mining software repositories, and from the interview results, we were able to get better insights into how and why things were done in the projects, and also the differences between what the students thought they did, versus what actually was done.

Since the supporting work products, or as we shall call them here as artifacts, are also a part of SE processes and contains a wealth of information behind the concepts of design, requirements, testing, etc., we decided to analyze the artifacts as well. We did this to primarily understand the artifacts created, if they were relevant to the project, and if the content of these artifacts were relevant to the artifact names, since that would point towards understanding of what the artifacts comprise of.

We shall discuss each of these 3 research methodology parts in detail in sections 3.3, 3.4 and 3.5.

3.3 Mining Software Repositories

Process mining can be defined as extracting essential information from a set of real executions based off on a structured process description [24]. It consists of mining software repository event logs and thus bridges the gap between traditional model-based process analysis and data-centric analysis techniques [19]. Process mining lets us gather information about the processes

that have been used in a software project. It may include process identification, new process discovery, analysis of process performance, prediction of various factors, identifying quality attributes, understanding the history of a project and its process, etc [19].It can be used in getting information on how people and procedures work. It can also help in the comparison of predefined processes and the actual processes [26]. These can lead us to assess software processes and help in software process improvement approaches. However, one of the challenges of process mining is to find useful and accurate data and to correctly and justifiably analyze them before reaching conclusions.

3.3.1 Software Process Mining

Repositories may contain a wealth of information, but not all information extracted from a repository may be relevant or helpful. With the advent of open access software repositories like GitHub, Bitbucket, etc, anyone can create a repository. Many times these repositories do not even contain software projects. When a researcher wants to access this information, it may result in getting a ton of data, out of which a large chunk can be irrelevant and unhelpful data, which is often called as noise. To wade through this data to find out relevant information is a part of software repository mining, and is considered one of the biggest challenges of mining software repositories. Even when working on well known software repositories, it may happen that we only want to extract certain information and ignore most of the data stored in the repository. This can be done manually but is a hugely cumbersome task.

For our study, we decided to create a software process mining tool to extract data that we want out of a repository. We were inspired by the repository mining tool called Reaper [24]. Reaper is an open source tool which helps cancel out the noise that mining repositories generates. It allows to differentiate between repositories containing engineered software projects from the noise (non-software projects) when working on a set of random repositories. The reason we wanted to replicate certain parts of Reaper and create a new tool are as below:

1. Reaper relies on the GHTorrent database [17] to access repositories. We had a list of repositories on which we wanted to conduct our study. These repositories are private and are on GitHub and hence, GHTorrent database did not have access to these repositories.

2. The GHTorrent database setup consumed a lot of space and hence to avoid that, we wanted to directly extract information using GitHub APIs [1].

Thus, we created our own tool inspired by Reaper, called the Process Miner [6].

3.3.2 Process Miner - Concept

For our research, we wanted to understand certain metrics for a project such as team structure, commit schedule, life cycle models, team roles, adherence to good SE practices, etc. We also wanted to identify application of SE

processes and hence needed to understand how to extract that information. We gathered information such as these from software repositories in GitHub using the Process Miner.

Process Miner uses the same concept of attributes as Reaper does. Attributes can be defined as a set of data which individually, and combined together, provides us various information about a repository. There are various attributes that we will be using in our research. They are listed as below:

3.3.2.1 Community:

Collaboration amongst team members is an essential part of software engineering. The presence of various team members and their contributions indicates collaboration and cooperation amongst them, [14] but contribution is subjective in a software engineering team. There may be teams comprising of designers, developers, testers, content writers, etc and each of their contribution is crucial for a project to be successful. It is not necessary that a designer executes the work of a tester, a content writer performs merges and commits.

However, when it comes to GitHub repository mining, it is difficult to find the various roles a team member has participated in as we do not always have the necessary artifacts, say, an architecture diagram, or user stories on GitHub. We followed what Reaper [24] did and focused primarily on source code commits as contribution and tried to extract the percentage of contributors who have accounted for at least 80 percent of the total number of commits

to a source code repository and call them as core contributors, where core contributors are a set of contributors who manage and steer a project towards the project objective [23] [29]. Commits help identify the software development process of a project. In Process Miner, we take the total number of commits up until a certain date, categorize them by authors, and find out each author's commits. This would help us compare between the commits of different authors and should help us identify the authors who have a higher number of commits thereby letting us identify the core developers of the project [14].

However, there is an issue with this process which was pointed out by Reaper [24] that, in using commits for understanding team contribution, there arises the concept of "fake users". These are the users who do not have GitHub accounts or have multiple account ids and often let the "real user" push their commits to GitHub using their account. This may result in incorrect core contributor identification. Also, some developers commit frequently and others minimally, and that is not necessarily a reflection of the actual amount of work done. These are issues that we intend to work on in our future work.

3.3.2.2 Documentation:

There are various forms of supporting documentation that a software engineering team may create like architecture diagrams, requirements documents, design documents,test plans and results, etc. There are also documentations that are part of the source files such as code comments, source code, etc. The reason documentation is considered to be a good software engineer-

ing practice is because it helps in maintainability [14] and helps in knowledge transfer.

Source code comments have been found to be among some of the most important forms of documentation [30] [14]. Comments are considered important because

a They are a rich form of data since they talk about the source code

b Comments point towards teams keeping maintainability in mind when working on software projects.

Just like in Reaper [24], we have used comment ratio which is the number of comment lines of code, to the number of non-blank lines of source code in Process Miner.

3.3.2.3 Languages:

This is mostly as information to know and for future use to see if there is a preference to use certain programming languages based off the software process being used.

3.3.2.4 History:

Long time lapses between changes made to source code may be the reason of non-uniform software development, with periodical bursts of work and then stagnancy for a period of time. Periodic and constant changes point to-

wards a uniform software development,which indicates appropriate refactoring during iterative development.

3.3.2.5 Continuous Integration (CI):

Software Engineers are expected to continuously integrate their code with code from other developers and they often do this using various CI services such as Travis CI, Hound, Solano, etc. Thus, the presence of such tools may indicate evidence of collaboration in a team. This is especially important for large projects, but may not be applicable to small academic course projects.

3.3.2.6 Issues:

GitHub defines issues as suggested improvements, tasks, or questions related to the repository [7]. Issue logging hints towards improvement of code. We want to find out the number of issues that occurs within a timeframe for a software project. This is due to the following reasons:

a We want to understand the frequency of issues occurring and their resolution – this shall indicate towards the practice of project management and issue handling practices.

b We want to find out the primary types of issues by analyzing the comment section.

Process Miner is able to extract issues that are actually filed under "Issues" by developers from the GitHub APIs [1].

3.3.2.7 Stars:

This allows us to identify hugely popular projects. We can have a look at these repositories to see if there are certain practices that are being followed which make them popular and decide if those practices need to be brought into the limelight and implemented by SEs to make their process, project and product better.

3.3.2.8 License:

Licenses are important in the course of open source projects [24]. A software with no license is protected by default copyright laws, and hence it is important to understand the license of the project we want to extract information from so as to avoid legal issues. For this research, we are only working on private repositories, but this would be useful for our future work.

3.3.3 Process Miner Tool - Build and Implementation

The process miner application is a NodeJS application that runs analysis on software project repositories.

3.3.3.1 Objectives

The main objective of the application is to run analysis on a software project repository and report the result of the analysis. The analysis itself is done by self contained units known as attributes. Think of attributes as small independent programs that, given an input, always produce an output

which is a measure of a specific metric. Together with the output of all the attributes, inferences can be made on a repository.

3.3.3.2 Usage

The process miner application runs using the concept of a session. A session represents a list of repositories to process, list of attributes to run on the repositories, and the data loader to use to load the repositories. The session itself is represented with the help of a configuration file. The configuration file is written in the JSON format (see figure 3.2).

```
module.exports = {
  repositories: [
    'https://github.com/angular/angular',
    'https://github.com/jquery/jquery',
    'https://github.com/lodash/lodash'
  ],
    attributes: ['documentation', 'continuous-integration', 'community', 'history',
'license', 'issues', 'stars'],
  loader: 'github'
};
```

Figure 3.2: Configuration File snippet

3.3.3.3 Example Configuration File

The fields in the configuration file are as follows:

1. Repositories - This is the list of repositories to be processed by the process miner.

2. Attributes - This is the list of attributes to run on each repository.

3. Loader - The data loader to use to fetch the repositories.

Within the code, there is a ProcessMiner class that contains a static level function called run, which accepts the configuration.

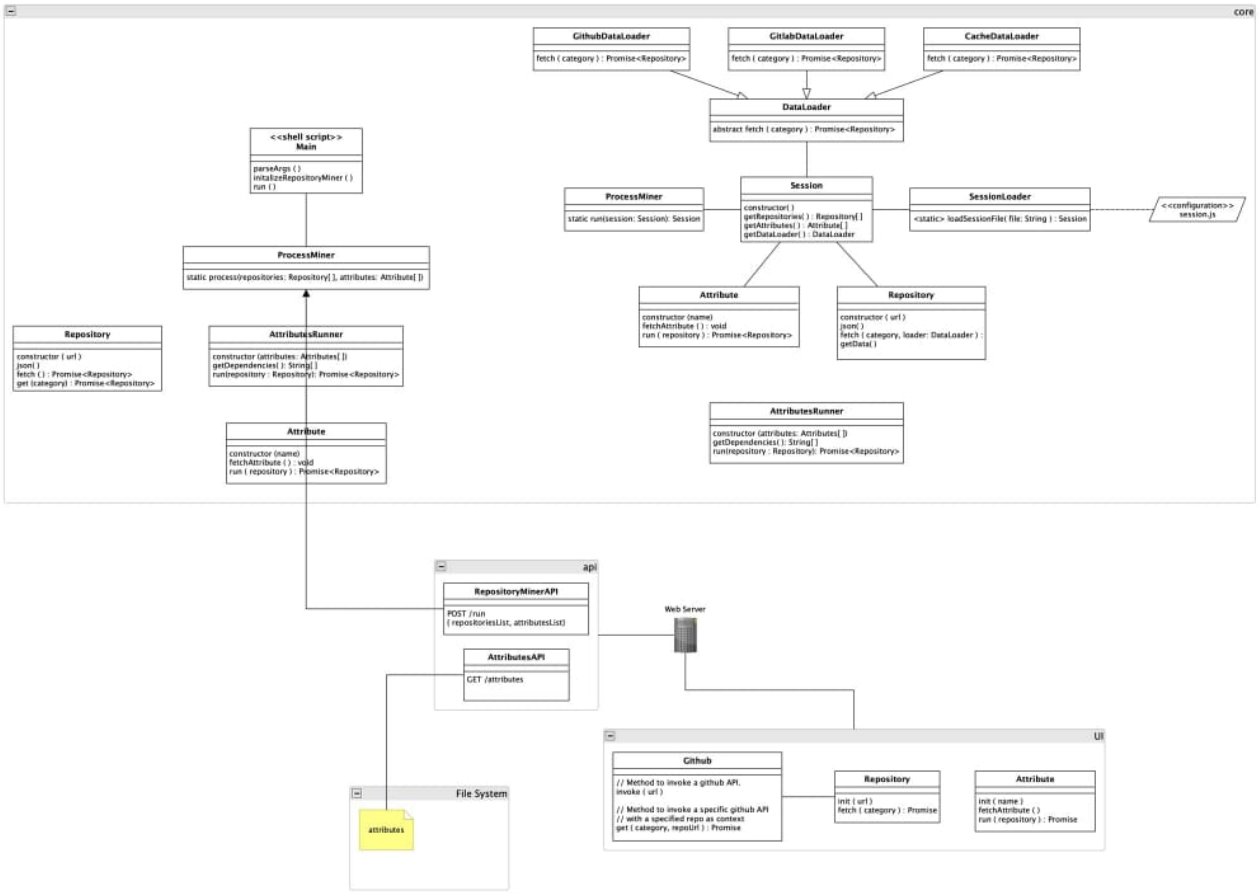

Figure 3.3: Process Miner Class Diagram

The diagram has been broken down in figures 3.4, 3.5 and 3.6 for better visibility.

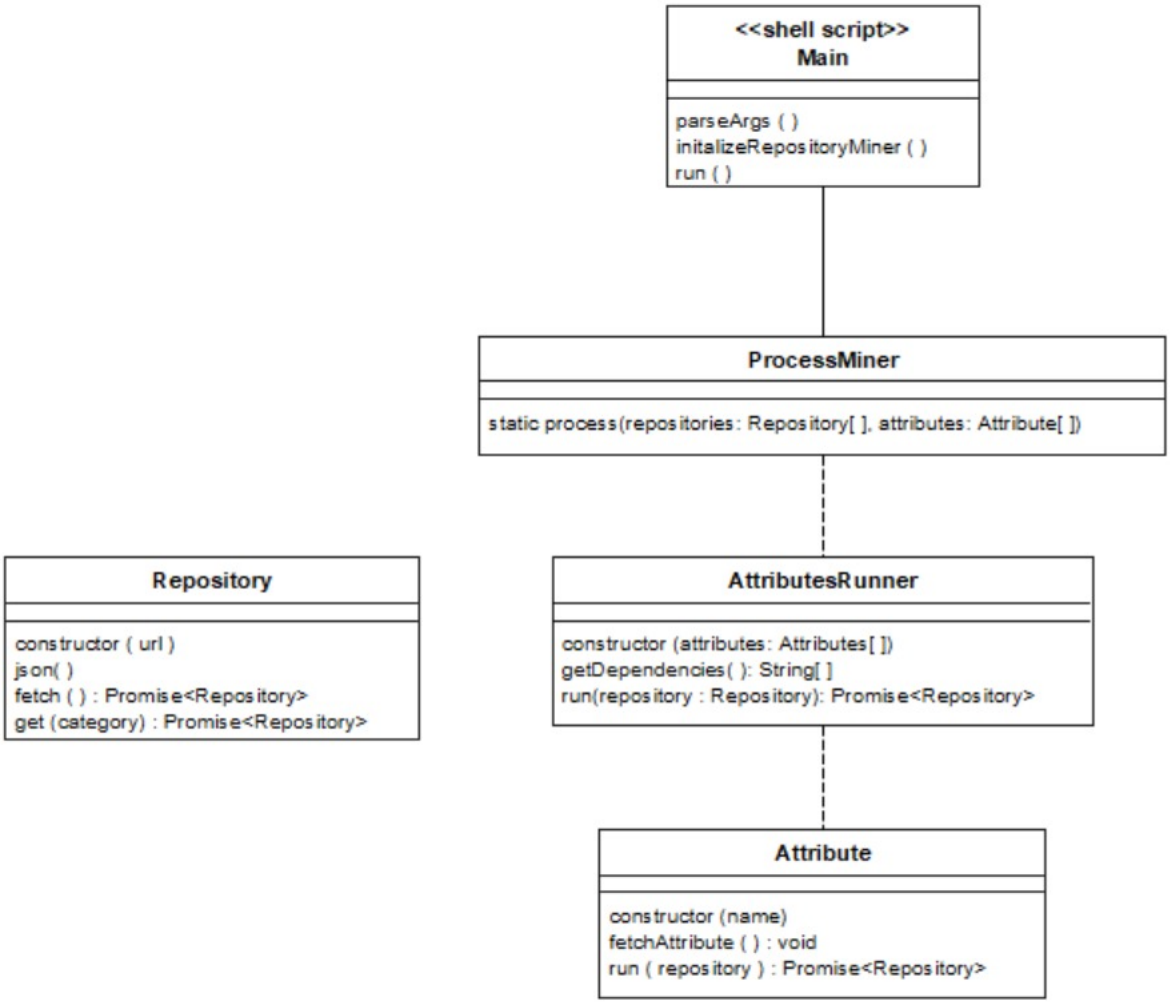

Figure 3.4: Classes containing core logic of Process Miner

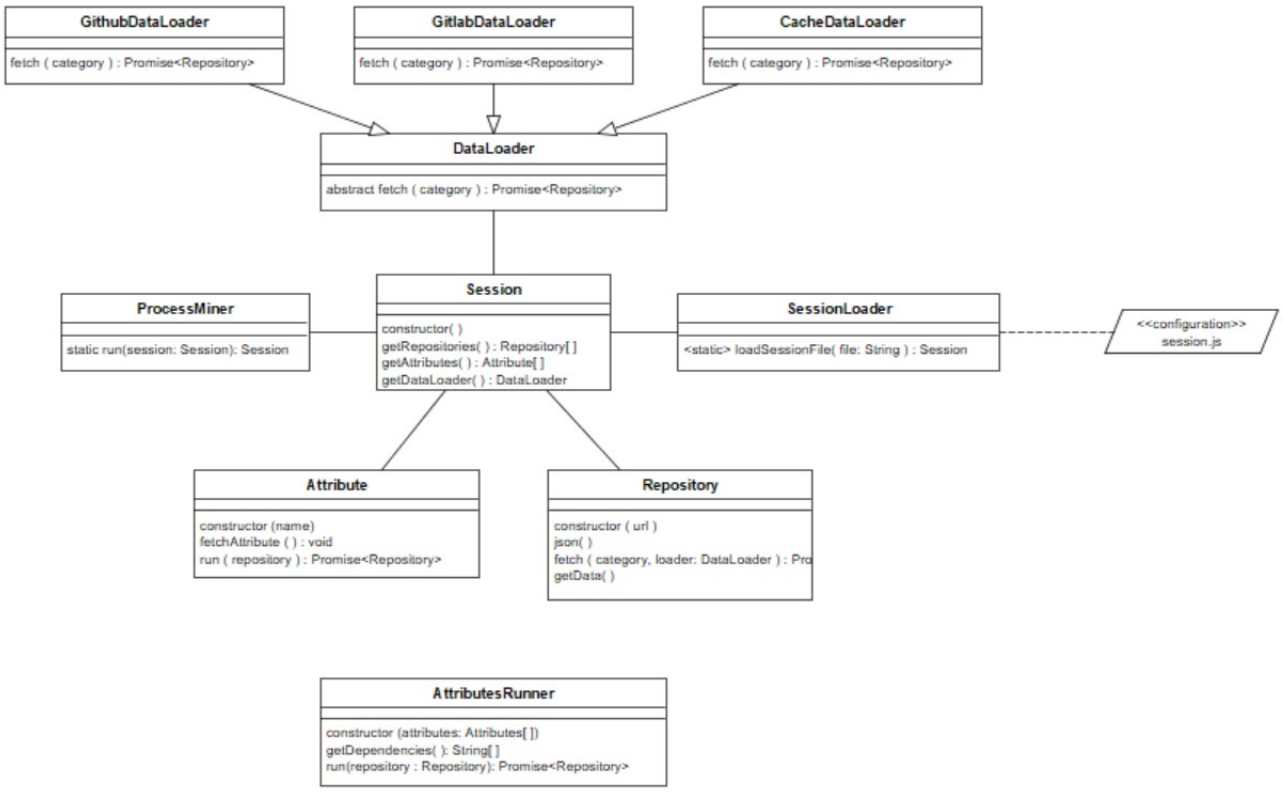

Figure 3.5: Classes representing the Process Miner, session and session data loading

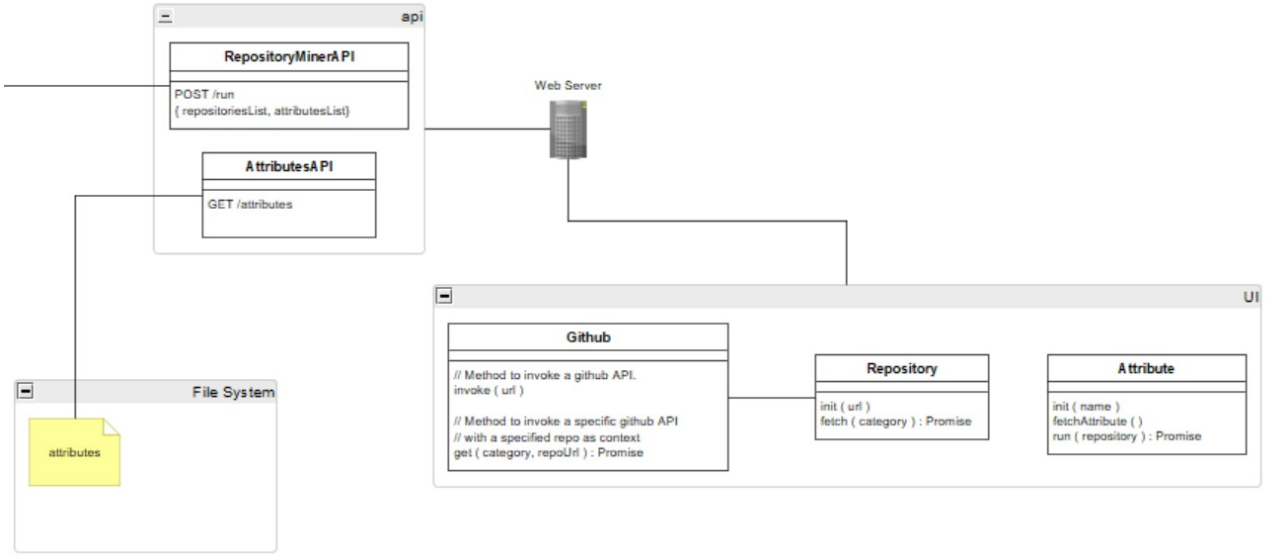

Figure 3.6: Web Server connecting to the rest API to render a UI on it

The application is composed of a few modules. They are represented by the following classes.

1 ProcessMiner - This is the main module that kicks off execution. This class is responsible for orchestrating the program flow to make sure the configuration file, attributes and repository data are loaded before the Process Miner runs analysis on the repositories.

2 SessionLoader - The module responsible for reading and parsing the configuration file. This class is responsible for any transformations required for the configuration object.

3 DataLoader - This is an abstract class that is used for representing a

repository data loader. This is useful in representing various sources of repositories, for example, Github, Gitlab, etc. Sub classes of this module help in actually loading and analysing the repositories. These subclasses can be supplied externally and can be used as a plug-and-play module through the configuration file.

Apart from these classes, the data structure classes are:

1 Repository - Used to represent a repository from any source. It does not contain any predefined fields and can be filled with any data of any data structure. However, the attributes receive an instance of this class as input and they operate on this object with the assumption of some data structure. So, that data structure has to be maintained in order for everything to be working. That structure is defaulted to the data structure returned by the GitHub API [1]. This is linked to the data loader module in the sense that the output of any kind of data loader has to conform to what the attributes are expecting. This is possible because every git repository essentially has the same data within it, its just the names of those pieces of data that changes based on the hosting platform. The actual data loader class will be responsible for transforming the data to the right structure for this class.

2 Attribute - Used to represent an attribute within the system. An attribute is a runnable class that receives a single repository as input. The attribute's run method is invoked again and again for every repository

to be processed. So, the attribute really just needs to worry about the single repository that's passed to it. Now, again, for the entire system to work, all the individual modules have to agree upon the same structure for representing a repository. When the repositories are loaded from their respective data loaders, they have to be transformed to the right structure so that the attributes do not have to worry about differentiating between different structures.

3.3.3.4 Architecture

The process miner application uses a plug-and-play architecture. Things are loosely coupled from each other but are tied together using a session. Any running instance of process miner is represented by a session. The session object is passed around to the different modules within the system. A session by itself is a representation of the list of repositories to be processed, the list of attributes to run on the repositories, and the data loader to use. The attributes work as a plug-and-play architecture. A session object contains the list of attributes to run by name. The process miner then, at runtime, loads a Javascript file with the name of the attribute and uses that to process the repository.

3.3.3.5 Attribute Use

Attributes are small programs that run on the repository given as input and produce a result which is a measurement of a specific metric that the

attribute represents.

3.3.3.6 Data Loaders

1 The GitHub Data Loader uses the GitHub API v3 to load repository data. The structure of data from the GitHub API is used as the common data structure for a repository across the application which means all data loaders would need to transform their output to match this format.

2 Cache Data Loader loads the repository from the cache directory. It expects a JSON file. The JSON file itself should be generated correctly with all the required fields as per the standard format established. Of course, this work can be done with the individual data loaders too. For example, the GitHub data loader stores a copy of the repository data in the cache directory. Similarly any data loader can cache their output by just setting the 'shouldCache' flag on a repository instance to true. On setting that flag to true, every-time data is fetched from that repository, it saves a copy to the cache.

3.3.4 Process Miner - Output

Attributes	Team 1	Team 2	Team 3	Team 4
Documentation	0.25	0.189	0.157	0.097
Continuous integration	0	0	0	0
Community - Core contributors	3	1	3	3
History - Commit frequency	30	15	15	15
License	0	0	0	0
Issues	0	0	0	0
Stars	false	false	false	false

Figure 3.7: Table comparing Process Miner outputs of repository of each team

Figure 3.7 compares the outputs of Process Miner when run on the 4 repositories of each team. Each of the individual repository outputs can be seen in figures 3.8, 3.9, 3.10 and 3.11.

We can see that Team 1 had better code comment documentation than the rest of the teams. As discussed in the previous sections, we evaluate Documentation (code commenting) by calculating the ratio of comment lines to the total number of lines in the source code.

Continuous integration (CI) is 0 for all teams since none of them used any CI tools for their projects.

One of the most crucial aspects of software engineering is collaborative software development and that the development of a software system involving more than one developer can be considered as an instance of collaborative software engineering [22] [24]. We can gather evidence of collaborative development in a team from its presence of a developer community [24]. Just as Reaper did, we calculated core contributors as the set of contributors whose total number of commits to a source code repository accounts for 80 percent or more of the total contributions [24]. The Process Miner output shows that in 3 teams, there were 3 out of 4 core contributors, which perhaps points towards a more even distribution of coding work in the team. We shall be able to find out more details about this and if this is true from our interviews with the teams. Team 2 has only one core contributor, which indicates a not very collaborative software development. Once, again, we shall gain more insight into this in our interviews with the team.

It is often stated that a source code must undergo regular and continual changes so as to avoid stagnation [16]. Committing regularly may also be considered as evidence of a more evened out software development with commits happening over regular periods of time and not just before the delivery deadlines. Thus, commits can be used to assess source code change and development. We calculate Commit Frequency by calculating the average number of commits per month [24]. Team 1 had a larger commit frequency than the other teams. It is interesting to note that all the other 3 teams had the same commit frequency. We discuss this in detail in our Analysis (Chapter 4).

None of the teams logged any issues on GitHub and since these were student course projects, there are no stargazers for them. We shall analyze these outputs with further details in our Analysis part (Chapter 4) in connection with the interview analysis and artifact relevancy analysis.

Figure 3.8: Process Miner output of Team 1 repository

Figure 3.9: Process Miner Output of Team 2 repository

Figure 3.10: Process Miner Output of Team 3 repository

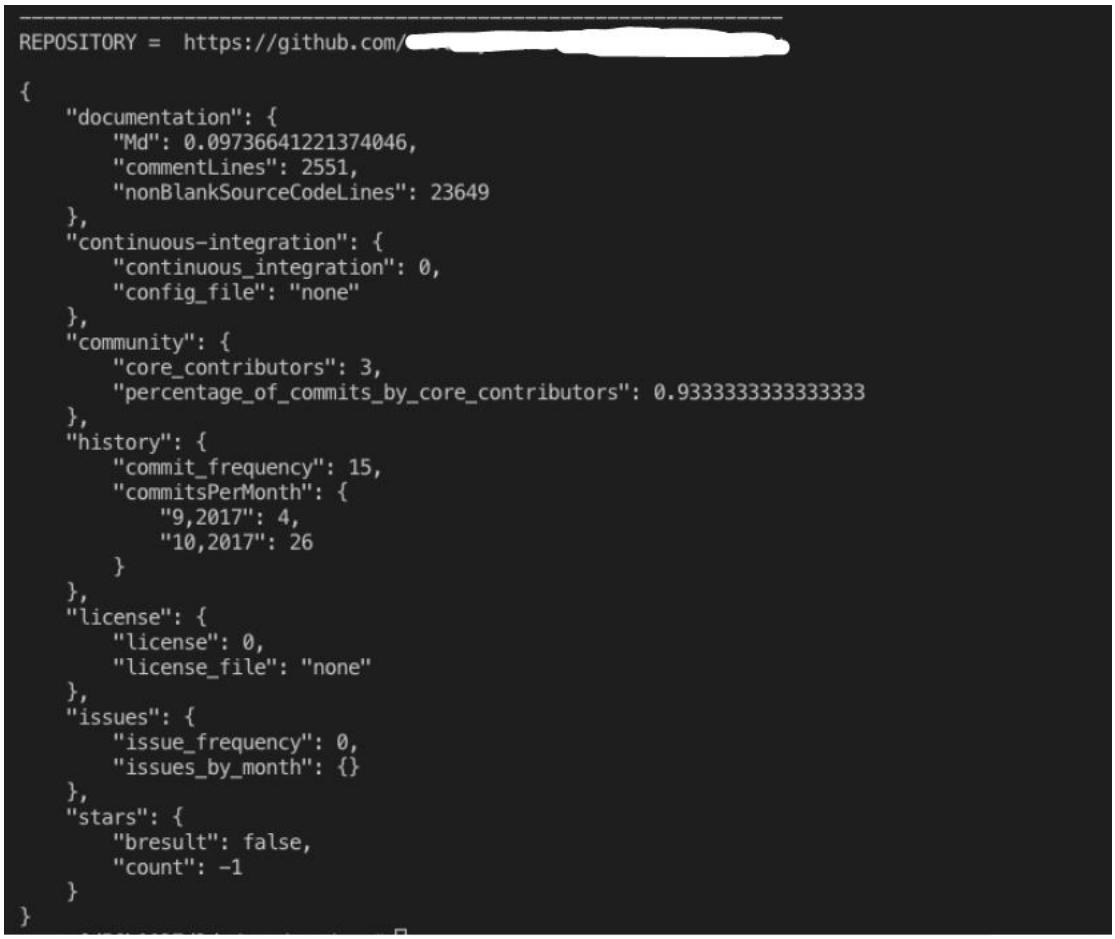

Figure 3.11: Process Miner Output of Team 4 repository

3.4 SE Student Interviews

Research on SE processes are incomplete without insights from the people executing those processes. To fully understand the gaps between what students learned in class about SE processes and what they actually implemented in projects, it was important to understand the reasons behind their

decisions and their opinions in addition to analyzing the outputs of Process Miner. Using Process Miner, we get quantified data, but it to qualitatively analyze this data, we also needed to interview students who worked on the projects we were researching on. For example, Process Miner output may show that a team has primarily 3 core code contributors in a 4 member team, but only on interviewing with the team can we get an idea as to why this is the case.

3.4.1 Observations from interview outputs

From our interview analysis, we were able to make a few observations about the students' opinions and thoughts on the SE process they followed. For Team 1, we were able to interview three team members, for rest of the teams, we were able to interview 2 team members. In our study, we decided to focus most on the below points. We shall discuss our analysis on these topics in detail later on in our study.

3.4.1.1 Distribution of work

a **Team 1** - We noticed that Team 1 fared better than the rest of the teams when it came to contribution of work from each team member. The other three teams felt that they did not have a very evened out distribution of work.

When we asked Team 1 about the way they divided their work, the team members said that Teammate 1 had more experience in working

in software development teams in an Agile environment and also had more knowledge in the technology stack provided. So, Teammate 1 took up the role of a 'Lead' and the rest of the team allowed that to happen without feeling threatened which could have been the case since they were all classmates and hence, such decisions can lead to ego clashes.

Teammate 1 discussed and spent quality time with the rest of the members during the Elaboration phase and helped everyone understand the process, the project, the requirements, and the designs and documentations. The whole team then contributed more or less equally to design.

When it came to the development, the lead once again took charge, and though the requirements were broken down into user stories and each member had user stories they were responsible for, the lead went ahead and tried taking some of the more complex ones for themselves and also helped the rest of the members with bugs and technical issues.

For testing, the Lead said that since they did a large part of the development, it was ensured that the rest of the team contribute more to the testing. This way, not only was there less bias during testing, the team had a evened out distribution of work throughout and no team member felt that they contributed less or felt overwhelmed with a ton of work.

b **Team 2** - Team 2 believed that they had primarily two core contributors who worked the most. During our interview, both the teammates (Teammate 1 and Teammate 2) stated that there were no clear roles defined or followed by the team and that one of the teammates (Teammate

3) was very knowledgeable and had a lot of experience working in Agile and hence decided to take up most of the work.

Initially, there was some quality time being spent over discussions among team members, but later on, Teammate 3 mostly did a large part of the code implementation work. Teammate 1 felt that one of the reasons there was uneven distribution of work was that Teammate 3 worked very fast and it was difficult to catch up with him/her and that often this teammate would also start doing the work which were responsibilities of other members and perhaps make the other teammates feel a little incompetent.

Teammate 1 also felt that Teammate 2 and Teammate 4 did not contribute as much as they could have for two possible reasons:

a they could not keep up with the progress as they may not have had much coding background or knowledge, or

b they were not much interested in the project.

Interestingly, Teammate 2 had an opinion that was contrary to the reasons provided by Teammate 1 above for not being called a core contributor. Teammate 2 felt that he/she did contribute enough to the team in terms of documentation, some of designs, and testing, but that many people do not feel that it is considered contribution and instead feel that only coding implementation and testing are considered work contribution.

c **Team 3** - Team 3 also felt that their work distribution was very uneven almost throughout the Elaboration, Construction and Transition phases of the process (Inception phase has been done already for the students)[27]. Team member 1 felt that the first Sprint saw a more even distribution but as the work progressed, it got very uneven and felt the primary reason behind this was the lack of knowledge in Java, and that the course did not teach the functionality of what they needed to do technically, or coding aspects or Java Spark and was more oriented towards following an SE process. Hence, there was a learning curve which took a sufficient amount of time. The team also felt that the immensely mismatched skill-set impacted their work distribution negatively a lot as well.

d **Team 4** - Team 4 had very similar opinions about the process and project as Team 3, with primarily one to two people who did the most work.

3.4.1.2 Steep learning curve of technology stack provided

Each team spoke about having somewhat of a steep learning curve with some of the technologies provided and a short span of product delivery time which impacted their adherence to the SE process they were using. They stated that Java Spark took a little extra time to learn because

a Spark was a completely new framework for all of them and,

b They could not find much documentation for Spark online or anywhere

else.

This was an issue because they were SE students having to work on various other courses and hence they could not give this project the dedicated time and priority that they could provide if they were working in an organization on a project. The extra time to learn and use this framework with the project to be delivered in a span of few sprints within a semester was, according to most of them, a little challenging and hence they mostly focused on delivering their work and sacrificing some of the quality practices they would have adhered to had the more time.

Some of the team members stated that it may have helped had they been given a simpler project to work on with the focus more on the SE process and less on the software product.

3.4.1.3 Code commenting and other documentations

Teams spoke of the importance of documenting their project, and appreciation for code commenting. But all teams except Team 1 stated that they did not have sufficient code comments in their project. The reasons behind this were primarily stated as:

a since these were student projects, the appreciation behind code comment importance to have a product that is readable and maintainable, were far less than what they have now.

b with little time to deliver the product, the focus sometimes got diverted from following process best practices to delivering the product.

Interestingly, all teams did say that code commenting is now considered very crucial for them and they do follow it in their work now.

All the teams stated that they did do the documents that were required of them from the course.

3.4.1.4 Communication and collaboration among team members

All teams believed they have had proper and sufficient communication and collaboration between the team members but three out of four teams were not able to resolve team issues. Teams 2, 3 and 4 had issues with team members feeling that the work distribution was not at all even, or that teammates worked very fast or slow for their liking, or that there were mismatched skill-sets, etc. But when asked if they had collaboration issues, they mostly stated no. They felt they had sufficient communication amongst each other, even though most of them stated they did not bring up these issues in team meetings or during team retrospection. When asked how would they resolve these issues if they were facing them now, we did not get very clear answers.

3.4.1.5 Lack of understanding of SE Process Quality

It was interesting to note that most teams could define software product quality but could not define or provide a lot of information on 'software engineering process quality'. Most teams connected SE process quality to soft-

ware development best practices such as using SOLID principles, God classes, etc, which ultimately do help in increasing product quality, but are not really the reasons for SE process quality.

3.5 Word Frequency and relevancy of project artifacts

We used Atlas.ti on the artifacts created by the four teams for their projects to get word clouds for each of their supporting project artifacts that they created. We did this to check if the content of each of these artifacts are relevant to the artifact names. We were also able to see the various words, their frequencies, and relationships with each other. Relevantly and appropriately named artifacts would indicate a clear understanding of the documentation in the artifacts. This would also indicate appreciating the importance of creating these documentations so as to make the project readable and maintainable.

3.5.1 Team Artifacts

We were given access to some of the artifacts that each of these teams created for their WebCheckers [13] projects. We have one shown in figure 3.12 which is a Project Design Document (PDD) for Team 1. A PDD is an early stage document that should consist of a description of the project, domain, goals, outcomes, deliverable, features, functionality, etc. [11]. In our word cloud, we see most of these words, or their synonyms. Looking at the word cloud, we can state that it is about a game, which has players, users, usernames, win/loss and has pieces and moves. We see domain and

functionality, classes, diagrams and renderings etc. These all are quite relevant to a PDD and thus this word cloud shows a clear relevance of words and their relationships with each other,when compared with the artifact name.

Figure 3.12: Word cloud generated from Project Design Document for Team 1

To see all the word clouds generated by the supporting artifacts of each team, go to Appendix A. We analyzed these artifacts and found that each team documented pretty extensively. The reason behind this is because the course required these documents, but we cannot assume that this would have been the case if the course did not specifically state the artifacts that needed to be created. We also found that most of the artifact names were relevant to their content. This shows that the students understood the requirements for each of these artifacts. This would naturally allow us to assume that the teams had

an appreciation for the reasons these artifacts were being created. However, this may not be the case here, since the students were informed and required to create these artifacts and hence we cannot draw the conclusion that SE students have an appreciation for the importance of artifact documentation without conducting our research on projects where they do not have such requirements explicitly stated. We created word clouds for some of these artifacts from each team.

Chapter 4

Analysis

From the outputs received from Process Miner, the interview answers and the manual analysis of each team's artifacts, we were able to make some observations.

4.0.1 Software Process (Agile) understanding

We observed that each team was able to identify the software process they were following and understood its phases. They spent quality time on the Elaboration Phase, understood the requirements, broke them down into user features, events, user stories, created various design diagrams (domain diagrams, class diagrams, sequence diagrams, etc) that would help with project understanding, developed the product, tested, etc in increments and iterations. They had a minimum viable product (MVP) upon which they built all of this.

But it is to be noted, that the requirement of this course was to understand the fundamentals of software engineering and to let students have an appreciation for the importance of following a SE process, along with observing its nuances and understanding the challenges that are faced when following an SE process for a product development. Thus, a large part of the artifacts

created, and phases followed were because it was a course requirement. However, the main goal of the course, to understand and follow an SE process was achieved. From our analysis, we found that students had quite a clear understanding of what they were doing and why (from their interview responses), when it came to the SE process they followed.

4.0.2 Importance of having defined roles

It was important for each team member to work on each phase, but in our analysis, we observed that the only team which was able to get a more even distribution of work among team members, was a team which had a member having a clearly defined 'Lead' role. That team not just had better collaboration amongst each other, but were able to adhere better to the SE process. Each member contributed in various ways - each participated in all activities, but each also worked more at the activity they were more comfortable with, - some provided more contribution to design and documentation, while some implemented the code more. User testing was avoided by the developers so as to avoid bias and was primarily done by teammates who had somewhat lesser contribution to coding. This way, no team member felt they got overwhelmed with work, and no one felt they weren't contributing enough because of which the team morale stayed strong.

It was important to note, that for this team, there could have easily been ego clashes of declaring someone as 'Team Lead' since they were all classmates, but the personalities of both the 'Team Lead' and the rest of the

team members did not allow for such clashes to happen.

For this team, Process Miner showed that there were three core contributors when it came to committing the code. On interviewing, the team did state that there were primarily three members who worked a lot on the coding, with one of them doing a little extra work and working on the more complex logic. Thus, the output of Process Miner did match with their interview answers when it came to code contributors. The team did state that they did not have any 'fake users' (members committing code on other's behalf) and that each member committed their own code, thus validating the output of Process Miner.

Thus, our analysis raises the question if having defined roles is crucial for a well balanced and well collaborated student team. This may help in clearer allocation and ownership of work.

It also raises the question of understanding the personalities of each team member better and how to collaborate and involve different personalities in a team better. Perhaps, once a team has been declared, it is important to focus on the personality traits of each teammate and understand how to collaborate with them individually and as a whole team better. As we know, there is a wide range of personalities between highly extroverted, and highly introverted, and communication and collaboration can get difficult between different personalities. We need to keep in mind that the people are equally important part of the 4 Ps (the others being project, product and process), of an SE process.

We also observed, that with other teams, often team members felt they did not contribute enough, if they worked on other aspects of the project, but less on the development. It is important for software engineers to note that, though the development is a very crucial aspect of a software product, it is not the only crucial aspect. As the course tried explaining, the SE process was divided into various phases, with each phase having many activities, and each of these are important for a well designed and well developed software product. Thus, a software designer, developer, architect, tester, all have equal contribution to provide. A team where each team member worked on each phase, but contributed more where they were the strongest, and to acknowledge those contributions, would help a team develop a software product better and ensure better collaboration and team morale. We feel a team having clearly defined roles of Design Lead, Requirements Lead, Development Lead, Testing Lead, etc, where each Lead primarily works on their part while contributing a little less on the other activities, may help each team member feel they have contributed enough.

4.0.3 Steep learning curve

Each of the teams, though overall happy with the technology stack provided, did state that some of them were complex and impacted their SE process adherence. They specifically felt this with Java Spark. The teams were not familiar with the technology and stated that they could not find much documentation for it. Some of the teams stated that perhaps a week

extra time for tutorial on Java Spark, or a different language/framework may have helped with spending less time on learning the technology, and more on understanding and focusing on the SE process. Some team members felt that perhaps an easier project, with more focus on SE process and less on the software product may have helped as well. They felt that with other course pressures, it may have helped them follow and implement the SE process better, if they had a lower learning curve. Some of the students also stated that they got confused between the priorities of delivering the process versus delivering the product, with both being important, but the requirement of understanding the SE process being more crucial for this course and overall software engineering.

This definitely speaks to two noticeable points - for students to be able to focus and implement SE process better, it is important to provide a technology stack that does not have a very steep learning curve, and maybe some more time or simpler products to develop with explicit information about the focus being on the SE process more than the product delivered.

4.0.4 Documentation opinions

The teams overall felt that it is very important to have code comments to ensure understanding and maintenance of a software product, but it is important to not go overboard with it. Most teams felt that it is sufficient to have function definitions comments and complex logic comments and more than that would be going overboard and is unnecessary. The teams appreciated

the importance of commenting when it comes to code readability. But most of the teams also stated that they did not have sufficient commenting for their Web Checkers project since they were struggling with the Sprint deliveries.

One of the teams stated that they believe for a project like Web Checkers, a sufficient range of code commenting would be between 15 percent to 25 percent range, and interestingly, their code comment ratio was 25.22 percent, and it was the highest among all the teams. It was also important to note that the other teams felt they commented very poorly for their projects and their code comments ranged between 9 percent to 19 percent. Since the projects were the same with more or less same logic being used, this makes us notice the difference in opinions between teams considering poor and sufficient commenting range.

The students did state that the importance given to code commenting in class can be a bit hyped can become somewhat impractical to follow completely through when working on projects.

Perhaps it is important to let students grow an appreciation for code comment importance by letting students reverse engineer a software product with less or no comments for its code. The observations by students stating that commenting class definitions, function definitions and complex logic was also something interesting to note.

Some of the students also stated that some of the documentation are as important as is stated in school, but some are not. One such example provided

was the sequence diagram which some students stated were not as important in industry projects as say, a domain model.

4.0.5 Communication

We observed that most teams faced collaboration issues related to work distribution, and when asked if they had sufficient communication, they stated that they did. However, when we asked them if they brought up the uneven work distribution issues in their team meetings,they said they did not. Most of the students showed a lack of willingness to communicate and resolve conflicts.

Most teams also felt that the daily stand-up is a little overboard and does not need to be done every day. They stated that a stand-up every three to four days should be sufficient.

It is important to let students know the importance of communication and collaboration and there needs to be more focus on how to collaborate to resolve issues with team members having various personality traits.

4.0.6 SE process quality

It was interesting to note that when asked about software product quality, most students were able to give some definitions of what they feel software product quality is, but most could not state what they think SE process quality is.

It is important to let SE students understand that each phase of a SE process impacts its quality.

Chapter 5

Results and Discussion

Through our analysis, we found our answers to our research questions.

5.0.1 Do SE students adhere to established software engineering processes in their projects?

We found for our demographic of students this holds true, and that they did adhere to an established SE process in their projects. However, we need to keep in mind that these students were provided an SE process to use, and it was a course requirement to incorporate the different phases of the process. Hence, we could not find an answer for this question yet. We shall be working on a larger demographic of students who have worked on varying projects where it is not a requirement for them to incorporate any established SE process, to be able to answer this question more decisively. This is part of our future work.

5.0.2 What are the similarities and differences between SE students' opinions on the software process phases they believe they have used, and the actual implementation of them in the projects?

We found that the teams followed Scrum to a large extent, and that is what was required for the course. They designed, broke down requirements into user stories, understood team velocity, did poker planning, had daily stand-ups, collaborated using the various tools provided and face-to-face meetings, set up product backlogs, implemented, tested, delivered in Sprints of two weeks.

The students felt they had sufficient collaboration among team members, which was not the case for most of the teams, since there were pretty uneven distribution of work as well as other collaboration issues, which were never brought up amongst the team members or with the manager for resolution.

The teams created several artifacts which were relevant to their content, expressing an understanding, and they did state an appreciation for the artifacts to help in product maintenance in their interviews.

The students understood the importance of having a well designed SE process to deliver a good quality software product, but could not connect the SE process phases to its quality. When asked about the quality of an SE process, we mostly got confused or no responses.

Most teams stated that they had quite uneven distribution of coding

contribution, with primarily one or two members doing most of the code contribution work. All teams except one stated that each member committed their own code, thereby the concept of 'fake users' does not apply to those three teams. And yet, Process Miner output for all team repositories except one showed as 3 out of 4 team members were code core contributors.

5.0.3 What are their views and opinions about theoretical classroom knowledge versus the feasibility of applying said knowledge when working on real projects?

There are several views and opinions discussed in our Analysis. Some of them are as follows:

1. Code commenting is important, but primarily at the level of class definition, function definition and complex logic explanation. It can get difficult to comment code when working on projects and can become impractical if too much commenting is required. They also feel that when a project, which is not overtly complex, requires too much commenting, then perhaps it is important to make a note of it, and make the code simpler.

2. Activity diagrams, sequence diagrams, use case diagrams, are not as important as say, a domain model, or a class diagram.

3. Not stated directly, but from the interview responses, it could be gathered, that a lot of the students felt that the primary responsibility of a

Software Engineer is to be able to code, and the other contributions are not as important as developing the product.

Chapter 6

Threats To Validity and Future Work

Qualitative analysis brought a ton of information about the human aspects, decisions and opinions in our research, but it may also allow for certain biases to stem in. For our research, we could not interview all four team members within each team, which may have resulted in more understanding of the decisions and opinions of the teams, but to avoid bias, we ensured we interviewed at least half the team members from each team. Some of the team members did not remember all the answers to the questions since this interview took place 2 years after they had worked on the project, and had the students remembered the answers, it may have resulted in a deeper understanding of our interview data.

Also, we recognize that our data sample of four teams is comparatively quite small and also the teams working on the same project with a clearly defined SE process to follow may result in a not so diverse set of answers, but our primary goal was to understand the gaps between learned knowledge and its practical implementation, and we feel we have been on the right track for it because we were able to identify a few of the gaps between established software engineering process and its actual execution using these 4 projects.

For our future work, we would like to work with a larger data set of students who are working on different projects.

For the Process Miner attribute History, we had customized it to calculate commit frequency based on commits per month. But to track student projects better, it would make more sense to track commit frequency per one or two weeks since student projects are supposed to be delivered within a semester (which is usually a 4 to 5 month term). Thus, for our future work, we would like to customize History attribute in Process Miner by calculating the commit frequency weekly or biweekly.

We want to work on fine tuning and expanding our attributes for Process Miner to get more relevant data from a software repository using GitHub APIs which allows for extraction of a ton of information from a GitHub software repository. For example, GitHub APIs allow us to extract information from a repository about users who are members of an organization [2] as well as users who are outside collaborators of an organization [3]. There are pull request information [4] that can be extracted such as pull request reviews, comments, etc. There are search topic options [5] using text matching to pull out information about any topic that we want to search for. One of the most important topic that we would like to extract information for would be testing. Thus, we would like to make use of the GitHub APIs to work on expanding our Process Miner attributes.

We feel that Process Miner would have been more useful when working on a larger data set of SE student projects having different SE processes.

It would be quite interesting to interview SE students from other universities which has SE departments and to see what information we can find from them that can help bridge the gaps between learned knowledge and implementation further. We would also like to work on Process Miner to make it more user friendly, has more attributes that can extract meaningful data, and to enable it to be able to access private GitHub and GitLab repositories for which it has access permissions, since as of now, it can only access public GitHub repositories.

Appendices

Appendix A

A.1 Software Engineering student interviews

A.1.1 Team 1

A.1.1.1 Team Member 1

1. **What Software engineering process are you using for your project?**

    a. We used Scrum. We used this methodology because we were trying to find out how to apply this methodology and it was given to us (course requirements).

2. **What are the SE practices/activities that you are following?**

    a. We were using several communication tools such as Slack, Trello, etc. For e.g., we used Slack to have daily stand up meetings. We also did weekly maintenance so as to divide the work that each team member was going to tackle, because each week we had to implement new parts of the Sprint, so we did meetings specifically to know who was going to do what. Am unable to remember more than that.

3. **What kind of testing are you performing?**

    a. We did Unit testing. We also did this class activity where we interchanged our projects with other teams and we had to test each other's project and to see if there are any flaws in their project and that is another kind of testing we did.

4. **Did you document your testing on GitHub?**

    a. We documented the source code. We may not have explicitly said that in the documentation, but we used code coverage (I think with Mockito) and we documented that. They maybe on GitHub in the releases section. Each time we had a release, we had to update our documentation and we kind of added screenshots into those documentation to showcase the code coverage for that Sprint.

5. **What all SE quality practices are you incorporating in your project?**

    a. I don't remember a lot, but since we were using Scrum, we did user stories based on requirements, and based on those user stories, we created a version of that product for that Sprint in accordance to the requirements by following the user stories.

6. **How much importance do you place on commenting your code?**

    a. For WebCheckers, as a developer, I have mixed feelings. We always say it is really important, and it is really important mainly if we were working in teams, and we were working in teams for WebCheckers. So, if I had to work on some feature that our teammate was working on, I would like to see some documentation that can guide me on what was done before I start working on it. But personally, though I know it is really important, but we had a timeline, and we are rushing to deliver the Sprints in these timeline, so we don't necessarily give the importance that it has. When you have in a balance, having to finish Features and deliver them, versus, doing them the way that you would like it to be documented, you would choose to just finish and deliver it, because end of the day you need to deliver to get a grade. For me, code commenting should not be at the level where you are explaining Getters or Setters or Constructors, but at the level where if a method is doing a complicated task, you would want that method to be documented, so that when the next person is coming to your code, they can understand what you are doing and why. In a gist, code commenting should not go overboard defining everything, and also, code commenting can be tough to do when there are tight deadlines.

7. **What kinds of comments does your project have?**

a. I think we did a lot of code commenting because that was a requirement for the project.

8. **How much importance do you place on documentation for your project?**

    a. It is very important because now if I go back to WebCheckers project, I am going to know exactly what we did, and how we did it, and why was specific parts of the code are the way they are, and why we took X or Y decisions. While, we were working in the project, we did not think it was that important, but now that I am a year or so away from that project, if I need to go back to that project, it would be easy for me to understand the project because of the way it was documented, and so it is really important. I feel it should be done and was pretty feasible to document the project.

9. **What kind of documentation does your project have?**

    a. We have source code documentation. We also have documentation on GitHub. Each time we did a release for our project, we attached a document in which we explained all the features that contained that release.

10. **How has the work been divided between team members?**

    a. If I have to say in percentage, one team member about 40%, another about 20%, and then the rest between the other two members.

11. What do you feel is the contribution of each person when it comes to

    i. **Design**

        1. I don't remember much, but overall we had a very cohesive team setup, I was very happy with the team we had. At the beginning, each one of us were very communicative about the design parts and how we wanted to add them. I remember we were doing these meetings in the team rooms, and everyone was given their opinion and adding their features. I think that was the first thing that we did, and it was a very evenly distributed workload amongst team members.

    ii. **Documentation**

        1. What I remember, documentation was mainly one or two members who did it at the end after we finished the project, we decided to tackle that part of the project in that delivery. We grabbed the source code and added all the documentations that we added in that project. So half of the team members worked on the documentation towards the end of the project.

    iii. **Requirement Gathering**

        1. The whole team worked well in this.

    iv. **User Stories/Use cases**

        1. The whole team worked well in this.

    v. **Coding**

1. There is always someone who works way harder than the rest of the team. In my team, we had this one member, who was very fast at understanding the things that needed to be done, and also developing the things that needed to be done. For example, I was assigned two user stories, and he was also assigned two user stories, but by the time I was getting started with one user story, he was finished with his two user stories. So, he was like let me help you with the user stories everyone else has. So, he worked on a lot of the hard parts of the project. Mainly he was very efficient and fast. I would say he had the benefit of having more knowledge than the rest of us in Web development. Because, in this course, we were also learning about web development along with how to use a process as we worked on the project. He had a lesser knowledge curve than the rest of the team members. He was the one who worked the most on the features of the project, and then I would say, I worked the second most on developing the features. Our third teammate did some work. Our fourth teammate did not work as much as the rest. Like our fourth teammate mainly worked on the design and the documentation that we had to deliver. We didn't talk much about the fact that our fourth team member did not work on developing the features, so I wouldn't know if he wasn't, because he was busy with other courses, or was it just a learning curve. I do feel I could have contributed more if I had more knowledge on (or worked with before) some of the technology (Java Spark, etc).

vi. **Testing**

1. I would say, one person did around 5% of unit testing, another person 15%, and the rest between the other two. We really didn't talk much about why someone did not work. We did not communicate amongst each other, if someone was not understanding or lagging behind. Our concept was if someone was not working as much or as fast, we will help with his/her part and do it.

vii. **Issue Management**

1. If I had to implement a feature, I would have to deal with the issue of merging that part with the rest of the project. I would talk with the primary contributing team member on WhatsApp, or Slack, about any technical issues am struggling with and he would usually know and help me solve it. I also remember occasions when three of us were talking amongst each other about some technical issues we are struggling with.

12. **How did you collaborate with each other?**

a. We used Slack mainly and WhatsApp. We used weekly in person meetings as well.

13. **What are the collaboration issues that you have encountered?**

a. We did not face any collaboration issues, not really. I don't remember much. In my team, people were not willing to ask. If I don't ask, you don't' tell. So, I won't ask as to why you are not able to contribute more, so there won't be a response. It was more like, if you have not finished this, I will help you with the part. We just wanted to finish the work. I wouldn't say the tools were not appropriate. We probably might have had better communication to resolve team contribution issues had we met more in person than remotely. But I think it was fine, I think I was ok with the communication tools given to me and the balance that was maintained to meet in person versus remotely.

14. **How did you work on resolving any collaboration issues?**

    a. Not really applicable.

15. **What type of test artifacts are there on GitHub?**

    a. I don't think we have any testing artifacts. We had a Sprint word document, and the releases documents, in which we added screenshots of Mockito and code coverage.

16. **How do you define quality?**

    a. I would define Software Quality as the level of closeness that the source code and the product has to the requirements, both functional and non-functional requirements and the degree to which those couple to each other, that is the requirements and the actual product closeness.

17. **What are you doing to maintain that?**

    a. I believe we implemented at least one or two design patterns into the project. We were given the SOLID principles, but I don't think we applied those in our project.

18. **How do you handle your releases?**

    a. We had a semester and we had to divide three sprints in that semester. We were very evened out in our commits and merges and releases, mostly thanks to our most contributing teammate who made sure we were not finishing up work very close to deadlines.

19. **Did you document issues on GitHub?**

    a. We didn't.

20. **Did each one of you commit/merge your code?**

    a. Yes. Maybe once or twice, where we did team programming, where one person was committing.

21. **Does tool help or hinder process adherence?**

    a. I do feel I could have contributed more if I had more knowledge on (or worked with before) some of the technology (Java Spark, etc). I didn't know anything about Java Spark Framework and some of the other technologies that we used, and that was a learning curve and that slowed me down. At the beginning, things got really slow, and as we progressed, I got used to the tools, it was still weird, but kind of OK.

22. **What programming languages did you choose for your project?**

    a. Java.

23. **What were the main reasons for choosing the language you chose?**

    a. It was given to us.

24. **Does one language have any specific benefits over the other when it comes to incorporating SE practices.**

    a. I don't think the language matters.

25. **Code reviews?**

    a. To me they were completely unnecessary. By the time, we did the code reviews, it didn't matter because we had already merged those features. I wouldn’t be able to take their review into account to improve my code (since that was the purpose of the code review) because the merging was already done.

A.1.2 Team 1

A.1.2.1 Team Member 2

1. **What Software engineering process are you using for your project?**

    a. Scrum.

2. **What are the SE practices/activities that you are following?**

    a. We had sprints, after every two to three weeks, we were supposed to present the work we did. While we were following the entire lifecycle, that included considering the requirements that were given to us. So, in our case, we were told we were supposed to create the WebCheckers, basically the designs were kind of given to us as well because I remember we had a picture and we were supposed to create something like that and then the implementation part and the testing after the implementations. So, yeah, the Sprint thing was one. We had stories, subtasks in those stories, and the stories were under epics. Considering all of that we allotted which person will do what and be held accountable for what story and what task. We completed a few stories in the first sprint, tested those stories, and when the next sprint started, we took more stories from the product backlog into the to-do list.

3. **What kind of testing are you performing?**

    a. Unit testing and then eventually the whole system testing.

4. **Did you document your testing on GitHub?**

    a. We were. I remember having a read me file in GitHub pages.

5. **What all SE quality practices are you incorporating in your project?**

    a. We were commenting the code for functions. Since we were learning about God Classes, we tried to keep the functions that were mostly used in the same class, so as to reduce coupling between classes.

6. **How much importance do you place on commenting your code?**

    a. It is very important, because right now if I go and look at the code, and there are no comments there, I think I would be confused. For me, it would still be fine, but for someone who has never looked at that code before, understanding it could be a problem because even though the variable names that we have used are readable, they don't always convey exactly what the function is supposed to do. I feel every function that you write should have a comment associated with it, so that whenever you read the comment for that function, you would know what the function is going to do. This comment should not be something that is very long that would bore the reader if the person went through it, but just concise and to the point enough that the person understands what the function is about.

7. **What all kind of comments does your project have?**

a. Function comments, Parameters that were used in the functions, datatypes of the parameters and the return types. I do not remember how much commenting my team did for the project though.

8. **How much importance do you place on documentation for your project?**

    a. Well, I think if a third person who has never worked on the application before is looking at the application, the best way to go about that would be reading the documentation and understand what the application does while for developers, since they have developed, they know about the application, it may not be that important to them.

9. **What all kind of documentation does your project have?**

    a. Yeah, we had a system requirements specification which jotted all the functional and non-functional requirements and the eventual goals of the project. I do remember that we did have a page explaining what the application did but I do not remember anything else.

10. **How has the work been divided between team members?**

    a. **What do you feel is the contribution of each person when it comes to**

        i. **Design** - we all participated in the design.

        ii. **Documentation** - each one of us picked up one part of the project. Equal contributions.

        iii. **Requirement Gathering** - Requirements were given to us.

        iv. **User Stories/Use cases** - Even distribution of work

        v. **Feature Listing** - Even distribution of work

        vi. **Coding** - One of the team members had more experience with programming and contributed more than the others.

        vii. **Testing** - Testing was more evenly distributed work. I was testing for the first time, so it was easier to learn by implementing it. Most of us were learning it and hence all of us were implementing it so as to learn better. The learning curve was not steep, and hence it was easy to grasp and implement it.

        viii. **Issue Management** - Whenever an issue popped up, we took care of it before proceeding with the rest of the code. We didn't need much collaboration in this, as we each took care of the issues that popped up in our part of the work.

11. **How did you collaborate with each other?**

    a. We collaborated mainly through Slack and some meetings and doubts were worked upon in the meetings, but mostly it was remote work.

12. **What are the collaboration issues that you have encountered?**

a. We must have faced some merge conflicts, but we solved it individually. We had a few times when not all the people in the group were available for meetings which delayed the process.

13. **How did you work on resolving collaboration issues?**

    a. We decided on one fixed date that everyone will definitely be there and there is no chance of anyone backing out. We didn't have much collaboration issues.

14. **What type of test artifacts are there on GitHub?**

    a. No artifacts on it.

15. **How do you define quality?**

    a. If the code does what it is supposed to do and at the rate/speed that you want it to do it at. The app should be usable, should not break down. You know what you are expecting from the app and the app should be able to deliver that.

16. **What are you doing to maintain quality?**

    a. I do not remember.

17. **How do you handle your releases?**

    a. Via GitHub. Every two weeks. We released whatever is completed in each sprint and worked on whatever remained in the next.

18. **Did you document issues on GitHub?**

    a. I don't think we did.

19. **Did each one of you commit/merge your code?**

    a. Yes

20. **Did tool help or hinder process adherence?**

    a. Trello helped a lot as it helped keep track of what we were supposed to do, what was done, what needed to be done, and what was in progress. I don't think any of us worked with Java Spark before, and so if the language was left to us to decide, it would have been easier. We could have come up with something new and that may have helped us in the process faster. I am more comfortable with Python and so I may have chosen Python. The testing tools did help us. Slack did help us as far as the daily stand ups went and we could work remotely and stay updated still, so Slack helped as well. Remote meetings everyday helped but in person meeting once in a while is also important.

21. **What all coding issues have you encountered?**

    a. I don't remember.

22. **What programming languages did you choose for your project?**

    a. Java Spark.

23. **What were the main reasons for choosing the language you chose?**

    a. They were given to us

24. **Does one language have any specific benefits over the other when it comes to incorporating SE practices**.

    a. Have not written test cases in Python. So I don't know if Python would have worked as well as Java Spark worked.

25. **Code reviews?**

    a. Yes we did. It helped improve our code. We were working on our part and didn't always know the kind of code the other person wrote, and just going over it gave us ideas to help improve.

26. **Any Feedback** - The Agile methodology has been an important thing to learn. I have started projects in a group where we did not follow a life cycle process and just started to code. It started fine but eventually it becomes difficult to manage. It is always better to document what you are doing, have roles defined and knowing who is working on what. It is important to know what all has been done and what all needs to be done. The development lifecycle gives you the opportunity to make sure that your code is implemented right, tested right, and that requirements are being met and that is very important.

Daily standups are maybe a bit hyped. It may be better to have standups every two to three days when each one has enough to speak about what they have done. One day seems like a short time to talk about what you have done.

If I have to do WebCheckers again, I would definitely use Scrum again.

A.1.3 Team 1

A.1.3.1 Team Member 3

1. **What Software engineering process are you using for your project?**

    a. We used an Agile process. It was what was dictated to us for use, but in general that is the process that most students at this level are comfortable with and that is what would have been defaulted to either way.

2. **What are the SE practices/activities that you are following?**

    a. Throughout the Agile process we followed the pretty standard Agile process activities such as virtual daily stand up meetings, we worked with story creation, task creation, epic creation, we implemented some code reviews, etc.

3. **What kind of testing are you performing?**

    a. So I think testing was actually the area where we followed the Agile process the least. We did do testing but all of it wasn't necessarily the formal pull request and testing as it is drawn out in class. We definitely had testing, but it was most like either somebody was testing their own code as a specific activity or if we were working in pairs, we would have someone review the code like a full on pull request. We were not able to follow the Agile process when it came to testing was primarily because of the time crunch. With just the way the semester was scheduled, there was a lot less time dedicated to the actual implementation of the project and the testing at that end level, and just because we were balancing this class and other courses and trying to just finish on time with concerns about doing well and getting a good grade, I think that is what my group ended up sacrificing in terms of specifically following the Agile activities just to get done on time. We mostly did end user acceptance testing. We would finish a feature, upload it, verify through playing on the client that feature worked correctly. I acceptance tested my own code and invited other group members to test my code. We did have some (two or three) pull requests on larger features as well.

4. **Did you document your testing on GitHub?**

    a. With the pull requests, yes. But not with the basic acceptance testing. It would just be committed and the documentation can be that it was not reverted, which is not really documentation, but yeah. I don't believe we had a specific testing document. We did maintain a suite of unit test, so we have a pretty robust set of tests for each of the layers of the application. It's not extensive documentation, but we have screenshots of code coverage and code complexity metrics.

5. **How much importance do you place on commenting your code?**

    a. I place pretty high importance on them. I think it's one of those nobody really wants to do, but is very important for the maintainability of a project. Any code I am working on, that I know, or think that other people are going to look at, I am definitely trying to comment. So, for this project, we made sure that every class and method had Java docs style comments and then any particularly complex logic had inline comments and that goes a very long way towards the maintainability and reusability (in the sense of other developers working with that code) of the project. However, I think that you definitely

need to look at your comment to code ratio. If you have three lines of comments explaining one line of code, that line of code is probably too complicated and needs to be split up in some way. I mean there is of course, the off chance that it is just that complicated and needs to be in sort of your edge cases, but for the most part, you want to find the correct balance between the amounts of comments in your file paired with the amount of source code. One of the things I try and remind myself when I am writing code, is that I don't need to over explain things, so especially in Java doc style comments, I try to keep them very concise, and assume that if somebody is reading the Java doc comments, that they have enough understanding of the system to have the basics down and I try and make the shorter comments more meaningful, to avoid comment overload. I feel there is, for your general standard project, there is, an appropriate percentage of code to comment ratio that we should be aiming to hit. It definitely does not mean every project, as there are certainly edge cases for certain projects where a lot of documentation is critical, or some projects for which a lot of documentation is not necessary. But for your standard project, there may be an ideal percentage range of code to comment ratio that we should try to aim for. For WebCheckers specifically, I would say 15% -25% is a good range to aim for. You can get away with less if it's done very well but.

6. **What all kind of comments does your project have?**

    a. Java doc style definition for all classes and methods and inline commenting for any particularly complex logic.

7. **How much importance does documentation have in your project? What all kind of documentation does your project have?**

    a. We have a design document that contains an executive summary, a glossary with acronyms, definition of our minimum viable product, the section of the application domain, we defined our domain areas, and our like specific details about those areas. We had an application architecture section where we summarized that, provided an overview of the user interface, diagram the flow of user interaction through the diagram breakdown of each of our specific tiers in the architecture, the details of any enhancements that we added, breaking down static and dynamic models for some of our larger subsystems, so we had a class diagram for the overall application, a sequence diagram for some of our critical processes, and then beyond that, our overall high level summary of the project. We did this mostly because these were what the class required. In a professional setting, like an actual project, I have not personally done this deep level of documentation, but I definitely have done parts of this for each project as necessary, and I think the importance of this is helping any developers that are coming onto the project to be able to understand this system. I think what we learnt in class is a little overhyped but I think that is necessary, because you really need to hammer home the importance of it, because it does have importance. I think there are some activities that are hard to replicate in a classroom setting that are valuable in the real world. One of the big purposes of documentation, especially when working on class diagram and the like is that, at the start, everyone comes to a shared, concrete understanding of what the system is. I think that is something that is much harder in the real world because you have so many different types of projects you could be on. Or you are working with different clients who have their expertise in a very specific information domain, and you need to be able

to understand that domain as a developer. So I feel, to get that shared concrete understanding from documentation is much harder in the real world and is also much more important in the real world. Hence, they are valuable to be taught.

8. **How has the work been divided between team members? What do you feel is the contribution of each person when it comes to**

    i. **Design** - I think this was relatively even. I think I sort of led the design process, which is a role I think am very used to and is a role that I have played in a lot of the group projects I have worked on. But, I think in the design phase, it is pretty manageable to get good effort from everyone and get an equal amount of input from everyone and my group was a pretty good group. I definitely feel like I took charge, but got a relatively good effort from everyone in the team. The learning curve for me was not as high as for the other teammates and that definitely made an impact as I was facing an essentially no learning curve for this project. I had worked on Agile projects before so I was familiar with the process as well. I was familiar with some of the tools as well. It definitely played a role in the amount of work I was able to put in, because I did not have an adjustment period or learning curve.

    ii. **Documentation** - I did more. It was again the familiarity that I had with what was being done, that allowed me to act like a 'team leader' and through that role, I had more familiarity with the overall project, that was easier for me to describe the project with the detail it needed in documentation. It definitely wasn't all me. One person would handle the class diagram, one person would do the domain area diagram, one person would do the sequence diagram, and then I would put all those together in the documentation, and describe them, so most of the written documentation was definitely me but we had a pretty decent involvement in the diagrams.

    iii. **User Stories/Use cases** - Pretty even split. Two in person meetings as a team and we talked through it.

    iv. **Coding** - I did the most by a noticeable amount. It again comes down to not having a learning curve, and the familiarity that I had. I also had the advantage of this being a web project, and my undergraduate degree was in web, so I was very comfortable in this environment and was very willing to take on a larger amount of work. If there was more time or less external pressures from other classes, it definitely would have been a more even split.

    v. **Testing** - Testing was pretty evened out distribution. Because I did a lot of the coding, I made sure I wasn't testing my own code. So, I took the team leader role and made sure that everyone knew what needed to be tested and people were assigned certain areas of testing. I stepped back and let the team take a more even split on that.

    vi. **Issue Management** - Pretty even.

9. **How did you collaborate with each other?**

a. We had virtual and face to face meetings. We would see each other in class. We would try and do in person meeting with any extra time we had in class or brief in person meetings before or after class. We had virtual standups on Slack and any other Slack messaging as necessary for the project. I think that the in person meetings are generally better than the virtual meetings. It is easier to get more out of the in person meetings than the virtual meetings. But the virtual meetings are fine interim solutions. To have that virtual standup made daily standups more feasible and manageable. But face to face meetings can be easier to work with, especially in a real world scenario. But in a campus setting, the accessibility to do virtual meetings was easier.

10. **What are the collaboration issues that you have encountered?** - No. We had a really good team. Everyone was friendly. We kept it pretty light and loose. We were comfortable with each other. Personality wise, we had a good team that got along well with each other. I don't remember a ton about what my teammates' personality test results were, so I cannot speak authoritatively on how much of an impact that played, but I do think that looking at it, we had a set of personalities that got along well and complimented each other. I was comfortable stepping up and leading, and the others did not feel threatened by that and there weren't any fights about that. There were personalities that were good at thinking analytically and tracking everything and identifying anything that I missed at laying out the big pictures. So, we were pretty well-suited for each other. We made use of the class time available to us, made use of the Slack channel to maintain a pretty decent amount of communication.

11. **What type of test artifacts are there on GitHub?**

12. **How do you define quality?**

    a. Software engineering quality measures how well the software is designed and how well the actual implementation of that adheres to the design and looking specifically at software engineering, how well the process was executed in the actual creation of it.

13. **What are you doing to maintain quality?**

    a. Adherence to the Agile methodology introduces a lot of that. Using Slack and Trello to manage the creation of tasks and stories and scheduling our work based on importance, goes towards the quality of design and as we work through those, are the quality of our adherence to those designs. So I think the early stage planning that Agile introduces goes a long way in Software engineering quality. The unit testing that as well reinforces and helps catch any errors with the quality of the implementation to match that process. If done correctly, any process helps with maintaining software quality. I do think that Agile, in current setting, with higher education and students that have some familiarity with code but varying levels, I think Agile helps the most out of all the processes I am familiar with, as it gives the most flexibility to a team and the most exposure to everyone, in sort of incorporates whole team members to allow for easier sharing of knowledge and experience.

14. **How do you handle your releases?**

    a. We had a verification period, where everybody would look at what we had and made sure we were happy with it, and I was the one that specifically did the releases.

15. **Are you documenting issues on GitHub?**

    a. No

16. **Did each one of you commit/merge your code**?

    a. Yes.

17. **Did tool help or hinder process adherence?**

    a. Being specifically told of what we are going to use, both helps and hinders, but I feel that the amount of help it offers, outweighs the amount it hinders. Let me break this down. Individually, being told that we have to work in a certain way that comes with a learning curve, because everybody works in their own unique way and everybody is comfortable with the way they work, so they tend to resist changing to a different way. So, being told specifically what tools to use and how to use them steepens the learning curve a little bit. But being able to ensure that everybody on the team is working in the same way and using the same software goes a long way to eliminate a lot of the conflicts that may arise when introducing different tools and smoothens over a lot of potential areas where you could run into delays trying to reconcile the differences between different tools. So, I think that individually it increases the learning curve, but overall makes the full life cycle of the project to go more smoothly. The set of tools that were given to us for WebCheckers, in a holistic way helped. From a personal concrete example, I had never worked with Java Spark before. So it was not the solution I would have picked. Also, it wasn't particularly well documented. So the process of learning how to use Java Spark specifically added time on to my workflow at the start of the semester. But once I had gotten it down and so did the team, I knew we were all following the same process and doing things the same way. We were all working in IntelliJ, our code is being worked along in the same environment, and different IDEs can add different accompanying files, so we didn't have to worry about any of those being carried over to GitHub. We all knew how to operate in Slack and Trello so that reduced communication issues. Even down to GitHub, where we were told the appropriate way to commit and do pull requests, having everybody on the same page makes moving through the process effectively easier. Now that I look back upon it, the one thing I might have changed is, I am not nuts about the Java Spark library. So, if I had a choice in whatever language I could choose, I would have chosen Dot Net ASP, or the traditional stack - PHP, HTML and CSS.

18. **What all coding issues have you encountered?**

    a. Nothing uncommon.

19. **What programming languages did you choose for your project?**

    a. Java Spark.

20. **What were the main reasons for choosing the language you chose?**

    a. It was given to us.

21. **Does one language have any specific benefits over the other when it comes to incorporating software engineering practices?**

    a. No I don't think a language affects the software engineering process by itself. I think it is certainly possible to follow good software engineering practices on any language. A software engineering process is pretty language agnostic.

22. **Code reviews?**

    a. We did very little. This was where we sacrificed the most so as to get the process done in time. We had some bonus point incentives to add extra features, so we took the route of adding in a couple extra features, so we focused on that rather than doing full code reviews on everything. But of the code reviews we did, none of them exposed any particular technical issues.

A.1.4 Team 2

A.1.4.1 Team Member 1

1. **What Software engineering process are you using for your project?**

    a. It was Scrum Agile. I chose it because I found it quite efficient. It had two week cycles of Sprints which give you a rigid deadline and a proper way of working and collaborating with team and it provides you a basic framework of going from planning to development to testing, deploying and delivery. That's why we chose this process.

2. **What are the SE practices/activities that you are following?**

    a. I think the first thing that we did was Sprint planning, where the team gathered to break down the project into features and put them into timeline, and breaking those features into user stories and to assign those user stories and different parts of the user stories to different developers so that we can collaborate under different roles. Someone may be a developer. Someone maybe a tester, etc.

3. **What kind of testing are you performing?**

    a. We had some kind of testing module in the code itself. I am not very familiar with Java anymore, and also I cannot remember much, but I think we wrote different test modules and test cases. We did more of unit testing.

4. **Did you document your testing on GitHub?**

    a. I don't remember.

5. **What all SE quality practices are you incorporating in your project?**

    a. I don't remember.

6. **How much importance do you place on commenting your code?**

    a. It is very important. It not only helps you to go back and understand your code, but also lets other people understand and leads to readability of code. It is not necessary to comment each line of code. It is not enough to comment only one line either. Commenting should be a habit. It can be a burden, but it is not that high a burden that one doesn't do it since it is very important. I don't think our project had sufficient commenting, primarily because deadlines were tight and approaching.

7. **What all kind of comments does your project have?**

    a. I think we had some class level and function level comments.

8. **How much importance do you place on documentation in your project? What all kind of documentation does your project have?**

    a. It is very important. I think it starts with requirements understanding, so a requirement document or a user story document is very important. Feature level documentation is easier to understand than user story level documentation.

9. **How has the work been divided between team members?**

    a. I think one of the team members was a very good developer and he contributed a lot as he had good knowledge on software development and Java. I did not contribute as much as him, but I contributed the second most. The other two had minimal contribution. The learning curve and desire are two things that probably impacted the two team members who contributed the least. I was acting like a Scrum Master, facilitating between the team members. I noticed that among these two team members, one of them was eager to learn, but did not have prior knowledge of the technology. It took me one to two months to help him/her to go through the process and I was actually involved in helping him/her learn Java and GitHub, etc., but it was still very slow and his/her desire to learn could not overcome the lack of basic knowledge in such a short span of time. This is the reason the third member could not contribute as much. When it came to the fourth team member, there was a complete lack of desire to work. He/she was not that eager to take user stories on their own and they did not work much till the very end of the semester.

    b. What do you feel is the contribution of each person when it comes to

        i. **Design** - One of the team members had the most contribution in Design and coding.

        ii. **Documentation** - I did a lot of the documentation. Not much contribution from the rest.

        iii. **Requirement Gathering** - I did a lot of that.

        iv. **User Stories/Use cases** - User stories were done in even distribution.

        v. **Coding** - One team member did the most followed by me.

        vi. **Testing** - Whoever developed, they did testing. So mostly two members did it.

        vii. **Issue Management** - Here as well, it was primarily the one person who contributed most followed by me.

10. **How did you collaborate with each other?**

    a. We tried to schedule biweekly or weekly meeting. We had mostly face to face meetings. We had informal communication over WhatsApp. We did not do daily standups.

11. **What are the collaboration issues that you have encountered?**

    a. The primary contributor who did most of the work was very fast at coding, so sometimes it was hard for me to catch up with what he was doing. I tried to talk to him frequently to try and slow down the tempo and work together. So that was something I struggled with. The other one was the lack of knowledge or lack of desire to participate by two team members in the project.

    b. **How did you work on resolving them?**

i. We tried to schedule more meetings to help everyone understand the code and the development process and give an overview of what is being done.

12. **How do you define quality?**

    a. I think software ties up the tasks or something that it performs. The quality is to carry the process smoothly and then everything regarding that aspect is being handled properly and everything is being tested properly. And all this results in software quality.

13. **What did you do to maintain quality**?

    a. I don't remember.

14. **How do you handle your releases?**

    a. It was different phases of the project and defined by the requirements of the project itself. We basically released code on GitHub and created project versions.

15. **Did you document issues on GitHub?**

    a. Yes but not sure if it is on GitHub.

16. **Did each one of you commit/merge your code?**

    a. Yes. I helped one of the team members to commit from their account but.

17. **Did tool help or hinder process adherence?**

    a. I think the tool stack provided was fine. I did find using Java and Java Spark difficult to work with though because I was most comfortable with Python. Had I more knowledge on Java Spark, it may have made it easier to follow the process. All other tools were good.

18. **What all coding issues have you encountered?**

    a. Initially in the development phase, we may have gotten a few bugs and we tried resolving them ourselves. But if we had pull request issues, we tried to solve it together as a team.

19. **What programming languages did you choose for your project?**

    a. Java Spark

20. **What were the main reasons for choosing the language you chose?**

    a. It was given to us.

21. **Does one language have any specific benefits over the other when it comes to incorporating SE practices?**

    a. No. I think all languages assist in following a software engineering process.

22. **Code reviews?**

    a. Yes we did and it was pretty helpful. It gave us some understanding on how each developer was coding and if they were doing it in the right way or not.

A.1.5 Team 2

A.1.5.1 Team Member 2

1. **What Software engineering process are you using for your project?**

    a. Agile methodology under OpenUP. It was given to us.

2. **What are the SE practices/activities that you are following?**

    a. We divided work into Sprints. We updated our requirements as Sprints progressed.

3. **What kind of testing did you perform?**

    a. Yeah we performed testing at the end of the construction phase, towards the end of the project. We mostly did unit testing, some manual testing and some functionality testing.

4. **Did you document your testing on GitHub?**

    a. I don't remember documenting on GitHub, but we did document in our shared drive.

5. **How much importance do you place on commenting your code?**

    a. Highly important, especially as team gets bigger. But I feel our code should be readable and easy to understand the code commenting should be kept low. Overdoing the commenting is not recommended.

6. **What kind of comments does your project have?**

    a. We did not have much code commenting because first of all, we tried to develop this project sitting together as much as possible, because two team members of our group were not much experienced with programming, and we preferred coming together and developing it and that helped us reduce the commenting in our code. Mostly we started code commenting with topics at the top which states what the code is about, and some function definition comments.

7. **How much importance does documentation have in your project?**

    a. It is very important as it lets you see your progress and improvement in your project.

8. **What kind of documentation does your project have?**

    a. We had design document, requirement document, class diagram.

9. **How has the work been divided between team members?**

    a. It was not even distribution because the level of expertise in each group member was highly different. It did not make any sense to split the work equally. At the start, two of the team members were under the guidance of two other team members. Ideally each team member should be providing equal input, but in my personal experience, it is seldom followed. If there is a major gap in technical knowledge among team members, it becomes difficult for some team members to keep up with the other team members in

the spirit of development. Overall it is not fair, but it is practical. By the end of second Sprint, the members who were learning did try to contribute more.

10. **What do you feel is the contribution of each person when it comes to**

    a. **Design** - It was pretty evened out.

    b. **Documentation** - It was pretty evened out.

    c. **Requirement Gathering** - It was pretty evened out.

    d. **User Stories/Use cases** - It was pretty evened out.

    e. **Coding** - Primarily two contributors

    f. **Testing** - Manual testing were done by one, advanced testing/unit testing were done by others. Overall it was pretty even.

    g. **Issue Management** - Handled by members who coded but it was discussed among all.

11. **How did you collaborate with each other?**

    a. We used various tools such as Slack, WhatsApp, etc.

12. **What are the collaboration issues that you have encountered?**

    a. We had good collaboration among us.

13. **How did you work on resolving collaboration issues?**

    a. N/A

14. **How do you define quality?**

    a. How well a project is built in terms of software design and what processes are carried out for this software development and its implementation, how reliable the product is, all these would point to software quality.

15. **What are you doing to maintain that?**

    a. We did domain analysis, designs, class diagrams, level of cohesion through classes, etc. We were constantly bringing out metrics and from feedback from our professor, and code metrics, code coverage, etc.

16. **How do you handle your releases?**

    a. Before release, we tried resolving bug issues, etc. While planning the release, we tried resolving all the bugs and then do the release.

17. **Did you document issues on GitHub?**

a. No.

18. **Did each one of you commit/merge your code?**

    a. Most of the time except once I guess.

19. **Does tool help or hinder process adherence?**

    a. Helped mostly. But a different tech stack such as PHP, MySQL or any JavaScript framework may have helped more. This is because development in those are quicker than development in Java Spark and you would find more solutions for issues online for PHP, MySQL and JavaScript than Java Spark and are more newcomer friendly and are highly supported by web tech forums and users online and it gets easier to solve issues if you are working in those tech stacks. Trello and Slack were pretty on point with industry standards and were quite helpful.

20. **What all coding issues have you encountered?**

    a. Mostly code merge issues

21. **How are you working on resolving them?**

    a. We met more frequently and together tried solving.

22. **What programming languages did you choose for your project?**

    a. Java Spark

23. **What were the main reasons for choosing the language you chose?**

    a. It was given to us.

24. **Code reviews?**

    a. yes but very few, because we were always on time crunch solving issues, resolving bugs, and code development contribution wasn't even till Sprint 2.

A.1.6 Team 3

A.1.6.1 Team Member 1

1. **What Software engineering process are you using for your project?**
    a. It was an Agile process- Scrum. It was the course requirement.
2. **What are the SE practices/activities that you are following?**
    a. Sprint planning, user stories, planning poker to assign story points to user stories, Sprint retrospective at the end of every Sprint, documentation at the end of every Sprint, code reviews at the end of every Sprint.
3. **What kind of testing did you perform?**
    a. Unit testing - JUnit, and there was manual testing by default
4. **Did you document your testing on GitHub?**
    a. No
5. **How much importance do you place on commenting your code?**
    a. It is very important since it was a team project for understandability. We however did not do sufficient commenting for Web Checkers because we were so focused on getting things done, so we fell behind in maintaining a few of the quality practices.
6. **What all kind of comments does your project have?**
    a. Function based comments.
7. **How much importance do you place on documentation for your project?**
    a. It is very important. We had to document many things after every Sprint.
8. **What all kind of documentation does your project have?**
    a. There was a design document, I don't remember more.
9. **How has the work been divided between team members?**
    a. The work distribution was pretty uneven when it came to coding. The first Sprint was fine, but from the second Sprint, it got uneven, because not everyone was well equipped with Java coding, and we fell short there. I and another teammate were good at coding and it was primarily us who worked on the coding more. The course didn't teach us the functionality of what we had to do. It was really oriented towards following a particular process. It didn't teach us coding aspects or Java Spark or how to approach a functionality. So there was a learning curve and that took a while and had that learning curve been not so steep, it would have helped in a more evenly work distributed team.
10. **What do you feel is the contribution of each person when it comes to**

a. **Design** - There were two core contributors. This was because the other two were more focused on coding as they contributed to coding more.

b. **Documentation** - There were two core contributors. This was because the other two were more focused on coding as they contributed to coding more.

c. **User Stories/Use cases** - This was pretty evened out distribution.

d. **Coding** - Two primary contributors. (Not the ones who contributed to Design and Documentation).

e. **Testing** - The two primary developers did most of the testing, but there was not much collaboration.

f. Issue Management

11. How did you collaborate with each other?

a. **What are the collaboration issues that you have encountered?**

i. We lacked communication, especially towards the later Sprints. It was mainly because everyone was very busy with other courses and this course was very challenging. This project was also very challenging and the amount of time that we could dedicate was not sufficient to get the project done while following the Agile process practices. We just wanted to deliver and did not spend time on communicating. The lack of communication impacted us and team members started lacking interest in the work.

b. **How did you work on resolving them?**

i. The only way to resolve this was the people who were coding had to take extra work and let the other two not take up this pressure. We couldn't find a proper resolution to our lack of communication or uneven distribution of work.

12. **How do you define quality?**

a. Deriving the metrics to analyze the performance of your software. These metrics that you generate will be very helpful to understand where your software stands and based on that you can perform various activities like refactoring.

13. What are you doing to maintain quality?

a. We did code coverage, commenting/documenting the code.

14. **How do you handle your releases?**

a. It was through GitHub. It allowed us to release our code. We could also add information about our release on GitHub.

15. **Did you document issues on GitHub?**

a. I don't remember.

16. **Did each one of you commit/merge your code?**

    a. No. Sometimes someone did the work but because they did not know how to work on GitHub, they would send their code in a zipped folder to someone else and that person would upload the code from their own GitHub account. So when you see the GitHub authors in the repository, it is not necessary that it is indeed those authors who have done those work.

17. **Did tool help or hinder process adherence?**

    a. The tools helped but the learning curve for Java Spark, Mockito, etc, was steep. A separate tutorial explaining the tools (primarily Spark, Mockito, etc) or a mini project which would have shown what the expectations are and the targets before doing WebCheckers may have helped us out more as we would have had a better perspective. But the other tools were fine. Looking back, I may have chosen the same technology stack, but give the project more time and setup clearer expectations.

18. **What all coding issues have you encountered?**

    a. Some of the common ones were to make the functionality work.

19. **What programming languages did you choose for your project?**

    a. Java

20. **What were the main reasons for choosing the language you chose?**

    a. It was given to us

21. **Does one language have any specific benefits over the other when it comes to incorporating SE practices?**

    a. Languages that are very well established, having good documentations, usually help ease working with them, especially in a time crunch scenario, and thus help us follow the process better without having to sacrifice much on the process activities.

22. **Code reviews?**

    a. We did not do much code reviews because of time crunch. But they are very important, especially critical in the real world industries, and helps you not do much refactoring later.

Any Feedback? - Overall, now that I look back, I wished I had gotten a better team, in the sense of better matched skillset and it is important to have a team with the relevant skill set in the team. If a team is having an unbalanced skill set, it is important to communicate with the team member, give him/her some time to learn the technology, but if it still affects the project, then it is better to talk to him and maybe he/she is better off working in a different team, because in the end the product delivery

A.1.7 Team 3

A.1.7.1 Team Member 2

1. **What Software engineering process are you using for your project?**

    a. Agile. It was required by the course.

2. **What are the SE practices/activities that you are following?**

    a. Sprint planning, user stories, product backlog, Sprint retrospective, cross team validation, daily standups, continuous integration, Scum meetings, planning poker for weighing user stories.

3. **What kind of testing are you performing?**

    a. Unit testing, and user testing, end to end testing, edge case testing.

4. **Did you document your testing on GitHub?**

    a. There is a file where we did document testing, code coverage, etc. but it is not in a separate file under the name of 'Testing', but is under some file having different types of documentation.

5. **How much importance do you place on commenting your code?**

    a. I believe it is of highest priority. It's the first thing you should do once you know your functionality is implemented because that is what makes your code maintainable. It helps anyone new on the team to understand what is happening.

6. **What all kind of comments does your project have?**

    a. Since we had a starter file, some of the Java doc was already in place, and we generated a few more. We commented for function definitions. We actually commented very poorly. We did not realize the need and depth of comments at that time. Also, the Sprints were less than 2 weeks and there were only two of us coding, so we were barely making it to the Sprints, so we did not get the time to comment. If there was a more even distribution of work, we could have had better documented comments.

7. **How much importance do you place on documentation for your project?**

    a. If a project has too much documentation, I don't really read them. I go directly to the source code to try and understand what is happening. But if a document is one or two paged, which shows conciseness, I would read it. It is also important to have relevant document names, so that I can read the documents that I feel would help suffice my purpose. Documentation is very important, but you should not go overboard with it and to only document the relevant information. I also feel, in the real world, people don't care much about sequence diagrams or activity diagrams. They care about the events and the sequence, but they don't care about the sequence diagrams. So, it is important to have the relevant documentation.

8. **What all kind of documentation does your project have?**

a. We have a requirements doc, design doc, code coverage testing report.

9. **How has the work been divided between team members?** -

    a. **What do you feel is the contribution of each person when it comes to**

        i. **Design** - I did 70% of the design. One of the team members had a steep learning curve and so could not contribute a lot, and a second member was having some personal struggles and could not put in as much work as possible and communicated to us about it. The third member contributed the most to coding and hence he did not contribute much to design and documentation.

        ii. **Documentation** - It is the same as the design.

        iii. **User Stories/Use cases** - This was fairly even.

        iv. **Coding** - There were two developers. The third teammate was very new to programming and had to learn as the project progressed and hence could not contribute much. The fourth teammate wanted to handle the documentation and hence did not contribute much to the coding. I feel the third and fourth team members were not very interested in development.

        v. **Testing** - The developers did the unit testing. User testing was done by all 4 team members.

        vi. **Issue Management** - The developers worked on technical issues. So, it was just two of us.

10. **How are you collaborating with each other?**

    a. We used Trello for user story management, Slack for Team communication, we had meetings and daily standups.

11. **What are the collaboration issues that you have encountered?**

    a. Communication is a two way thing and initially I tried communicating with the whole team, but there was a lack of communication between the team members, specifically a lack of communication from the third and fourth teammates. If I can go back now, I would ensure the tasks are divided up fairly and everyone knew what they need to work on. We also had a huge collaboration issue with the fourth team member only attending the first 3 meetings out of 10 meetings we had.

12. **How did you work on resolving them?**

    a. We did try talking on Slack, and putting meeting minutes and updates on Slack, but later on there was no resolution.

13. **How do you define quality?**

a. A product that is easily maintainable, is user friendly and reliable and does what it is supposed to do.

14. **What did you do to maintain that?**

    a. Since we followed the Spark framework, which relied on the MVC architecture, so the routes and the data model was separate, so it was maintainable. Each of our data model piece was a separate class, so we were able to maintain it, even if there was a large scale user who would play. We modularized our code. Our code was not tightly coupled.

15. **How did you handle your releases?**

    a. We used GitHub to release at the end of each Sprint.

16. **Did you document issues on GitHub?**

    a. No.

17. **Did each one of you commit/merge your code?**

    a. Yes.

18. **Did tools help or hinder process adherence?**

    a. The technology stack overall helped. But I would have preferred Node JS instead of Java, because it was a web project, and it would have been easier to work with.

19. **What all coding issues have you encountered?**

    a. Nothing out of the normal logic and bug issues.

20. **What programming languages did you choose for your project?**

    a. Java Spark.

21. **What were the main reasons for choosing the language you chose?**

    a. It was given to us.

22. **Does one language have any specific benefits over the other when it comes to incorporating SE practices?**

    a. I feel Node JS would have helped in process adherence better. To adhere to a process, you need sufficient time. If you have a technology stack that is time consuming to use/learn, it does not make life easy and you spend extra time on that and not on following certain good software process practices.

23. **Code reviews?**

    a. We did it and it is very important and is helpful.

24. Any feedback?

a. I feel Daily standup does not need to happen every day. I feel it is a bit hyped and can be done in a couple of days. I also wished the Sprints were shorter and with lesser workload in each Sprint and should have started earlier in the course.

A.1.8 Team 4

A.1.8.1 Team Member 1

1. **What Software engineering process are you using for your project?**

    a. Scrum because it was part of the course requirement.

2. **What are the SE practices/activities that you are following?**

    a. We followed with initial planning, daily stand-ups, met two to three times a week as a group, we did documentation, and by the end of the Sprint, we did demo, and also had a retrospective.

3. **What kind of testing are you performing?**

    a. We did manual testing of the product. Am not sure if we did unit testing.

4. **Did you document your testing on GitHub?**

    a. No, I don't think so.

5. **How much importance do you place on commenting your code?**

    a. We had less comments in our project as I think we gave it a little less importance, especially since it was a class project. I also feel the code should be pretty self-explanatory. It should be as minimal as possible and only worth having code comments when some logic is quite complex. Maybe for APIs we need more code comments, but not for this. In some classes, the code comment importance may have been a bit hyped up, and maybe the professors just wanted all the students to grasp its importance and hence spoke about it often so that some students who did not get to understand its importance in the first or second time, may get it by the third or fourth time.

6. **What kind of comments does your project have?**

    a. I don't remember.

7. **How much importance do you place on documentation for your project?**

    a. It is more important than comments. If someone new joins the team, they will read the documents before jumping into code comments. There can be functional documents, technical documents, etc., and each has a purpose to make the project understandable. It is not important to have every single detail in the document. A high level architecture and overview is fine and once in a while it is important to update them.

8. **What kind of documentation does your project have?**

    a. We had to write user stories for each Sprint and a technical document - not sure what kind of details we had in it.

9. **How has the work been divided between team members?**

a. It was more or less an even distribution of work. Initially when we started the Sprint, we decided who will work on what, but it eventually ended up in mostly one person doing the job. I think there was too much to learn and different people were working at different paces which impacted us. The course was software engineering and I feel that this was a very difficult project for Foundations of Software Engineering. The learning outcome should not have been to learn Java Spark, but to learn the software process, which I did ultimately learn, but this was not an appropriate project for this course. A tutorial explaining the technology may have helped as well.

10. What do you feel is the contribution of each person when it comes to

a. **Design** - Mostly one person working on that

b. **Documentation** - Mostly one person working on that

c. **User Stories/Use cases** - Fairly even

d. **Coding** - It was primarily one team member

e. **Testing** - Fairly even

f. **Issue Management** - Primarily the developer worked on this

11. **How did you collaborate with each other?**

a. We had good communication between team members. We had WhatsApp groups, we met every week, we did not have any personality clashes amongst team members, and we never had any problems.

12. **What are the collaboration issues that you have encountered?**

a. We did not have any issues

13. **How do you define quality?**

a. It is important to follow a process and the appropriate one and then prioritize the tasks that are there, and by the end of the Sprint, it is important to have a retrospective, and other than that good coding practices such as code reviews, etc.

14. **What are you doing to maintain quality?**

a. We worked together and shared knowledge which helped because you can learn from each other and use each other's logic.

15. Did you document issues on GitHub?

a. No

16. Did each one of you commit/merge your code?

a. Yes

17. **Did tools help or hinder process adherence?**

    a. We mostly used WhatsApp even though Slack was given to us. We communicated on Slack for the sake of grading because WhatsApp is easier.

18. **What programming languages did you choose for your project?**

    a. Java Spark

19. **What were the main reasons for choosing the language you chose?**

    a. It was given to us

20. **Code reviews?**

    a. A few.

A.1.9 Team 4

A.1.9.1 Team Member 2

1. **What Software engineering process did you use for your project?**
    a. Agile-Scrum process. It was in our curriculum.
2. **What are the SE practices/activities that you followed?**
    a. We participated in Sprints, and after every Sprint we used to have small retrospectives about what went wrong, what went good etc. , we had presentations after a few sprints.
3. **What kind of testing did you perform?**
    a. Yes, we had some kind of testing, unit testing.
4. **Did you document your testing on GitHub?**
    a. I don't remember.
5. **How would you define quality?**
    a. The ease with which anyone can pick up my code and work with it, read it, understand it, and develop my code would be quality.
6. **What did you do to maintain quality?**
    a. The code was modular, it had followed the MVC structure, so that was explanatory. We had separation of concerns. Do not remember much more.
7. **How much importance do you place on commenting your code?**
    a. For me I would like to comment on the actual logic. Code comments are important for someone new to understand what we have coded.
8. **What kind of comments does your project have?**
    a. Complex logic, etc. Some basic comments about MVC, etc. We could not do much because of time crunch.
9. **How much importance does documentation have in your project?**
    a. It is important, but what is more important is the way a project is documented. I do not want to read two to three pages of documentation. I would want to have step by step

concise documentation describing the project. Maybe a video, etc. While working in the real world, you are usually not exposed to sequence diagrams or a lot of other diagrams. At least as an engineer when we enter the industry, it is important to have the knowledge about all sorts of documentation and design. The user story diagrams are more abstract than the other design diagrams, but not really very important.

10. **What kind of documentation does your project have?**
    a. We had user stories, activity diagrams, sequence diagrams, class diagrams, help document for running code on GitHub, etc.
11. **How has the work been divided between team members? What do you feel is the contribution of each person when it comes to -** Everyone had their own strengths and weaknesses and overall everyone contributed fairly equally.
    i. **Design -** Everyone pitched in but one of us contributed a bit more, though we all discussed all the design decisions.
    ii. **Documentation -** Because there were two primary code contributors, it was them who did the documentation as well since as and when they documented, they coded as well, since documentation helped in coding as well. But for other kinds of documentation, there was a fairly even contribution.
    iii. **User Stories/Use cases -** Because we all sat together while working, we would work together. We helped each other out.
    iv. **Coding -** Because there was a huge coding part, and one of the teammates was very good at it and it was primarily him/her who worked on it followed closely by a second code contributor.
    v. **Testing -** It was primarily the two developers who worked on the testing as well.

vi. **Issue Management -** We solved them together since we mostly had face to face meetings.

12. **How did you collaborate with each other?**
    a. We had face to face meetings.
13. **What are the collaboration issues that you have encountered?**
    a. No.
14. **How did you work on resolving collaboration issues?**
    a. N/A
15. **How did you handle your releases?**
    a. Do not remember
16. **Did you document your issues on GitHub?**
    a. No
17. **Did each one of you commit/merge your code?**
    a. Yes, I think I did.
18. **Did the tools help or hinder process adherence?**
    a. If the learning curve was less, we could have contributed more evenly in the team and we could have accomplished more. There was not a lot of documentation on Spark which also made it difficult.
19. **What general coding issues have you encountered? How did you working on resolving them? -** Common ones
20. **What programming languages did you choose for your project? What were the main reasons for choosing the language you chose? -** Java Spark was given to us.
21. **Code reviews? -** We did it once. The amount of time was very less and we could not do much.

A.2 Word clouds generated from supporting artifacts of each team

Figure A.1: Word cloud generated from Project Design Document for Team 1

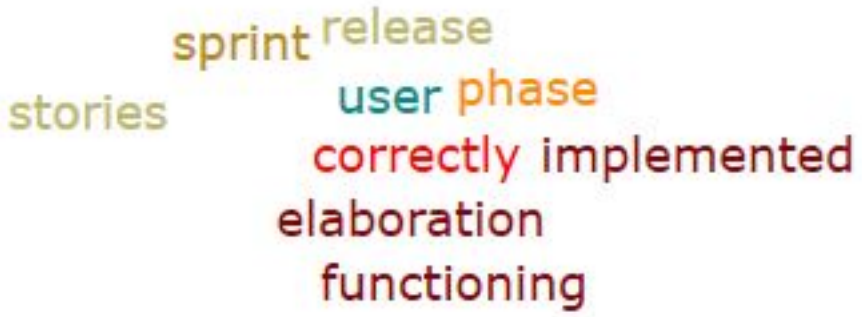

Figure A.2: Word cloud generated from artifact documenting Sprint 1 for Team 1

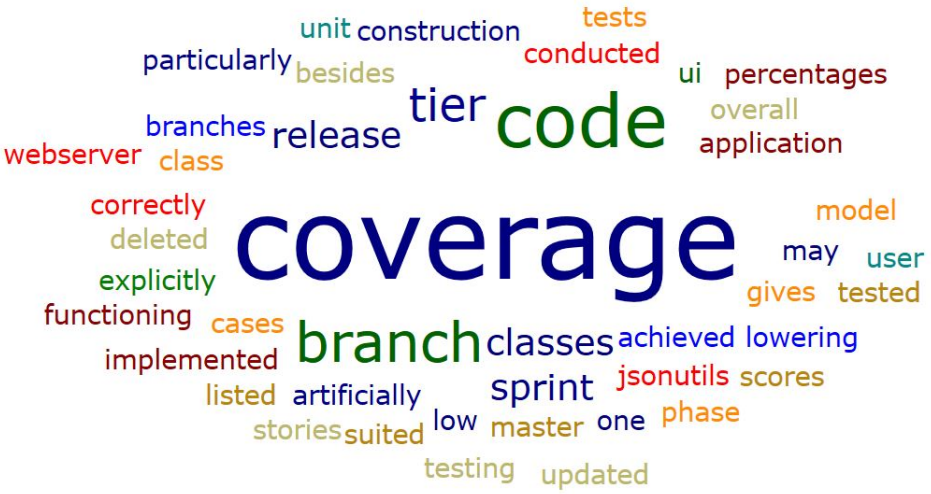

Figure A.3: Word cloud generated artifact documenting Sprint 2 for Team 1

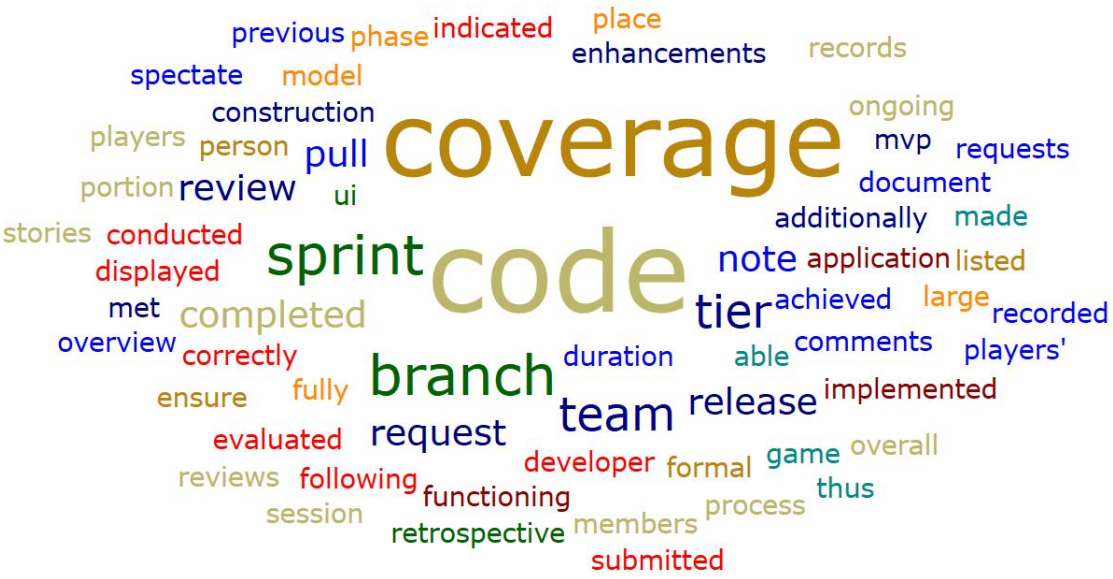

Figure A.4: Word cloud generated from artifact documenting Sprint 3 for Team 1

Figure A.5: Word cloud generated from artifact documenting Project Requirements for Team 2

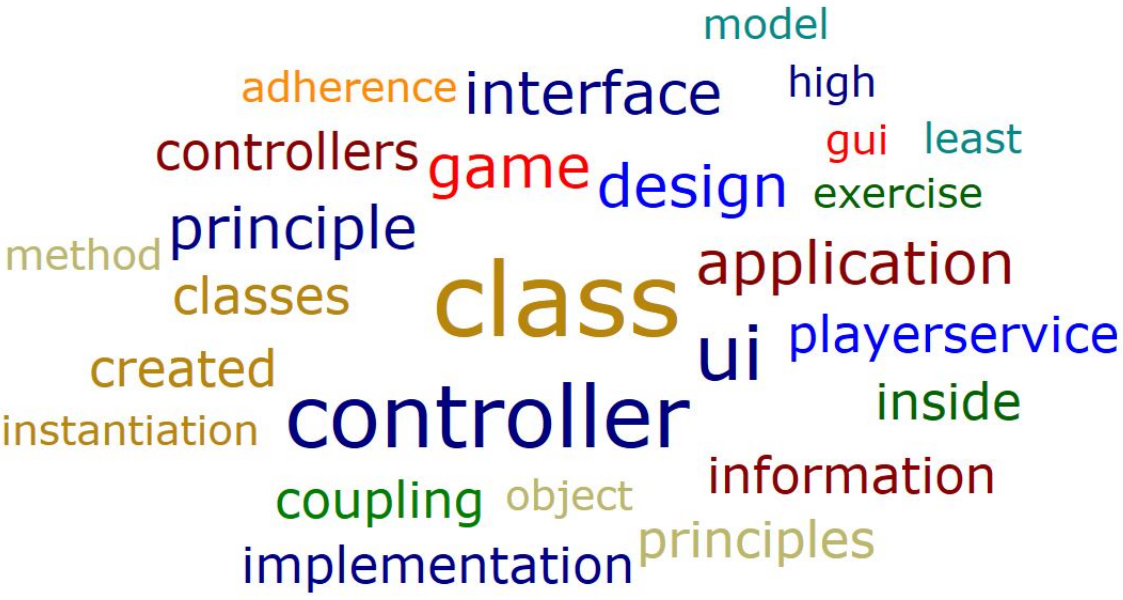

Figure A.6: Word cloud generated from artifact documenting Object Oriented Design for Team 2

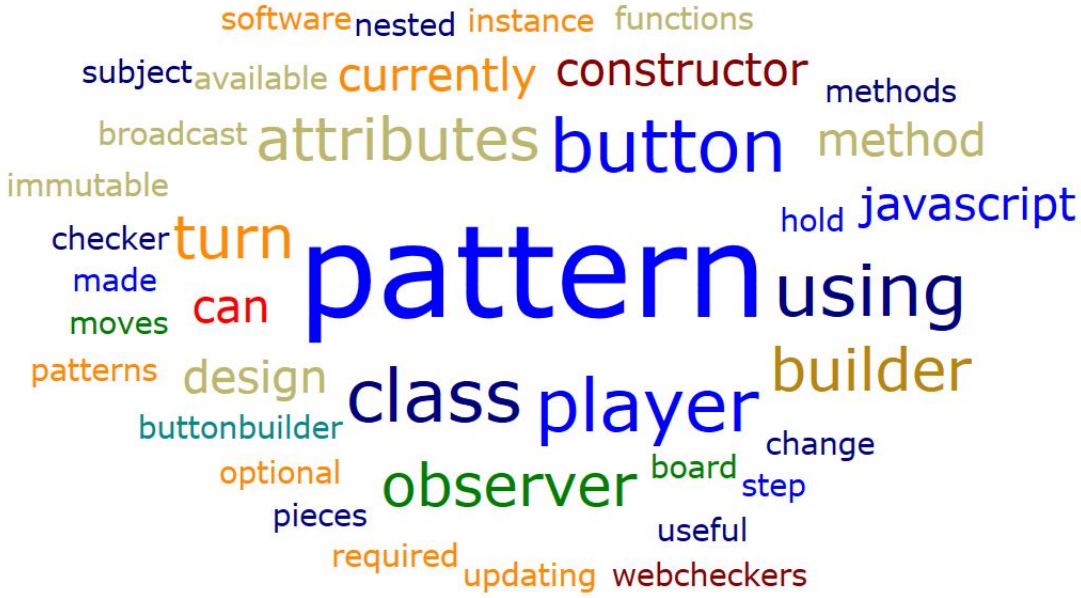

Figure A.7: Word cloud generated from artifact documenting Patterns and Refactoring for Team 2

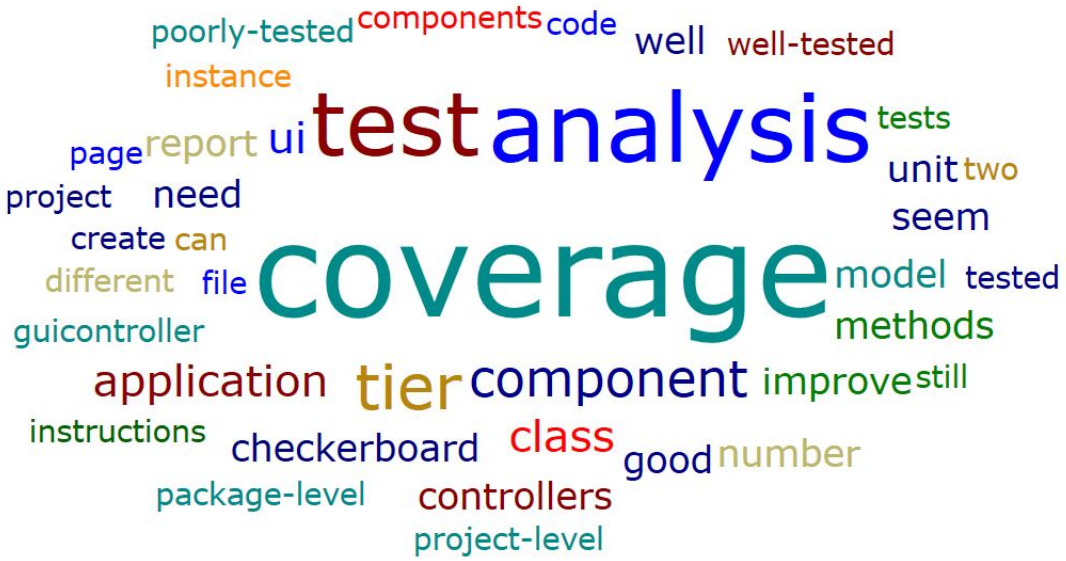

Figure A.8: Word cloud generated from artifact documenting Code Coverage for Team 2

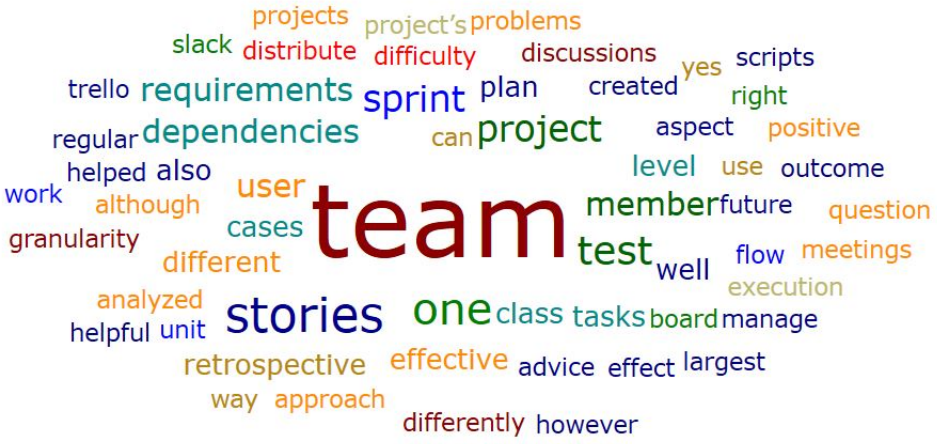

Figure A.9: Word cloud generated from artifact documenting Team Retrospection for Team 2

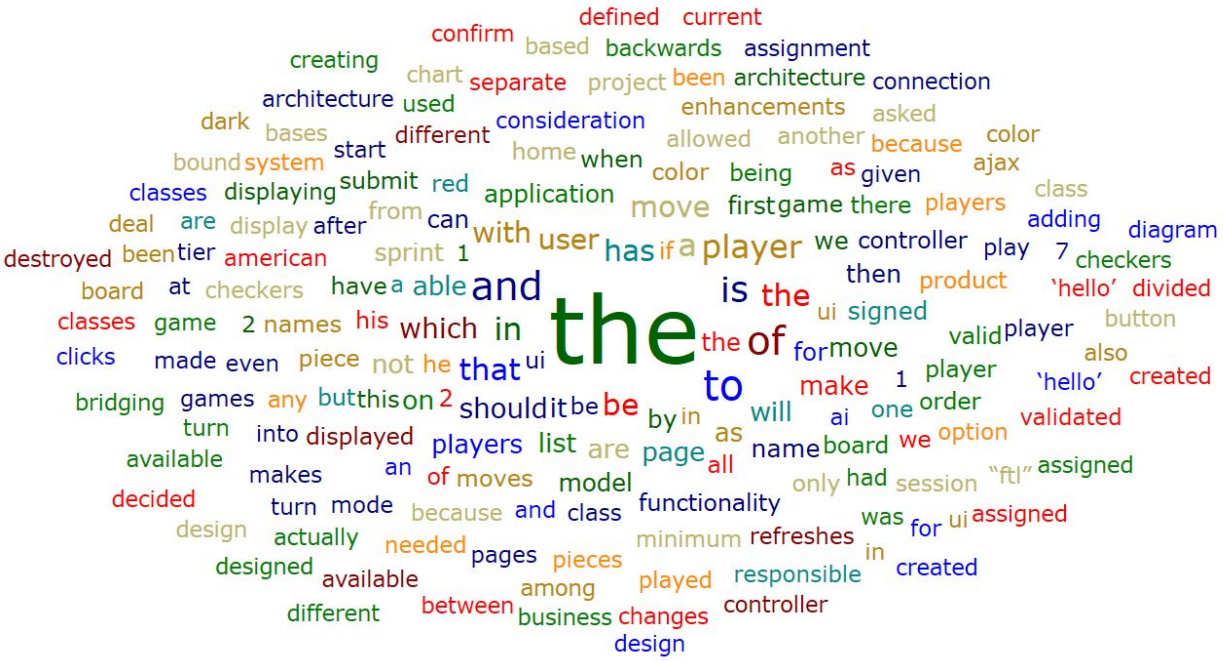

Figure A.10: Word cloud generated from artifact documenting Design Document for Team 3

Figure A.11: Word cloud generated from artifact documenting Team Retrospection for Team 3

Figure A.12: Word cloud generated from artifact documenting Design Document for Team 4

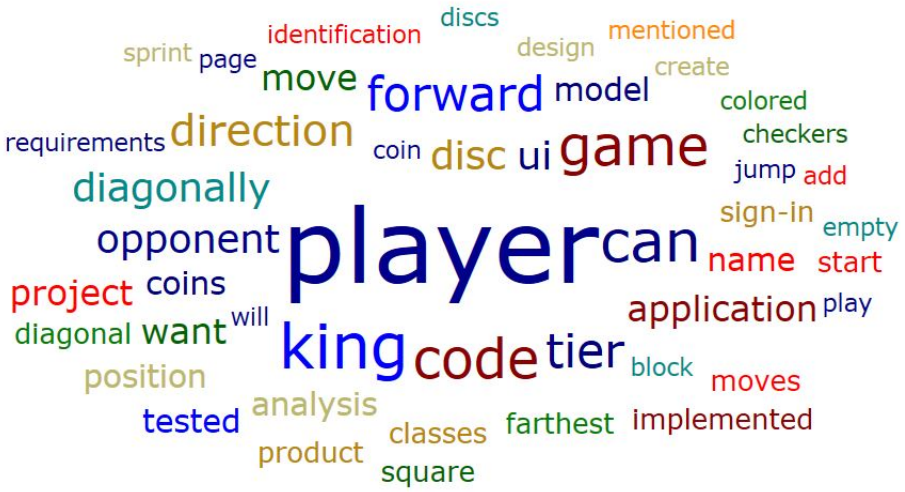

Figure A.13: Word cloud generated from artifact documenting Design Document for Sprint 2 for Team 4

Figure A.14: Word cloud generated from artifact documenting Design Document for Sprint 3 for Team 4

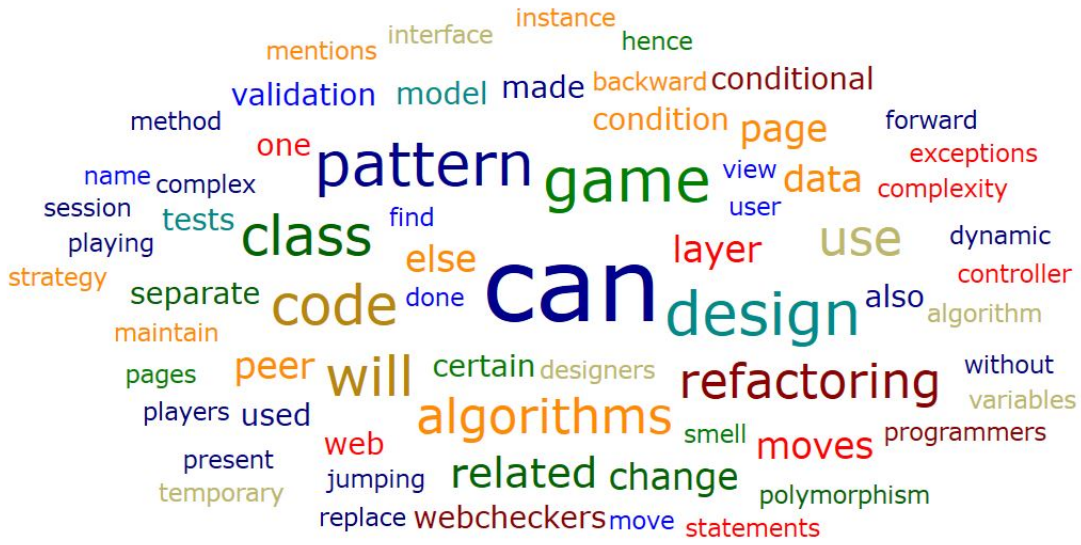

Figure A.15: Word cloud generated from artifact documenting Design Refactoring and patterns for Team 4

Figure A.16: Word cloud generated from artifact documenting Team Retrospection for Team 4

Bibliography

[1] https://developer.github.com/v3/. In *GitHub API*.

[2] https://developer.github.com/v3/orgs/members/. In *GitHub API for organization members*.

[3] https://developer.github.com/v3/orgs/outside$_c ollaborators/. In GitHub API for outside colla

[4] https://developer.github.com/v3/pulls/. In *GitHub API for pull requests*.

[5] https://developer.github.com/v3/search/search-topics. In *GitHub API for searching for topics*.

[6] https://github.com/sayantika23/process-miner. In *Process Miner*.

[7] https://help.github.com/en/github/getting-started-with-github/github-glossary. In *GitHub Glossary*.

[8] https://slack.com/help/articles/212675257-join-a-slack-workspace. In *Slack Workspace*.

[9] https://trello.com/en-us. In *Trello*.

[10] https://www.rit.edu/study/software-engineering-mscurriculum. In *Course Description of Foundations of Software Engineering (SWEN-610)*.

[11] https://www.smartsheet.com/project-design-any-industry. In *Project Design Document*.

[12] https://www.thesprucecrafts.com/play-checkers-using-standard-rules-409287. In *American Rules for Web Checkers*.

[13] http://www.se.rit.edu/ swen-610/projects/project.html. In *Web Checkers*.

[14] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting &Amp; Designing for Pervasive Information*, SIGDOC '05, pages 68–75, New York, NY, USA, 2005. ACM.

[15] Tore Dybå, Rafael Prikladnicki, Kari Rönkkö, Carolyn Seaman, and Jonathan Sillito. Qualitative research in software engineering. *Empirical Softw. Engg.*, 16(4):425–429, August 2011.

[16] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus. Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, 27(1):1–12, Jan 2001.

[17] Georgios Gousios and Diomidis Spinellis. Ghtorrent: Github's data from a firehose. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 12–21, Piscataway, NJ, USA, 2012. IEEE Press.

[18] M. Gupta, A. Sureka, S. Padmanabhuni, and A. M. Asadullah. Identifying software process management challenges: Survey of practitioners in a large global it company. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 346–356, May 2015.

[19] J. Joe, T. Emmatty, Y. Ballal, and S. Kulkarni. Process mining for project management. In *2016 International Conference on Data Mining and Advanced Computing (SAPIENCE)*, pages 41–46, March 2016.

[20] Per Kroll and Bruce MacIsaac. *Agility and Discipline Made Easy: Practices from OpenUP and RUP (Addison-Wesley Object Technology (Paperback))*. Addison-Wesley Professional, 2006.

[21] Phillip A. Laplante. *What Every Engineer Should Know About Software Engineering (What Every Engineer Should Know)*. CRC Press, Inc., Boca Raton, FL, USA, 2007.

[22] Ivan Mistrík, John Grundy, Andre van der Hoek, and Jim Whitehead. *Collaborative Software Engineering*. 01 2010.

[23] Audris Mockus, Roy T. Fielding, and James Herbsleb. A case study of open source software development: The apache server. In *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, pages 263–272, New York, NY, USA, 2000. ACM.

[24] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. Curating github for engineered software projects. *Empirical Softw. Engg.*,

22(6):3219–3253, December 2017.

[25] G. C. Murphy, D. Notkin, and K. J. Sullivan. Software reflexion models: bridging the gap between design and implementation. *IEEE Transactions on Software Engineering*, 27(4):364–380, April 2001.

[26] R. Saylam and O. K. Sahingoz. Process mining in business process management: Concepts and challenges. In *2013 International Conference on Electronics, Computer and Computation (ICECCO)*, pages 131–134, Nov 2013.

[27] Robert Schuppenies and Sebastian Steinhauer. Software process engineering metamodel (spem). 01 2002.

[28] Jeff Sutherland. *Jeff Sutherland's Scrum Handbook*. 01 2010.

[29] Mark D. Syer, Meiyappan Nagappan, Ahmed E. Hassan, and Bram Adams. Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open-source android apps. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13, pages 283–297, Riverton, NJ, USA, 2013. IBM Corp.

[30] W. M. P. van der Aalst and A. J. M. M. Weijters. Process mining: A research agenda. *Comput. Ind.*, 53(3):231–244, April 2004.

Vita

Sayantika Bhattacharya was born in Kolkata, India on March 23rd, 1990, to Niloy and Tulika Bhattacharya. She received the Bachelor of Engineering degree in Computer Science from East Point College of Engineering and Technology, Kolkata, India in 2012. She is currently pursuing her Master of Science degree in Software Engineering from Rochester Institute of Technology, United States of America. Her research interest includes Software Process Engineering, Software Process Improvement Frameworks, Process Mining and Software Engineering Requirements. Her current research includes understanding gaps between theoretical software engineering process knowledge and its real world application.

Permanent address: Crittenden Way
Rochester, New York 14623

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.