Building an intuitive and friendly data collection framework to analyze human factors involved in software development

by

Anirudh Mandalapu

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science

(Computer Science)

At

The University of Wisconsin–Whitewater

April 30, 2021

Graduate Studies

The members of the Committee approve the thesis of
Anirudh Mandalapu presented on April 30, 2021

Dr. Zachary Oster, Chair

Dr. Naomi Aguiar

Dr. Hien Nguyen

Building an intuitive and friendly data collection framework to analyze human factors involved in software development

By

Anirudh Mandalapu

The University of Wisconsin-Whitewater, 2021
Under the Supervision of Dr. Zachary Oster

Despite the most important aspect of software products being the people who design them, the research literature concerning the people in question is somewhat sparse and limited. The research that currently exists, while promising, only studies for small, controlled sample sizes. These studies are also condensed and focused down, often focusing on a very specific factor, at times in great detail. In this research, we analyze past literature concerning some of the interpersonal factors involved in software engineering. We specifically delve into the impact of team dynamics, personality, communication and skill level on the quality of the final product, often measured as a function of code quality, using a non-invasive testing framework. In contrast to other researchers who have used the popular, albeit controversial MBTI framework as a way to assess the impact of developer personality on the output of the software quality, we propose using data about the moment to moment emotional responses that occur during different stages of software development as a means of capturing this information. These data are captured from a survey that has a short enough response time and little or no technical jargon. We present and discuss our results to date from the survey, along with challenges that we faced and advice to future researchers in this area.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

DISCARD THIS PAGE

LIST OF TABLES

Table Page

DISCARD THIS PAGE

LIST OF FIGURES

Figure Page

Chapter 1

Introduction

The past two decades have seen the software industry become ubiquitous, to the extent that the average person is now surrounded by software in every waking moment of their lives. Organizations have come to emphasize their technology now more than ever, and the ones that embrace their technology come out ahead, and customers tend to embrace these companies better. With how prevalent software is, the complexity of an average software product has similarly grown. The civil engineering inspired waterfall ideology of starting at inception and executing each stage of development isn't the silver bullet it once used to be, and several new development techniques like Agile, Rapid Application Development, Extreme Programming, to name a few have started cropping up. Organizations are often faced with the dilemma of picking the best technique for their product, company culture, team size and experience level, and this is something my paper will discuss. This rapid growth and complexity highlights how important research in this somewhat under explored field is, and this study will attempt to enable researchers to collect larger volumes of data and receive a higher response rate of responses from participants.

As software products get bigger in scale, both in terms of complexity and user reach, evaluating the quality of the product and the people working on it is extremely essential. This assessment involves both qualitative and quantitative data items, as there are a lot of things about the people working on software projects that simply can't be assigned to a number because they're simply not tangible. Assessing how many bugs the team often runs into, or how

many of their initial feature checklists they've delivered is fairly straightforward and most organizations can track them fairly easily using tools like JIRA to look at unresolved tickets and GitHub to see a checklist of issues.

Despite the meteoric rise of the software industry, there's not a lot of research out there that tries to establish relationships across product quality, team emotional states, experience levels, type of development methodology and type, team communication levels and demographics. Another surprising revelation is how many authors focus on agile development techniques, in favor of several other, maybe not as prevalent development methodologies [18, 20, 28]]. This study aims to use that as a guide and attempt to extrapolate for other development types.

Another interesting facet is how little authors have chosen to describe their surveys, [1, 30, 9]. The studies follow the trend of stating a problem, delving into literature review, speaking about the data and subsequent results and conclusions. These are a few examples, but these point to a much larger trend of authors skipping over parts of the survey design, and instead delve into the mathematical side of things, or just prefer to talk about the data outright. We have attempted to improve upon here by discussing my survey design in depth, talk about biases and problems that might occur backed up by literature from the National Institute of Health, who talk about some of the biases they experienced when collecting clinical data [22].

Staying true to our previous statements about software growing in scope, this statement brings with it the obvious conclusion of software rarely being a one person effort, as most companies have large teams of developers just working on one feature. A corporation like Google has a teams of engineers solely dedicated to working on features that we take for granted as end users, one good example being the comments feature on Google Docs. As ludicrous as that sounds, we need to take a moment to look at some of the challenges involved with a feature as simple as comments, and another to consider how many failure states can happen:

1. The feature is user facing, and user facing products will always produce bugs far beyond the developer's expectations and testing capabilities

2. Multiple people can work on the same document at once

3. There's always millions of users using Google Docs all across the globe

4. Everything is saved immediately on Google Docs, with very little margin for data loss

While comments sounds like a simple feature, the truth of software simplicity is that there is a huge time and money investment on the developer's side. And with this, comes the complexity of keeping this team of extremely qualified engineers running like smooth cogs in a well oiled machine. Anything less than a perfect team dynamic would be disastrous, financially and otherwise.

Now that the reason and importance of teams has been established, this study will address an issue with existing literature, which is that most of it is based on small, controlled sample sizes. One barrier involved here is that data collection, especially when involving human subjects is difficult, tedious and expensive. We attempt to provide a solution in the form of a survey that takes into account every factor that existing literature has tried to study, and capture those within a survey, while balancing the survey for factors like response time and question complexity in an attempt to enable participants to better provide data.

1.1 Research Questions and Goals

As stated in the introduction, data collection for studies like this can prove to be slow, expensive, time consuming, and sometimes, impossible. As we observe in a deeper dive of our literature review in future sections, a lot of code quality assessments need authors to install testing frameworks on machines. While this is a trivial ask for a student body or a small organization, gathering such data from a larger organization is a huge challenge, considering the security implications of introducing a new, foreign piece of software into their ecosystem, and the amount of red tape involved with such an effort.

Our original research question was an attempt to find the best software techniques for different types of teams. The goal was to evaluate team member emotional states, communication

frequency, team interpersonal dynamics, code quality and experience level to find the best development technique for specific teams. Our metric to judge what was "best" was to evaluate code quality, and this is where our challenge from the first paragraph comes in.

The reality of collecting such a vast volume of data, coupled with literature review that demonstrated studies into each individual factor listed above on code quality, raises the following questions.

1. Can we truly correlate all or a subset of these variables to find a perfect or a close to perfect software engineering technique for a specific team? A potential approach would be to highlight variables that seem like they have the strongest correlations to the quality of the final product, and present a set of variables that can reliably predict the best development technique for such a team.

2. Is it possible to collect such data in an easy to answer survey style questionnaire instead of having to assemble teams with testing frameworks installed on their machines? If possible, how well would the results compare to historical findings?

Chapter 2

Literature Review

The aim of this paper is to analyze correlations between various factors involved in software development, some technical, but mostly human. For simplicity, we can break down these factors into "left and right" halves of an equation. The "left half" of this equation are the software quality variables, namely the number of bugs, frequency of bugs, and the initial feature wish list as compared to the features on the delivered product. The "right half" will cover everything else in the survey, including emotional states, development technique, type of work, communication frequency and team size and experience levels. This paper will now discuss some historical findings and literature derived from similar works in the field.

2.1 Teamwork

One of the factors we're testing for is teamwork, and there's a foundation in the paper "Group Developmental Psychology and Software Development Performance" by Lucas Gren and Khaled Al-Sabbagh [15]. Their development method of focus is primarily Scrum, as evidenced by their quality metric of choice, Schedule Performance Indicator.

One of the more interesting facets of this paper is the author's explanation for Group Developmental stages and the comparison to how humans grow throughout their lifetime. Their group development stages, cited as is from the paper, are as follows.

1. Stage One: The first stage is a period of Dependency and Inclusion, where members tend to show significant dependency on the leader in resolving new issues. At this stage, members spend a significant amount of energy to achieve a feeling of safety and inclusion

in their group. As a result, members become leader-focused, in a sense that he/she will provide protection for the members. Members are indulged in an exploratory phase for the sake of identifying their roles, rules, and the structure within the group. Their exploration is characterized by being tentative and overly polite since they fear being rejected.

2. Stage Two: The second stage is referred to as Counter-dependency and Fight. At this stage, members feel freer to express conflict between each other or among members and leaders since some needs for safety have been achieved in the previous stage. The group tries to free itself from being leader focused, and tends to fight about the group's goals. The authors of [15] state that conflict is an important part for the development of cohesion, as it provides the opportunity for setting the psychological boundaries, which facilitate the establishment of goals, shared values, and structure. The occurrence of conflict is a result of the members' attempt to reach a unified direction out of the many divergent viewpoints. The rise of coalitions between members who share similar values and ideas is very much apparent.

3. Stage Three: After navigating the inevitable stage of conflicts, communication becomes more open and members' trust and cooperation increase. Feedback and information sharing increase rather than being kept as a way to gain power. The aforementioned characteristics consolidate a more solid and positive relationship between members, which allow the group to carry out more mature negotiations about their goals and procedures. The group is at a stage where it is designing and preparing itself to start working effectively. Although work occurs in all the stages of group development, the group's focus on structure and goals at this stage significantly increases the group's capacity to work more productively.

4. Stage Four: As soon as the goals and structure of the group are set from the previous stage, the group's focus is diverged into getting the work done well at the same time as

the group cohesion is maintained, and remains cohesive while engaging in task-related conflict.

It is worthwhile to consider the amount of time this would take for each different type of group. For students it is possible that they might have to navigate these stages very quickly in order to accomplish the level of communication needed to accomplish their goals. An exception to this is if students select their own teammates, in which case they will lean towards people they know well, effectively bypassing stage 1 and 2.

For enterprise teams, this could potentially correlate to team size, team experience level and the number of projects they've worked together. For a smaller team of experienced developers who have worked on a lot of projects, it is easy to assume they're well into stage 4, as most teams have a well oiled routine of how to tackle projects. While not as easy for an experienced developer joining a new team, their experience and knowledge should help them quickly assimilate into a new team. This model and the associated pitfalls will most likely be seen with a team of new grads, where it is a reasonable hypothesis that quality might suffer on account of the teams not fully knowing each other's communication and working styles just yet.

These hypotheses are confirmed by the authors' results graph in [15] plotting planning effectiveness as the dependent variable and the fourth group development scale as the independent variable. The authors arrive at the conclusion that when group development increases, the planning effectiveness also increases and vice versa.

2.2 Personality Types

The next piece of our puzzle is understanding some of the how personality types influence Software Development. Speaking from personal experience, Software Development can be emotionally taxing, but also extremely rewarding when things fall in place, and different personality types handle emotions differently. It's imperative as developers that we have a better understanding of how our emotions influence the overall quality of our work, and this study by

Anderson S. Barroso, Kleber H. de J.Prado, Michel S. Soares and Rogerio P. C. do Nascimento uses the MBTI Model to come to some conclusions, most notably about the work done by students [4].

The MBTI model has been long established as the standard identification tool for defining an individual's characteristics and strengths. The model is split into pairs of preferences, known as dichotomies, quickly explained.

- Dichotomy attitude (E-I): Introversion and extroversion make up this dichotomy. Subjects identifying with the Introversion trait tend to be involved with ideas, think before acting and need time to think and recover energy. Extroverts are a stark contrast, they tend to act first, enjoying themselves in the process and are very sociable.

- Dichotomy Functions (S-N and T-F): The first half of this dichotomy is comprised of Sensing(S) and Intuition.(F) Subjects falling into the Sensing group rely heavily on facts and details, basing their decisions on concrete information. The subjects falling under the Intuition bracket work well with abstract information, preferring to make their decisions based on prior data obtained from prior knowledge. The second half of this dichotomy is split into Thinking and Feeling. Subjects falling into the former bracket prefer to make decisions based on logic and rational arguments, the latter use their feelings as a basis.

- Dichotomy Lifestyle (J-P): This last dichotomy is home to the Judging and Perceiving brackets of people. The first group feels calm when decisions are taken, and the second feel calm staying open to a final decision to take more information.

These dichotomies can be arranged into four letter pairs to form the MBTI personality type.

Using this as the personality part of the test, the authors then apply CK (Chidamber and Kemerer Metrics), to assess software quality. CK Metrics are specific to code quality, defining metrics like how many dependencies exist, how tightly coupled classes are, how many methods can be executed in response to one object of a class and the complexity of a class.

We next turn to another study by Vreda Pieterse, Mpho Leeu and Marko van Eeklen, analyzing the impact of personality on team performance in student software engineering teams [25].

This study, similar to [4], uses MBTI to measure student personalities. There are no dichotomies dividing participants up, with dimensions being used. These are: Social Interaction, Information gathering, Decision making, and dealing with the external world. In this study, data were gathered from peers within the team, where they were asked to rate each other, potentially introducing a form of Self-Serving and In-Group bias [22].

The second set of data is Participatory Levels, being split up into Social Loafer, Complacent Worker, Insightful Shaper and a Diligent Isolate, each level going from low to highest responsibility, with the highest tier, Diligent Isolate, taking over parts of the project to overcompensate. These metrics also have potential for Self-Serving and In-Group bias.

The authors hypothesize that:

1. Diversity of MBTI personality dimensions among the members of the members in a short-lived team is positively correlated with team synergy.

2. Diversity of MBTI personality dimensions among the members of the members in a short-lived team is positively correlated with the quality of the deliverables of the team.

The author's data, taken from a total of 123 teams, did not ultimately conclude that personality diversity is correlated with team synergy and only found a very weak correlation between personality diversity and the quality of the product delivered by the team.

Kortum, Klünder, and Scheider [17], similar to Barroso et al. [4], also use the MBTI model to assess student personality. They've gathered data from students, who were asked to rate their peers across four dimensions, namely Social Interaction, Information Gathering, Decision Making and Dealing with external factors.

Participants of this study were subjected to a series of micro-projects, with teams being randomized at each new project. This accurately reflects the average student programming

experience, as students in real life constantly shift classes and tend not to work with the same group for very long.

Another important factor is the level of communication and feedback that exists among team members, and this is of extreme importance in an agile environment, which is essentially built on communication [2].

Kortum et al. state that in an agile environment, it is essential to identify weaknesses, and also find the root cause to solve said issues. To address this, the authors introduce an approach that proactively presents feedback about the interpersonal behaviors of a team during sprints. The authors echo similar sentiments as several other authors studying similar fields, in that human factors are often difficult to analyze.

This test manifests itself in the form of a JIRA plugin that lets users submit self-assessments. This plugin was distributed to 130 undergraduate students, working on 15 software projects. A control group was established using half of this group, who didn't use the JIRA plugin to evaluate themselves. The authors used this data to investigate behavior-driven dynamics and the effects of fast feedback on team performances.

2.3 Developer Skill

Another huge factor in understanding and analyzing output quality is the skill of the developers involved. Common reason suggests that a better skilled developer will put out a better piece of software. In [5], Justin M. Beaver and Guy A. Schiavone test this theory in a scientific manner. Their motivation echoes the reason why our study is being written, which is the fact that there's surprisingly sparse literature delving into the human factors involved in software development, most notably assessing the developer skill as a factor.

Beaver and Schiavone in [5] use NASA's Competency Management System as the approach to measuring individual skills. This CMS is a measure of framework that's applied throughout the organization to intelligently manage and allocate workforce where needs exist, and also to assign staff to where their competencies can be best utilized. This model measures developer

skills across Implementation, Design and Requirements. This measure holds up and can accurately identify the skills of a development team, and can paint a well rounded picture of the team's skills. For instance, a team scoring high in implementation and design would design an amazing product, but might end up having to revisit chunks of their work on account of their less than ideal requirements analysis skills [5].

The next metric would be the quality of the software, captured using the ISO/IEC 9126 standard as a guide for quality expectations within a delivered product. These metrics are as follows:

1. Design Adequacy (DA)
    - How adequate are the checked design modules?
    - The ratio of the number of design modules evaluated to function adequately to the number of design modules.
2. Implementation Adequacy (IA)
    - How adequate are the checked source code units?
    - The ratio of the number of source code units evaluated to function adequately to the number of source code units.
3. Functional Implementation Completeness (FICMP)
    - How complete is the functional implementation?
    - One minus the ratio of the number of functions detected as missing during evaluation to the number of functions described in the requirements.
4. Functional Implementation Coverage (FICOV)
    - How correct is the functional implementation?

Phase Skill	ISO/IEC 9126 Software Product Quality Metric				
	DA	IA	FICMP	FICOV	FSV
Requirements Skill	Incr.	Incr.	Incr.	Incr.	Decr.
Design Skill	Incr.	Incr.	Incr.	Incr.	Decr.
Implement Skill	N/A	Incr.	Incr.	Incr.	Decr.

Figure 2.1 Results found by Beaver and Schiavone in [5]

- One minus the ratio of the number of functions detected as missing or incorrectly implemented during evaluation to the number of functions described in the requirements.

5. Functional Specification Volatility(FSV)

   - How volatile is the functional specification after baseline?
   - The ratio of the number of requirements changed after baseline to the number of requirements.

For their approach, Beaver and Schiavone came up with two pieces, a correlation coefficient equation and a third table hypothesizing the relationship between the skill buckets and the ISO/IEC metrics. Their findings, which are shown in Figure 2.1, have been cleanly broken into these pieces:

1. Effects of Skill on Adequacy: This was in line with the researchers' expectations. The increased presence of more highly skilled software developers on the development team was positively correlated with the quality measured in the final software product, especially with the various skill levels as seen against the measures of Design and Implementation Adequacy.

2. Effects of Skill on Specification Volatility: As skill levels grew, there was reduced volatility associated with functional specifications. But the authors do have an interesting finding here: skill doesn't actually affect the quality of the requirements gathering process as much as one might expect.

3. Effects of Requirements Skill: Requirements skill on software quality's impact was very strongly seen in Design and Implementation, which suffered a lot at lower skill levels.

4. Adverse Impact of Skill Level 2: The authors observe that this is where the divergence of Skill Level 2 happens.

5. Effect of Design Skill: This section's results align with the intuitive assumption that those with a higher level of design skill and experience will produce a more adequate design, and therefore a more adequate implementation. Similarly, those with less design skill and experience will produce a less adequate design and implementation.

6. Effects of Implementation Skill: These results also intuitively make sense, as the increase in low skill developers was negatively correlated with the Design and Implementation phases of the project.

The authors point to some issues with their results, in that all of their data has been gathered from small, object-oriented software efforts. So while there is definitive evidence that increased developer skill correlates positively to overall developer quality, future work would need to assess this on a much larger scale.

A similar study is based on the Capability Maturity Model (CMM) [23], which has become a popular methodology for improving software development processes with the goal of developing high-quality software within budget and planned cycle time as stated in Manish Agarwal and Kaushal Chari's study in their efforts to analyze the importance of development effort and cycle time on the resulting software quality [1]. One reason this study stands out from others is a stronger focus on data collection.

2.4 Other Factors

As for quality standards, we can turn to Luis Corral's work [8] for analyzing the quality of mobile software products. While the study may be tailored to mobile software products, it's an obvious assessment that mobile phones form the bulk of how users access and interact with content, and this is simply too large of a market segment to ignore. Corral's study brings up a good point about mobile development, namely in the added complexity of constraints like wireless communications, mobility issues, different standards, security, just to name a few. Furthermore, mobile quality is regulated by the manufacturer or OS provider's own set of standards, and actual consumer perception is impacted by factors like reviews and word of mouth. As a result of the aforementioned factors we have mentioned for mobile, it stands to reason that different quality standards might be needed to better evaluate mobile quality.

Corral's study is one of the few studies to provide insight into the problem this study is trying to address, which is defining in straightforward, concise terms what questions to ask. He has three well defined questions, which identify the most relevant requirements, how much the mobile environment impacts work, and the actual quality assessment. In [8], Corral has broken his approach into 3 steps, which we can quickly describe.

1. A comprehensive survey of the publishing standards of the major mobile platform owners, which would be iOS and Android for an overwhelming majority of developers. These standards take care of the first requirement, which is defining how well the application in question handles mobile constraints.

2. Conduct an analysis that allows researchers to relate mobile-specific characteristics with quality requirements. Mobile-specific constraints will also be studied through standard techniques like benchmark numbers and performance analysis, presumably tracking metrics like load times, memory usage and animation/transition quality.

3. With these two pieces of information, Corral's Goal-Question-Metric Methodology will then be employed to propose a strategy for quality measurement.

Corral's methodology directly speaks to the three questions that he's raised in his 3 step approach. The first step would be to analyze how well these applications meet the metrics defined within the Goal Question Methodology. The second step is to use a non invasive testing framework to analyze the Java/Swift code across 500 open source applications. These applications must represent a vast array of categories to highlight different use cases, which is an easy thing to do considering both marketplaces already have easy sorting capabilities in place. These quality metrics will then be compared against the number of downloads, the number of commits and releases that are in the code repository.

Wellington F. Gonçalves, Carlos B. de Almeida, Leonardo L. de Araújo, Mateus S. Ferraz, Rogério B. Xandú and Ivaldir de Farias Junior [14] have another study discussing the human factors behind a historically neglected part of the development process, which is testing.

Remo Ferrari, Nazim H. Madhavji, and Mark Wilding take a similar approach to their study [12] as we did to ours, which is to focus more on the non-technical side of software development, and to be more precise, software architecture. The authors echo our initial motivations for this study, which is that most literature in this field is heavily focused on technical issues.

Delving further, the authors of [12] conducted a study on 15 student-level teams, and analyzed problems like procrastination, poor planning and missed meetings. In addition to this, the authors also analysed the current state of the IEEE/ACM software engineering and computer science curricula to find whether there was a dependency between these factors.

The question being investigated here is: what is the impact of non technical factors on Software Architecture? To answer this, the authors conducted an empirical study across 15 teams, each comprised of 4 people. The teams were given the same project, with about 80 high level requirements. The data to identify non technical factors was collected using unconventional means like capturing internal team emails and recording team conversations. The authors identified that being late for meetings was the most observed problem across all teams, and that this was causing communication issues among the group. This was followed by procrastination, poor planning, weak individual contributions and other factors of lower importance like

poor leadership and mistrust across team members. Taking these factors into consideration, the authors did identify a positive correlation, meaning that the teams with the least number of problems performed the best. Having observed that these problems were affecting the quality of the architecture to such a degree, the authors looked to the ACM SE and CS curricula, only to identify that neither particularly covered these non-human factors.

2.5 Survey Design

In addition to data driven studies discussing different factors, we have looked at a selection of studies that specifically focused on the survey design aspect of research. While there was a lot of detail surrounding the data, including collection strategies and threats to the validity as seen in studies like [19] and [16], they have also not provided a lot of insight into the often overlooked, practical side of data collection, which is building out the questions in a way that elicits good data from the participant in a quick and low effort fashion, with detail poured into potential biases and distribution strategies.

Chapter 3

Building Survey Questions

The literature review has shown that this aspect of Software Engineering is vastly under-researched. But we have also seen one consistent answer, which is that the people developing software are just as important as technology. And as the paper "No Silver Bullet" [7] describes, there is no perfect way to build software, just incrementally better ones. This line of thinking could be very easily applied to the people building said software, and for that it is of paramount importance that developers are better understood as humans.

Another common trend in this literature is that (to our knowledge) nobody has really spoken of the practical side of things, which would be the data collection aspect. While previous authors have highlighted the metrics used, their results and their methodology, the actual surveys used to collect initial data are under-researched. This is an issue, since it can be quite challenging to create and deliver effective research surveys.

A recurring thread in all of these studies is how the sample sizes of data have been collected using small, focused efforts of software development as references. There is little literature for research involving real world, human subjects, for a host of reasons. One major problem is that data collection is time consuming, expensive and often met with disinterest from potential participants.

In this chapter, we lay out a template for a survey that has been balanced carefully for length, response time, layout and least possible completion time. This IRB-approved survey (UW-Whitewater IRB protocol number IRB-FY2019-2020-122) is intended to serve as a starter point for future researchers and to provide guidance on data collection where there is none.

This template provides plenty of room for potential modifications to suit future researchers' individual studies.

3.1 Biases and Threats to Validity

The survey and its conclusions are largely driven by human data. Considering some of the more emotional aspects of the survey and the fact that the survey involves participants occasionally thinking back to the past to answer questions, there is scope for a variety of biases that must be considered. While efforts have been made to minimize how much bias creeps in through survey design, this is not foolproof, as a participant letting their biases color responses can never fully be compensated for.

In no specific order, here are some of the biases the National Institutes of Health encountered frequently in their research [22]. We discuss how each bias potentially applies to our research, establishing a baseline for understanding and interpreting our data better.

1. Pre-trial bias: "Sources of pre-trial bias include errors in study design and patient recruitment" [22]. As described in the earlier section outlining survey design, we have designed the survey with a lot of thought put into it so as to best avoid errors. However, unexpected issues while recruiting participants may have introduced some unintentional pre-trial bias on account of our participants potentially being only students versus professionals, as evidenced by our data.

2. Selection bias: "Selection bias may occur during identification of the study population" [22]. We tried to minimize selection bias by sending the survey invitation to a reasonably large group of software developers. However, since participants self-selected into the survey group, there are risks that our survey is affected in unpredictable ways by the same self-selection biases associated with many Web-based surveys [6].

3. Channeling bias: "Channeling bias can occur if one intervention carries a greater inherent risk" [22]. The NIH paper uses the example of surgeons treating fractures in young people more aggressively, while being much more gentle with the elderly. They also

highlight surgeons tolerating imperfect treatments for the elderly more than they would with a younger patient [22]. This bias does not apply to this study, as the data and the survey are completely anonymized and the researcher is not performing an "intervention".

4. Interviewer bias: Again unlikely, considering that data were gathered solely through a Web-based survey that participants completed remotely and therefore none of the data collection has any personal aspect to it.

5. Chronology bias: Not applicable, since the study has no control groups of any sort.

6. Recall bias: "Recall bias refers to the phenomenon in which the outcomes of treatment (good or bad) may color subjects' recollections of events prior to or during the treatment process" [22]. The idea is crystal clear: our survey data is certainly going to suffer from a degree of recall bias, and this is by design. The effects will be more pronounced in the questions that deal with satisfaction, emotional states and confidence in quality of work.

7. Transfer bias: Not applicable, since the survey is a one time affair. Participants are not identified, eliminating any possible way to follow up.

8. Performance bias: "In surgical trials, performance bias may complicate efforts to establish a cause-effect relationship between procedures and outcomes. Surgery is rarely standardized and that technical variability occurs between surgeons and among a single surgeon's cases. Variations by surgeon commonly occur in surgical plan, flow of operation, and technical maneuvers used to achieve the desired result. The surgeon's experience may have a significant effect on the outcome" [22]. This is the closest analogy to software development across this entire paper, and it's easy to see how different experience and skill levels can influence the quality of a software project. But this is also interesting in the sense that this isn't so much of a bias as something we can almost expect to see: it seems fairly obvious that a more experienced developer will produce a software product of higher quality as opposed to a rookie.

9. Citation bias: “Citation bias refers to the fact that researchers and trial sponsors may be unwilling to publish unfavorable results” [22]. I have no financial or publication ambitions apart from this thesis, and by extension no motive to doctor my data or findings in any way.

10. Confounding: Confounding should not be a concern for this study in theory, since this is primarily an exploratory study. It is possible that a variable that is not specifically asked about in the study might be more influential on developer performance than the variables that we are studying here.

Here are other common biases not included in the NIH publication [22], but which are very much relevant to the type of data this study deals with:

1. Self-serving bias: Especially prominent in events of an unsuccessful project, and could lead to negative feelings toward collaborators.

2. In-Group bias: A more favorable assessment of oneself in a group, and a subjectively harsher evaluation of collaborators.

3. Halo Bias: One characteristic potentially overshadowing other characteristic, both of self and a collaborator.

4. Response Bias: The tendency of a person to answer questions on a survey in a misleading manner.

The problem with biases like the four mentioned above is that for a survey where the participants are not known to the researcher, the burden to supply accurate responses falls on the respondent. An easy example could be a respondent introducing In-Group and Self Serving bias for something like a bad day at work, and there’s a limited amount of control the researcher has in edge case scenarios like this one. In conclusion, while we as researchers try our best to avoid errors, are inherent to human qualitative research.

3.2 Survey Design

3.2.1 Consent Check

This is a very short section, and is essentially a formality. As per IRB Guidelines for working with human subjects, this section allows the author of the survey to present a Consent for Data Collection form explaining the participants' rights, data handling methods and measures taken to protect participants' privacy. The survey presents a Yes/No question, and selecting No ends the survey and thanks the participant for their time and interest.

3.2.2 Technical Information

This section has a heavy focus on the technical side of the development, ranging from the method of development, project types, team weaknesses, bugs and feature completion. All of the questions are straightforward, except for feature completion. The criteria for feature completion is asking the team how often the team meets MVP (Minimum Viable Product) or is able to deliver an extremely polished, high quality product. My initial hypothesis would be to assume that:

1. The data would trend towards either an MVP or an MVP with a few added features, with extremely polished products being outliers.
2. A lot of teams either end up adopting agile or waterfall, with some exceptions.
3. A lot of teams work on new products or modifying off the shelf products, or either serve as a software support center for a business where software is an expense.
4. Most teams struggle with communication.
5. Teams will classify about 25% of all their bugs as severe.
6. Most teams will run into severe bugs about half the time.

3.2.3 Communication Methodologies and Frequency Data

This section has some very straightforward questions, and almost all of them have options that are a slight variant of a Likert scale. All questions seek information about participants' communication preferences, frequency and how they feel about them. A bonus distraction checker that is meant to eliminate responses from participants' that are not focused on the survey, and by extension, act as a check against bad data. My hypothesis would be to assume that:

1. Most participants receive either too much or too little communication from their manager.

2. Almost all participants meet their manager at least once a week.

3. Most participants might feel like the amount of communication is either too much (the classic Meeting versus Memo debate) or too little.

4. An Agile team meets for daily stand-up meetings, but most other times might meet about once or twice a week.

3.2.4 Interpersonal Data

This is where the survey shifts from the technical aspects and the broader aspects of the team to the minutiae and the ground level details. Questions include, but aren't limited to: How long does the team spend on projects? How many projects has the team worked on? What is the average age of your coworker? My hypothesis about each one of these questions would be:

1. Most student projects will clock in under 6 months, given the semester structures of most schools, 6 months to 1.5 years for service projects and variants of products that already exist. The 1.5 years option is most likely going to be cutting edge new products where there is not a lot of existing inspiration to draw from.

2. For students, this number might not be high, considering students move sections at a lot of mid to larger sized programs. But for enterprise developers, we can expect a lot more variation.

3. Most student teams will have around 2-5 members, and enterprise teams can vary vastly depending on the size of the organization.

4. The 18-24 age option is a near constant for any developer that identifies as a student, with very few exceptions. But beyond that, there is almost no way to correctly hypothesize enterprise developer ages.

5. The next few questions are all about personal perception towards the collective group, and the general hypothesis would be from neutral to slightly dissatisfied, with extreme reactions being the outliers.

3.2.5 Mental Analytics

We derive the next major part of the data to address my problem here, the human and more specifically, the emotional side of software development. This is also the section with the strongest room for error, on account of how many biases can be involved answering questions that are so deeply emotional and intimate.

Another important fact to note is that while some of the studies we have discussed employed the MBTI model to measure personality and analyze how personality would affect software engineering. MBTI isn't perfect as a measure of personality. Its validity has been brought into question, with researchers suggesting that it puts participants into strongly defined stereotypes. David J. Pittinger [26] explains the reasoning for this in terms of the Standard Error of Measurement, which is a concept that psychologists use to decide how close the scores of a personality test taken at close intervals are. The problem, as Pittinger states, is that the MBTI method obscures this Standard Error of Measurement far too much, and it isn't possible to have any wiggle room on account on how rigid MBTI's dichotomies are. So in response, Pittinger proposes instead that we should analyze moment-to-moment emotional responses, as well as

overall emotional reactions observed during different stages of the development process. In addition to this, Vox [29] published an interview with Adam Grant, an organizational psychologist at the University of Pennsylvania. Adam Grant states that the categories defined within the MBTI structure weren't carefully controlled and developed based on data, but rather based on noted psychologist Carl Jung's personal experiences which then lived on because of Jung's influence on the field. Jung himself went on to state that people are a mix of several of these types, and that "there is no such thing as a pure extrovert or a pure introvert. Such a man would be in the lunatic asylum" [10].

Is there still a way to capture the effect of the human mental and emotional state on software quality? Yes. According to Muhammad Ali Pervez, organizations employing people with better emotional control over negative emotions tend to produce work of better quality, and concludes that emotions do strongly influence job performance [24]. This point is further proven in this blog post by Natalie Mendes, writing for Atlassian's blog. Mendes writes that teams where positive emotions are high, individuals are treated well and where the importance of emotions is understood tend to work better [21]. Atlassian as a corporation is no stranger to workplace productivity and quality, being the creator of tools like JIRA Software, JIRA Align and Confluence, so their impact on influencing the way software teams operate cannot be understated.

In light of this, we have chosen to move away from MBTI to understand the developer's mental workings, and instead opted to measure metrics like team camaraderie and moment to moment emotional states.

Our hypothesis for the answers on these questions would be as follows:

1. Speaking from personal experience and anecdotes from fellow developers, this is probably the hardest to hypothesize, and this question is an absolute wildcard in terms of what to expect.

2. Much like the earlier question, this is a hard question to predict, because of the sheer complexity of the biases involved. One easy hypothesis is that students will likely pick

coworkers, on account of their projects being self-managed. Developers working on products will choose between management or coworkers, and it's a reasonable assumption that this might shift based on how friendly coworkers are to each other outside the workplace.

3. This could be a function of the developer's personality, someone more introverted is likely to enjoy the coding/testing phases, whereas a highly extroverted, outgoing personality could favor meetings and requirement analysis phases just as much.

4. This item could be a function of the organizational size, as a smaller, start-up like organization can be reasonably assumed to have employees that are close to each other, whereas things could be a lot more serious and professional at a much bigger organization.

5. This could vary with seniority, as younger, more inexperienced developers might tend to second guess some of their deliverables, unlike a much more seasoned and experienced developer.

3.2.6 Demographics

This section includes some standard demographic questions. The data will strongly lean to students, simply on account of who I'm surrounded by as a university student. There's no expectations for the rest of this section, as mentioned in the IRB proposal that the survey acknowledges software professionals are the same as students as far as how data is collected.

Chapter 4

Unpacking our Survey Data

4.1 Reading the Responses

This section is lacking, as the factors mentioned in the section that follows outline a failing in our data collection strategy. We have reason to believe that those factors prevented us from getting a large volume of data that was necessary to prove the validity of this survey, and that this is something future researchers can take note of and improve on. But for now, we can talk about the results obtained.

We have obtained a total of 19 responses, with 9 of those being incomplete, bringing us down to 10 complete responses. Of these 10, respondents are 18-34 years old, with the majority of responses trending towards having about a year or less of software development experience.

4.1.1 Team-Specific Technical Information

The first section of the survey is trying to find what development techniques are commonly favored, and Waterfall is the most preferred, with Agile and Scrum following. This is in line with our initial hypothesis, where we expected that a lot of developers would be sticking to Waterfall and Agile methodologies, working mostly on new products.

Student teams tend to pick technologies they are somewhat familiar with or know very well, explaining the results of Question 3. We also observe that student teams do not classify a lot of their bugs as severe, but all respondents run into severe bugs sometimes, which is expected behavior.

Table 4.1 Question 1

Question	Options	Responses
What is your team's most preferred development technique?	Waterfall	1
	Scrum	2
	Agile	5
	Extreme Programming	0
	Product Driven	0
	Lean Software Development	0
	Feature-Driven	0
	Other	0

Table 4.2 Question 2

Question	Options	Responses
What project types has your team worked on?	New Software Development	6
	Enhancement	4
	Modifying Off-The-Shelf Software	2
	Software Integration	1
	Migration	4
	Others	1

Table 4.3 Question 3

Question	Options	Responses
How quickly do you feel your team adapts to new development tools?	Very Fast (a few days to a week at most)	2
	Fast (1-2 weeks)	1
	Moderate Speed (2 weeks to a month)	3
	Slow (more than a month)	2

Table 4.4 Question 4

Question	Options	Responses
Which of these have historically been weak points for your team?	Crunch periods	3
	Communication within the team	2
	Consistent code quality	2
	Communication with stakeholders	3

Table 4.5 Question 5

Question	Options	Responses
How many of your team's bugs are classified as severe?	more than 75 percent of all bugs	0
	50-75 percent	0
	25-50 percent	0
	less than 25 percent	7

Table 4.6 Question 6

Question	Options	Responses
How often does your team run into severe bugs?	Always	0
	Most of the time	0
	About half the time	0
	Sometimes	8
	Never	0

4.1.2 Communication Methodologies and Frequency Data

The data collected from this section is largely irrelevant, as the concept of managers does not really exist in a student environment. The organizational questions are similarly irrelevant to student respondents. As for project duration, this lines up with what is expected of student projects, that are usually restricted to one or two semesters in length.

We can expect that a survey with more experienced developers taking it can yield different results. A potential hypothesis for a data point is that more experienced developers may classify fewer bugs as severe, simply because a more experienced developer might have a different perception of what is severe.

Table 4.7 Question 7

Question	Options	Responses
How would you describe the amount of communication you receive about your organization from your manager?	Far too much	0
	Slightly too much	1
	About right	6
	Slightly too little	0
	Far too little	1

Table 4.8 Question 8

Question	Options	Responses
How frequently do you meet one-on-one with your manager?	Daily	2
	2-3 times a week	3
	Once a week	0
	Once every 2-3 weeks	2
	Once a month	1
	Less than once a month	0

4.1.3 Interpersonal Data

We have already made it abundantly clear that the overwhelming majority of our respondents are in fact student or junior engineer participants. The results from questions 11 to 14

Table 4.9 Question 9

Question	Options	Responses
How satisfied or dissatisfied are you with the amount of communication that you receive about your organization?	Extremely satisfied	2
	Slightly satisfied	5
	Neither satisfied nor dissatisfied	1
	Slightly dissatisfied	0
	Extremely dissatisfied	0

Table 4.10 Question 10

Question	Options	Responses
How frequently does your whole team meet?	Daily	3
	2-3 times a week	3
	Once a week	1
	Once every 2-3 weeks	1
	Once a month	0
	Less than once a month	0

make it even more clear. Participants are not on long term projects: students might end up taking several classes together, so it is not unusual for the same students to be paired on multiple efforts. Recent university graduates who are now junior engineers might not have formed the same mindsets toward their work that their senior colleagues have. This section is also where there is room for potential in-group bias, particularly with Questions 15, 16, 17 and 18.

Table 4.11 Question 11

Question	Options	Responses
How long does your team usually spend on projects?	less than 6 months	3
	more than 6 months to 1.5 years	4
	more than 1.5 years	0

Table 4.12 Question 12

Question	Options	Responses
How many projects has your current team worked on?	New team, yet to start a project	1
	1-3	4
	4-6	3
	7-10	0
	Veteran team, we've done more than 10 together	0

Table 4.13 Question 13

Question	Options	Responses
How many people are on your team?	2	1
	3-6	5
	7-10	1
	11-20	1
	21-30	0
	31-60	0
	more than 60	0

Table 4.14 Question 14

Question	Options	Responses
What age group are the biggest majority of your coworkers?	Under 18	0
	18-24	1
	25-34	4
	35-44	3
	45-54	0
	55-64	0

Table 4.15 Question 15

Question	Options	Responses
How satisfied or dissatisfied are you with the quality of work produced by other members of your team?	Extremely satisfied	2
	Slightly satisfied	5
	Neither satisfied nor dissatisfied	1
	Slightly dissatisfied	0
	Extremely dissatisfied	0

Table 4.16 Question 16

Question	Options	Responses
How satisfied or dissatisfied are you with the ability of your team to communicate effectively with each other?	Extremely satisfied	2
	Slightly satisfied	6
	Neither satisfied nor dissatisfied	0
	Slightly dissatisfied	0
	Extremely dissatisfied	0

Table 4.17 Question 17

Question	Options	Responses
How satisfied or dissatisfied are you with how fairly tasks are shared throughout your team?	Extremely satisfied	4
	Slightly satisfied	2
	Neither satisfied nor dissatisfied	2
	Slightly dissatisfied	0
	Extremely dissatisfied	0

Table 4.18 Question 18

Question	Options	Responses
How satisfied or dissatisfied are you with your relationship with your team?	Extremely satisfied	4
	Slightly satisfied	2
	Neither satisfied nor dissatisfied	2
	Slightly dissatisfied	0
	Extremely dissatisfied	0

4.1.4 Mental Dynamics

This is the part of the survey where the volume of usable responses dropped off sharply. We have data for two questions, with one of them stating that their biggest negative emotions are often invoked by the clients that they work with, and the other stating that the coding/implementation piece of the development process brings forth the most frustration to a developer who is relatively inexperienced.

However, the emotions running through the participants were relatively on the positive side, with some frustration laced in the results. This works with our previous observation of participants being frustrated with implementation and being relatively inexperienced.

4.2 Statistical Significance of Results

A very basic practice when conducting any data-driven survey is to determine the statistical validity of the data obtained. The basic definition of statistical significance is whether your results are merely due to blind luck or some actual factor of interest. While these results are statistically significant on account of our data being the result of a focused set of respondents, these data are not practically significant [13], as the volume is too low and there's missing pieces, making any potential conclusions wrong at worst and inconclusive at best.

Furthermore, this paper by Faber and Fonseca [11] speaks more about the issues of testing with very small sample sizes. The crux of the problem is that a smaller than normal sample

Table 4.19 Question 19

Question	Options	Responses
Which of these emotions did you experience during development?	Happiness	7
	Enthusiasm	7
	Optimistic	5
	Content	3
	Excitement	5
	Frustration	7
	Anger	2
	Hostility	1
	Disappointment	1

Table 4.20 Question 20

Question	Options	Responses
What was the biggest source of hostility?	Co-Workers	0
	Management	0
	Client	1

Table 4.21 Question 21

Question	Options	Responses
What part of the development process typically invokes the strongest emotions in you?	Requirements Analysis	1
	Internal Stand-up Meetings	1
	Coding	3
	Client Meetings	1
	Testing	1
	Deployment	1
	Post release	0

size increases the chance of assuming a true as a false premise. They prove the dangers of this by speaking about a potential scenario where a researcher is testing out a newer, albeit more uncomfortable experimental treatment. The researcher, who only successfully obtained half of the required number of respondents, was unable to guarantee the power needed to extrapolate the statistical findings to a general, large population.

4.3 Practical Problems and Roadblocks

This study, while dealing with a lot of factors, and blending aspects of Software Engineering, Psychology and even Sociology with the data collection part of it, is not without issues. Some of the key issues with the initial data are outlined below:

1. The data collection aspect of my research started right around the time the novel coronavirus that causes the respiratory illness COVID-19 was sweeping the globe hard. This meant an uncertain time of economic recession, general stress and organizations struggling to quickly transition to a Work from Home approach. A 33 question, unpaid survey proved to be a hard sell for a lot of people the researchers interacted with, and was often met with no response. Out of approximately 100 people who received the survey invitation, only 19 responded, and then only about half of those responses (9) were complete. This response rate was much lower than we projected.

2. The participants are all concentrated in the Midwest region, so this study likely cannot account for various cultural and societal differences that inevitably will influence work patterns.

3. The small data size and missing pieces from completed responses here will require a great deal of extrapolation and assumptions on the researcher's part, and the findings will be influenced on account of this.

4. The types of work organizations where our participants work might not be diverse enough, considering that our participant pool only focused in the Midwest region, where many

companies are more service-based. The West Coast—especially the San Francisco Bay Area—is where a lot of product-driven organizations are present, and that is one area this study was not able to target.

5. Responses might be heavily skewed toward agile or waterfall, with student survey submissions following said development techniques even more loosely than professional organizations on account of student schedules and general college work habits.

Chapter 5

Closing Thoughts

Building this study and paper has been a very interesting experience, and has provided us with several takeaways, both academic and practical.

5.1 Practical Takeaways

1. Start designing a survey backwards. Survey design followed by a literature review will end being a lot closer, and the ideal is to not make changes after a period of design and an IRB approval. One such example from our literature review is a set of metrics by Chidamber and Kemerer [3], which we found after creating the survey design. Knowledge of such tools and metrics could have altered sections of our study and potentially improved the validity of our findings.

2. Seek out a grant program that will assist in compensating subjects for their time in the study. The likelihood of quality, completed survey submissions would probably rise substantially when participants are compensated.

3. Starting data collection early and reaching out to a lot more subjects than your intended volume is a great strategy to avoid risk.

4. Having a well thought out plan to deal with junk data or noise early on is instrumental to the integrity of your data, and by extension, the validity of the experimental results.

5.2 Future Research Questions

This study is an attempt to abstract away some of the complexity involved in software data collection and make future research attempts a lot more efficient, in terms of finances and the speed and volume of responses. As outlined above, there have been a myriad of issues with our initial data collection efforts, on account of some external factors beyond the author's control but also some issues with our survey deployment strategy. These issues and initial attempts cycle back to our initial research questions, and also some additional ones, as outlined below:

1. The results had incomplete data even when the participants completed the study? What could have been the missing link? An initial hypothesis would be the lack of an interpersonal element.

2. Would these initial findings look the same when statically meaningful sample size of responses are present?

3. Is there room for the survey to be shortened or focused while still collecting meaningful data?

LIST OF REFERENCES

[1] Manish Agrawal and Kaushal Chari. Software effort, quality, and cycle time: A study of cmm level 5 projects. *Software Engineering, IEEE Transactions on*, 33:145–156, 04 2007.

[2] M. O. Ahmad, V. Lenarduzzi, M. Oivo, and D. Taibi. Lessons learned on communication channels and practices in agile software development. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 929–938, 2018.

[3] Aivosto Oy. Project metrics help - Chidamber & Kemerer object-oriented metrics suite. Web page.

[4] Anderson S. Barroso, Kleber H. de J. Prado, Michel S. Soares, and Rogerio P. C. do Nascimento. How personality traits influences quality of software developed by students. In *Proceedings of the XV Brazilian Symposium on Information Systems*, SBSI'19, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Justin M. Beaver and Guy A. Schiavone. The effects of development team skill on software product quality. *SIGSOFT Softw. Eng. Notes*, 31(3):1–5, May 2006.

[6] Jelke Bethlehem. Selection bias in web surveys. *International Statistical Review / Revue Internationale de Statistique*, 78(2):161–188, 2010.

[7] Frederick P. Brooks Jr. No silver bullet: Essence and accidents of software engineering. *Computer*, 20(4):10–19, 1987.

[8] Luis Corral. Using software quality standards to assure the quality of the mobile software product. In *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, SPLASH '12, page 37–40, New York, NY, USA, 2012. Association for Computing Machinery.

[9] Alpana Dubey and Dhivya Muthukrishnan. An approach for collaborative quality assessment of software. In *Proceedings of the 9th India Software Engineering Conference*, ISEC '16, page 190–195, New York, NY, USA, 2016. Association for Computing Machinery.

[10] Richard I. Evans, Ernest Jones, and Carl G. Jung. *Conversations with Carl Jung and Reactions from Ernest Jones*. Van Nostrand, 1964.

[11] Jorge Faber and Lilian Martins Fonseca. How sample size influences research outcomes. *Dental Press Journal of Orthodontics*, 126(2):619–625, Jul-Aug 2014.

[12] R. Ferrari, N. H. Madhavji, and M. Wilding. The impact of non-technical factors on software architecture. In *2009 ICSE Workshop on Leadership and Management in Software Architecture*, pages 32–36, 2009.

[13] Amy Gallo. A refresher on statistical significance. *Why the Myers-Briggs test is totally meaningless*, 2016.

[14] W. F. Gonçalves, C. B. de Almeida, L. L. de Araújo, M. S. Ferraz, R. B. Xandú, and I. de Farias. The influence of human factors on the software testing process: The impact of these factors on the software testing process. In *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, 2017.

[15] Lucas Gren and Khaled Al-Sabbagh. Group developmental psychology and software development performance. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C '17, page 232–234. IEEE Press, 2017.

[16] Ho-Won Jung, Seung-Gweon Kim, and Chang-Shin Chung. Measuring software product quality: a survey of iso/iec 9126. *IEEE Software*, 21(5):88–92, 2004.

[17] Fabian Kortum, Jil Klünder, and Kurt Schneider. Behavior-driven dynamics in agile development: The effect of fast feedback on teams. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pages 34–43, 2019.

[18] Martin Kropp, Andreas Meier, Craig Anslow, and Robert Biddle. Satisfaction, practices, and influences in agile software development. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, EASE'18, page 112–121, New York, NY, USA, 2018. Association for Computing Machinery.

[19] D. S. Kusumo, M. Staples, L. Zhu, He Zhang, and R. Jeffery. Risks of off-the-shelf-based software acquisition and development: A systematic mapping study and a survey. In *16th International Conference on Evaluation Assessment in Software Engineering (EASE 2012)*, pages 233–242, 2012.

[20] Jason Chong Lee. Embracing agile development of usable software systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, page 1767–1770, New York, NY, USA, 2006. Association for Computing Machinery.

[21] Natalie Mendes. *How do emotions affect productivity? [New research]*.

[22] C. J. Pannucci and E. G. Wilkins. Identifying and avoiding bias in research. *Plast. Reconstr. Surg.*, 19(4):27–29, Jul-Aug 2010.

[23] Mark C. Paulk, Charles V. Weber, Bill Curtis, and Mary Beth Chrissis. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.

[24] Ali Pervez. Impact of emotions on employee's job performance. Ontario International Development Agency, 2010.

[25] V. Pieterse, M. Leeu, and M. van Eekelen. How personality diversity influences team performance in student software engineering teams. In *2018 Conference on Information Communications Technology and Society (ICTAS)*, pages 1–6, 2018.

[26] David J Pittenger. Measuring the mbti...and coming up short. *Measuring the MBTI...And Coming Up Short*, 1993.

[27] Y. Qiu, W. Zhang, W. Zou, J. Liu, and Q. Liu. An empirical study of developer quality. In *2015 IEEE International Conference on Software Quality, Reliability and Security - Companion*, pages 202–209, 2015.

[28] Doug Rosenberg, Barry Boehm, Bo Wang, and Kan Qi. Rapid, evolutionary, reliable, scalable system and software development: The resilient agile process. In *Proceedings of the 2017 International Conference on Software and System Process*, ICSSP 2017, page 60–69, New York, NY, USA, 2017. Association for Computing Machinery.

[29] Joseph Stromberg and Estelle Caswell. Why the myers-briggs test is totally meaningless. *Why the Myers-Briggs test is totally meaningless*, 2015.

[30] L. R. Vijayasarathy and C. W. Butler. Choice of software development methodologies: Do organizational, project, and team characteristics matter? *IEEE Software*, 33(5):86–94, 2016.