An Experiment on the Effects of Native Language Communication on Work Result Quality

Freie wissenschaftliche Arbeit zur
Erlangung des akademischen Grades
"Master of Science"

angefertigt am

Department Informatik
Professur für Open Source Software
Prof. Dr. Dirk Riehle

Friedrich-Alexander-Universität Erlangen–Nürnberg

Eingereicht von:

Kramer, Simon
21567835
International Information Systems

Betreuer: Prof. Dr. Dirk Riehle

Nürnberg, den 31. August 2012

Abstract

The motivation of this work was the research on distributed global software engineering and its challenges concerning linguistic barriers of international software engineering teams by Prof. Dr. Riehle at the University of Erlangen-Nuremberg. It led to the research question, stating whether the communication in a common native-language within one team in software engineering projects does have a positive effect on the work result quality or not, which could help to improve the team compositions in global software engineering projects. This work reflects the importance of this research question in the context of global software engineering as well as the general effects of communication on software engineering work results on the basis of existing scientific publications.

Derived from reliable literature, linguistic barriers could, for example, lead to cultural divergences, misunderstandings with consequences and a shift of hierarchies inside a software engineering team. In addition, linguistic barriers could have an influence on the level of trust among a team, which might have direct effects on the results of the work. In order to determine these findings, this empirical research does examine the hypotheses of how communication in a native language affects the software and requirement quality of different SCRUM projects. It also investigates the level of trust as well as the personal satisfaction of each team member of these projects. The SCRUM projects were conducted as a quasi-experiment on different native and non-native speaking team compositions of national and international students.

The results of this quasi-experiment mean that the communication in a native language within a software development team correlates positively with the results of work quality as well as with the level of trust among the team members. A positive effect on the personal satisfaction could not be detected. Due to its experimental design, these results only have internal validity. Hence, this experiment can be seen as an exploratory research which should provide a base for future research.

Keywords: software engineering, SCRUM, language, language barriers, native language

Table of Contents

Abstract I
Table of Contents II
List of Abbreviations IV
List of Figures V
List of Tables VI
Chapter 1 Introduction 1
1.1 Research Question and methodology overview 2
1.2 Structure of the thesis 3
Chapter 2 Background and Motivation 4
2.1 Distributed software development and globalization 4
2.2 Distances effecting Global Software Development 7
2.2.1 Temporal Distance 8
2.2.2 Geographic distance 8
2.2.3 Social-cultural distance 9
2.3 Linguistic barriers among international teams 10
2.4 Effects of lacking trust and language barriers 13
2.5 Theory and ambitions 14
Chapter 3 Research Methodology 16
3.1 Experimentation in the field of software engineering 16
3.1.1 Humans as subjects examined in controlled experiments 18
3.1.2 Alternatives to controlled experiment 20
3.1.3 Classification of the conducted experiment 21
3.2 Experimental design 22
3.2.1 Definition of the experiment 22
3.2.2 Definition of the wording work result quality 23
3.2.3 Discussion of hypotheses investigated 24
3.2.4 Variable definition 25
3.3 Instrumentation and development environment 27
3.3.1 Experimental setup 28
3.3.2 Team settings 30
3.3.3 Development environment 31

3.3.4 Product owners 32
3.3.5 Developers 33
3.4 Experimental data collection procedures 34
3.4.1 The ‘Likert Scale’ method 34
3.4.2 Pre-Survey 35
3.4.3 Evaluation of software quality and requirement quality 35
3.4.4 Final Surveys 39
Chapter 4 Experimental Operation 40
4.1 Personal observation 41
Chapter 5 Analyses and Results 43
5.1 Pre-survey 43
5.1.1 Calculation of the DEV group weighting 44
5.1.2 Calculation of the PO group weighting 45
5.2 H1: Software quality and requirement quality 46
5.2.1 Requirement quality results 46
5.2.2 Software Quality evaluation 47
5.3 H2a: Analysis of the level of trust survey 50
5.4 H2b: Analysis of the personal satisfaction of the team members 52
5.5 Summary 53
Chapter 6 Discussion 54
6.1 Limitations 54
6.2 Discussion of the results 55
Chapter 7 Conclusion 58
7.1 Suggestion for future research 59
Acknowledgment 60
Literature VII
Versicherung XII
Appendix I – Use Cases XIII
Appendix II – Personal Satisfaction Survey Questions XVII
Appendix III – Level of Trust Survey Questions XVII
Appendix IV – ANOVA Personal Satisfaction XVIII
Appendix V – ANOVA Level of Trust XIX

List of Abbreviations

DSD = Distributed Software Development

DEV = Developer

ELF = English as Lingua Franca

GSD = Global Software Development

GSW = Global Software Work

NS = Native Speaker

NN = Non Native Speaker

PO = Product Owner

List of Figures

Figure 1: Impact of distance (Noll et. al (2010)) 7
Figure 2: Motivation for the research question 15
Figure 4: Communication channels among teams 26
Figure 5: SCRUM Model (Adapted from scrumit.wordpress.com) 29
Figure 6: Total software and requirement quality 50
Figure 7: Show the average value of each team member as well as the average value of each team (ANOVA p > .004) 51
Figure 8: Show the average value of each team member as well as the average value of each team (ANOVA p.003) 52
Figure 9: Result summary radar chart 53

List of Tables

Table 1: Categorization of software experiments (Basili V. (1995)). 22
Table 2: Team settings..... 31
Table 3: Software Quality Evaluation Model..... 38
Table 4: Illustrates the results from the developer groups of each team 45
Table 5: Illustrates the results from the product owner groups of each team..... 46
Table 6: Illustrates the results of the requirement quality 47
Table 7: Software quality evaluation model..... 48
Table 8: Weighted Software Quality and Requirement Quality results 49
Table 9: Software quality based on Defect Rate 57

Chapter 1

Introduction

Beginning in the last century, the subsequent years have shown continuous and increasing movements toward a globalized business. Economic forces are turning the national markets into global markets that demand new ways of collaboration and communication (Herbsleb & Moitra, 2001). Particularly in the IT sector, many local and global companies decided to follow the principles of outsourcing and offshoring (Carmel & Agarwal, 2001). These processes have the objectives of saving labor costs on software development as well as gaining higher-quality software on a shorter development cycle (Holmstrom, Ó Conchúir, Ågerfalk, & Fitzgerald, 2006). According to Gartner (Outsourcing & Strategic Partnerships, 2012), the Worldwide IT outsourcing revenue totaled $246.6 billion in 2011. This is a 7.8 per cent increase from the 2010 revenue of $228.7 billion.

Primarily locally distributed and globally distributed teams request new ways of communication and collaboration for their daily business. In the IT sector, a global software development (GSD) team is assembled from distributed members, who collaborate on a common software project while working across geographic, temporal, cultural, political, and organizational boundaries to accomplish an interdependent task (Smite & Borzovs, 2006). This kind of working environment presents challenges with

respect to communication, coordination, and control (Ågerfalk & Fitzgerald, 2006). Hence, Moe & Smite (2008) argue, "GSD is recognized as being considerably more complex to manage than even the most complex in-house project."

In addition to coordination and control, successful communication among team members is the precondition on distributed projects because large groups of software engineering specialists must communicate their decisions and coordinate their activities to produce a high quality software product to the customers' specifications (Favela & Pena-Mora, 2001). Hence, more often than not these international groups face linguistic barriers due to different cultural backgrounds and dissimilar languages. In the beginning of GSD, the involved software engineers had to handle these linguistic barriers without any or little training on a GSD environment (Favela & Pena-Mora, 2001). Therefore, the networked world has grown closer together and GSD has found its place in daily business as well as in university courses to better prepare software engineering specialists for a multinational work environment. But in particular, linguistic barriers still present key challenges to GSD (Noll, Beecham, & Richardson, 2010).

At the beginning of GSD and also today, an important approach to overcome these barriers is to agree on English as the project communication language to simplify the communication among the global teams (Carmel & Agarwal, 2001). Using a third language seems to fill the gap between team members with dissimilar native languages. But, for example, what are the effects of using a third language as the project communication language on the software product quality? Or thinking the other way around, can full or partial communication in a native-language produce a better software product?

1.1 Research Question and methodology overview

On the basis of the preceding questions, the research question "What are the effects of being able to communicate in one's native language on the quality of work in software development processes?" was formulated and a number of assumptions were made. In this first attempt to investigate this research question, it was required to measure and evaluate whether project communication on native-language or, vice versa, non-native-

languages, does influence the results of work quality in a controlled environment or not at all. Therefore, an experiment on linguistic barriers among different software development teams was conducted. This should provide the data base for this research question.

Twenty graduate students with similar and dissimilar native-languages were the participants of this experiment. The students were divided into four teams. Each team must develop a software product to the same customer requirements. After a development period of seven weeks, the final software product of each team was analyzed and compared to the other products to examine the research question. Additionally, two surveys were answered by the participants to obtain further information for analyzing the research question. This investigation might be interesting for future compositions of software development teams as well as to minimize the risk of low quality software products.

1.2 Structure of the thesis

The chapters describe the implementation of the study to examine the research question, analyzing the results of the conducted experiment and discussing the findings.

The second chapter provides the literature for the research question and the derived theory. It describes the main challenges in GSD and, in particular, the linguistic barrier affected through different native languages. Chapter three discusses experiments in software engineering in general and describes in detail the experiment design of this conducted experiment as well as the data collection procedure for the analyses. A concise chapter four illustrates the experimental operation and the observations of the author during the software development period. Interesting insides are described which, to some degree, match the literature in chapter two. After that, chapter five analyzes the results of the conducted experiment and tests the hypotheses in order to reject or confirm the theory. Chapter six discusses the findings of chapter five and limitations on the study. The last chapter provides several suggestions for further research on this topic as well as a conclusion reflecting the findings in the context of the theoretical background.

Chapter 2

Background and Motivation

To understand the complexity of the research question, it is necessary to describe and to analyze distributed software development (DSD) as the area under discussion and the role of language as a vital element within this context. Thus, this chapter describes the global business movement through DSD, chances, and the challenges mainly influencing the process communication. Communication turns out to be an important factor in a successful software engineering project. Linked to the process communication of a DSD project, the correlation between linguistic barriers and communication will be under discussion in association with the effects of native language. Finally, after illustrating the theoretical background of this study, the chapter illustrates the motivation for the overall theory.

2.1 Distributed software development and globalization

The levels of distribution in DSD can vary from software development teams being located in the same corridor, building, or city to those on different countries or continents (Prikladnicki, Marczak, & Audy, 2006). Allocation of software work to different countries is termed Global Software Work (GSW) or Global Software Development (GSD) and defined by Sahay (2003) as “software work undertaken at

geographically separated locations across national boundaries in a coordinated fashion involving real time and asynchronous interaction." The globalization of business, particularly in the software industry, has become a steady, irreversible movement that rapidly increased the interest in distributed software development in the last decades (Sarker & Sahay, 2004) and is becoming a norm in the software-intensive high-technology industry today (Damian & Moitra, 2006). Advances in communication technology facilitate this trend (Noll et al., 2010). DSD is established in many organizational operations either by offshoring parts of their software development facilities to subsidiaries located abroad or by outsourcing their software development activities to third party organizations. According to Ågerfalk & Fitzgerald (2006), adopting a GSD model should be considered by organizations to access a larger labor pool and local expertise. Further organizations could benefit by using time zone differences in "round the clock" development to take advantage of cycle-time acceleration, cutting IT-labor costs by producing IT in low-cost countries such as Eastern Europe, Latin America, and the Far East, or from the quick formation of virtual corporations as well as virtual teams to exploit market opportunities (Carmel & Agarwal, 2001) .

In the literature, several terms describing virtual teams in a globally distributed environment such as virtual team, international virtual team, and globally distributed team exist (Powell, Piccoli, & Ives, 2004, Ebert & Neve, 2001). Virtual teams are described as a social group of individuals undertaking and coordinating their activities to achieve common goals, share responsibilities for outcomes, interact through interdependent tasks, and operate across geographic, temporal, and organizational boundaries (Powell, Piccoli, & Ives, 2004). Powell et al. (2004) also describes virtual teams as the core building block of a GSD organization. Hence, GSD project success largely depends on the successful implementation and collaboration of virtual teams. For example, an implementation of a virtual team can be achieved by combining technical skills and experience of low- and high-cost center engineers with the focus on performance and reducing costs (Noll et al., 2010). To gain efficiency, the members of a virtual team have to communicate whenever it is necessary (or possible) to understand requirements of the software product, facilitating knowledge transfer between team members, and to help other team members to perform development activities (Ebert &

Neve, 2001). Communication among virtual team members is facilitated by technological advances in asynchronous and synchronous communication, too.

In general, communication is an effective and necessary process for collaboration within projects. In the field of software engineering, it is essential to communicate with each other to coordinate and control projects (Carmel & Agarwal, 2001). High communication quality, as well as frequent communication, is essential to achieve team effectiveness and cohesiveness between developers, managers, and users (Herbsleb & Mockus, 2003). Otherwise, it is critical to meet specified cost, schedule, and software quality goals (Herbsleb & Mockus, 2003). Research undertaken by the Standish Group Inc. in 2009 corresponds with this statement. The Standish Group found out that 24% of all projects fail and at least 44% were highly challenged due to issues regarding cost, schedule, and performance (quality) (Green, Mazzuchi, & Sarkani, 2010). Therefore, for example, the agile software engineering method prefers collocated development teams to enable face-to-face communication in order to realize effective communication. Face-to-face conversation is one of the principles behind the Agile Manifesto in conveying information to and within a development team (Manifesto for Agile Software Development, 2001).

However, in comparison to in-house software engineering projects leveraging collocated teams, DSD and especially GSD projects are recognized as much more complex to manage and several studies indicate that communication among organizations and participants is the main challenge for GSD (Moe & Smite, 2008). For example, within a distributed agile development project, the usage of continuous face-to-face communication is expansive and sometimes simply not feasible due to high travel costs (Carmel & Agarwal, 2001). In this particular case, due to their research on communication and quality in distributed agile development, Green et. al. (2010) recommends adapting a blend of asynchronous and synchronous communication methods and tools, which might be as successful as collocated teams. The following part describes the main influences on communication and possible solutions to prevent these challenges in GSD.

2.2 Distances effecting Global Software Development

According to Holmstrom et al. (2006), "GSD is technologically and organizationally complex and presents a variety of challenges to be managed by the software development team." A GSD unit cannot function without coordination, communication, and control processes. It is believed that in a GSD organization these processes are mainly challenged by temporal, geographical, and socio-cultural distance (Damian D. , 2002). Noll et al. (2010) identifies geographic, temporal, language, and cultural distance as major categories of barriers reported in 24 papers of research on GSD and collaboration, as well. Figure 1 show that distance challenges negatively influence communication, coordination, and control within projects. Furthermore, the distance influences on communication in turn again effects coordination and control.

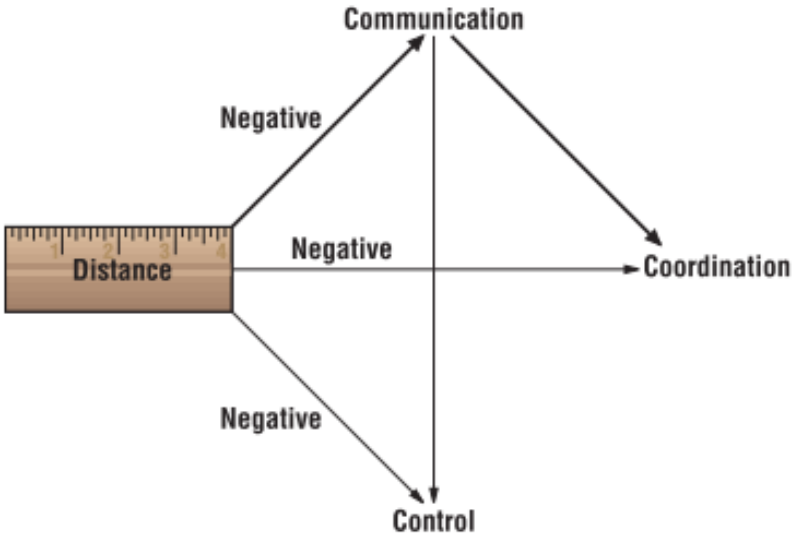


Figure 1: Impact of distance (Noll et. al (2010))

The next sections describe temporal, geographical, and socio-cultural distance challenges effecting communication, coordination, and control. In addition to these three identified dimensions of challenges, other researchers have documented further challenges on GSD. For example, Carmel has identified five centrifugal forces that have the potential to derail GSD (Carmel, Global software teams: collaborating across borders and time zones, 1999); but with many researchers referring to the three

dimensions of temporal, geographical, and socio-cultural challenges, this work will also focus on the three distance challenges. Bird, Nachiappan, Premkumar, Gall, & Brendan (2009) conclude that it is possible to conduct in-house globalized distributed development without impacting the software quality by adapting practices to prevent challenges and improving communication. In his research on a Windows Vista Case study, he attributes the success of the project to a well-structured organization, which prevents emerging GSD challenges. Hence, the three identified distance challenges can also be eliminated, for what reason not only the challenges but also their solutions will be discussed in this section.

2.2.1 Temporal Distance

The temporal distance measure can result from time zone differences caused by distributed development around the globe and from dissimilar time shifting work patterns of the virtual team members; for example, in Germany and Spain (Sarker & Sahay, 2004). Ågerfalk et al. (2005) characterize temporal distance as a measurement identifying the dislocation in time experienced by two GSD collaborators. As a consequence, temporal distance can be seen as a delay in response time and the reduction of opportunities for synchronized communication between GSD actors (Sarker & Sahay, 2004). For example, Herbsleb & Mockus (2003) observed that a change request on software functionalities takes twice as long to be completed in a distributed environment than in a collocated Hence, working within the same time zone or time zone bands, for example, within Europe or a collaboration of Europe and South Africa, facilitates synchronized communications. Suitable solutions for the temporal distance challenge could be the “follow-the-sun” development, which forwards certain tasks under construction from side to side to take advantage from temporal distance (Holmstrom, et. al, 2006).

2.2.2 Geographic distance

Geographic distance is the distance between two actors of GSD projects. This distance is measured in the ease of relocating resources from one location to another; for example, by air links (Ågerfalk, et al., 2005). Thus, it is likely that one potential location is closer on a km perception, but the second potential location provides a better

air link or railway connection and thus, the second location fetches better. High geographic distance reduces the share of explicit knowledge and the possibility of developing a personal relationship between team workers because distributed teams must use mail, video, or teleconferencing technologies to communicate among one another (Noll, et al., 2010). Holmstrom et al. (2006) finds that actors among high geographical distance projects have difficulties in establishing trust and creating a feeling of "team-ness" without face-to-face communication. Geographical distance can be overcome by spending time on travel, which on the one hand facilitates face-to-face meeting, but on the other hand is very cost intensive. Additional company structures and services like profile information of every team member in the company internal websites, near shoring, and taking advantage of modern communication tools for synchronized communication such as video and teleconferencing technologies is seen as the best solution for collaboration across geographical distance (Noll, et al., 2010).

2.2.3 Social-cultural distance

The last challenge of GSD is social-cultural distance, which includes language barriers as a central element (Noll, et al., 2010). However, language barriers do play a crucial role in the motivation for the research question of this work. The subchapter "Linguistic barriers in Global Software Development" describes language barriers and their effect on communication in detail. In this section, cultural distance, which is the second, central element of social-cultural distance, will be described.

Social distance is described by Ågerfalk et al. (2005) as a measurement of understanding between the actors' values and normative practices as well as differences in organizational culture and national culture (Carmel & Agarwal, 2001). Organizational culture does include applying methodologies and project management practices in organizational systems. National culture takes individual motivation and work ethics, language, ethnic group norms, and politics into account (Carmel & Agarwal, 2001, Holmstrom, et. al, 2006). It is possible that two actors do have a high social-cultural distance but share a common organizational structure, and vice versa.
Combining the social-cultural distance with the temporal distance allows several different combinations. For example, the USA and Australia have a large temporal

distance because of different time zones, but have a similar cultural and language background. In turn, the European and South African people share the same time zones but have dissimilar cultural backgrounds. Cultural distance may lead to different interpretations and misunderstandings; for example, when Europeans misinterpret a polite expression of acknowledgment by an Asian colleague as commitment or agreement (Noll, et al., 2010). Likewise, different norms in software development processes could be a cultural difference and might be lead to conflict approaches or could be misinterpreted as incompetence or discourtesy (Noll, et al., 2010). There are different approaches by organizations to solve issues relating to cultural distances. One approach is an offshore-onshore bridgehead where 75 percent of the personal work occurs offshore while 25 percent of the work remains onshore on the customer side (Carmel & Agarwal, 2001). This structure facilitates closeness to the customer while optimizing cost savings by offshoring the development work. Another arrangement could be the cultural liaison by an internationally experienced project manager, who travels between the stakeholders to facilitate the cultural, organizational flow and prevents cultural misunderstandings (Carmel & Agarwal, 2001).

2.3 Linguistic barriers among international teams

Challenges on effective and good communication in GSD projects can occur in various ways as described in the previous subchapter. Moreover, linguistic barriers among team members do influence the communication quality and quantity within projects (Holmstrom, et. al, 2006). Linguistic barriers are a central element of the social-cultural distance described by Holmstrom et al. (2006). Due to the fact that multinational team members speaking different native languages and have an individual lingusitic and educational background, linguistic barrieres can be characterized as a major challenge on communication within GSD. Furthermore, linguistic barriers could occur through different organization languages as well as cultural distances, such as dissimilar verbal business etiquette.

The native language (first language or mother tongue) is defined as the language a person has learned from birth and which is usually given from his or her parents (Bloomfield, 1995), or the language that a person speaks the best. It is also possible that

a person speaks more than one native language; for example, in case of the parents speaking different native languages. The native language of a child is part of its personal, social, and cultural identity (Bloomfield, 1995) and due to an element of social-cultural distance (Noll, et al., 2010). This statement might imply cultural habits in the way of communication.

Clearly, team members in multinational teams with different native languages need a bridge for effective communication. To overcome the language barriers, many global organizations have implemented a lingua franca (Noll et al., 2010). Lingua franca is defined as a language which is used to communicate between persons who do not share the same native languages (Lutz, 2009). Typically, the lingua franca is a non-native language for the speaker. Non-native language, or foreign language, is the language, which is spoken in another country or culture and, for example, taught in an individual's school or at the university that is attended (Bloomfield, 1995). It is the second language a person speaks in addition to his or her first language. Since the Second World War, English has become the lingua franca dominating politics, science, and business (Bloomfield, 1995). The rise of English as lingua franca is well documented and due to its global popularity, English has also become the standard lingua franca on GSD projects. Various decision makers see the advantage of a common language linked to the project success. Hence, language background and skills are important factors; for example, offshoring IT work to countries with strong English competences such as Singapore, India, and the Philippines (Carmel & Agarwal, 2001).

However, communication among teams with members who speak different native languages could cause various problems even when they use a lingua franca such as English. In the past, much research had been done on the native speaker (NS) vs. non-native speaker (NNS) conversation, for example, on comprehensible input, because NS-NNS conversation often leads to misunderstandings or even non-understanding (Long, 1982). Lutz (2009) argues that a misunderstanding of the spoken words is much worse compared to a non-understanding of the whole sentence. A sentence that is not understood by the other team members will usually be discussed again. In contrast, misunderstanding an instruction during a business meeting could be processed without a discussion afterwards. Lutz (2009) also observed interesting tendencies in his lesson

learned in international Software Development Division. In this context he named linguistic false friends caused by linguistic similarities in slightly different native languages of team members with the same cultural background; or similarities between the native language of the team members and the lingua franca implemented by the organization, which might cause false friends. One example are the German words “bekommen” and ”aktuell” and the English words “become” and “actual”, which have dissimilar meanings.

Another challenge in using English as lingua franca (ELF) could occur if a NS or a very good NNS of English is part of a project team. Team members with poor language skills could feel ashamed or be reserved because of their reduced communication style and speak less than the others (Lutz, 2009). There is also the feeling of losing power in collaborating with a native speaker when team members have different levels of English skills (Noll, et al., 2010). In turn, the native speaker is or thinks he or she is of a higher status, especially if a native speaker has foreign talk experience or the conversation occurs spontaneously. Due to this, Long (1982) mentioned that in this particular team combination it is even worse for native speakers because they are not aware of what non-native speakers of English are talking about. He continues that idioms and communication style by non-native speakers are hard to understand and prone to intercultural misunderstandings (Ebert & Neve, 2001). Consequently, native language proficiency could be bad for communicative success in lingua franca settings; for example, ELF communication should be restricted to a core vocabulary and simple grammatical constructions (Long, 1982). A general agreement by research is that NS’s use a reduced variety of their own language if they are speaking with a non-native speaker. In many cases, this kind of communication is ungrammatical because articles and copula are deleted by the native speaker, which might lead to several misunderstandings (Long, 1982).

These misunderstandings like power shifting, linguistic difficulties, and adaptations by NS and NSS could lead to big process and implementation faults, which results in a decrease in the level of trust among the project team members (Holmstrom et al., 2006). In the literature, different linguistic and organizational approaches are discussed as solutions to handle linguistic barriers within GSD; for example, asynchronous

communication (email) can help to improve communication because the team member has much more time to answer a question. Additionally, Lutz (2009) recommends both various training in the organizational lingua franca and asynchronous communication, which is easier to understand than oral communication. Nevertheless, asynchronous communication does not fill the gap of face-to-face communication, which allows virtual teams an informal conversation to develop working relationships and trust among the team members. The following subchapter describes the importance of trust within GSD and the effects of language barriers in establishing and maintaining trust.

2.4 Effects of lacking trust and language barriers

Beside temporal, geographical, and social-cultural distance dimensions affecting GSD projects, lack of trust is an additional factor relating to a decrease in productivity, quality, and overall team morale (Moe & Smite, 2008). The lack of trust might result in an increase of relationship conflicts, the prioritization of individual goals over group goals, and competitions between sub teams. The impact on trust among the virtual team members is also influenced by temporal, geographical, and social-cultural distance (Noll, et al., 2010, Moe & Smite, 2008). For example, in particular, the lack of face-to-face meetings as a result of geographical distance goes along with cost saving strategies by the management. Moreover, the effects of poor EFL skills as well as cultural distance like it is described in the previous subchapter similarly reduces the level of trust (Moe & Smite, 2008). Finally, job uncertainties in GSD also have an impact on trust because many engineers and developers in high-cost locations are afraid of losing their jobs as the result of shifting the software work to low-cost locations. However, Battin, et. al (2001) found that when people are working together in close proximity for a longer period of time, many issues such as trust, perceived ability, and delayed response to communication requests were assuaged.

A study by Muhammad, June, & Phong (2007) on Vietnamese developers might confirm the statement that language barriers do influence the level of trust. In his study, he asked twelve Vietnamese software developers developing software for Far Eastern, European, and American clients about the main factors influencing trust between them and their international customers. He identifies cultural understanding, the knowledge

of norms, skills in the native language of both parties, and business ethos as being very important in establishing relationships of trust with their customers. For maintaining trust, the cultural understanding and the communication either in the customer or the software developer's native-language, were critical factors in preserving trust with the customer. Indeed, this study was compared to a study on Indian developers by Oza, Hall, Rainer, & Grey, (2005), which asked more or less the same range of questions to the Indian developers. Both studies agreed that initial face-to-face meetings are vital to gain trust in relationships; but the Indian counterpart has a different view on the role of "cultural understanding" in particular languages. Compared with the statements of the Vietnamese software developers, for the Indian developers, speaking in the customer's native language or vice versa was not an important criterion for establishing and maintaining trust. The explanation for that disagreement might be the familiarity of Indian culture and English language due to the Indian history and the majority of US clients according to Muhammad et al. (2007).

2.5 Theory and ambitions

Summarizing the findings of chapter two, DSD and, in particular, GSD, on the one hand provide various chances for IT-industry organizations related to cost savings, knowledge expansion, entering local markets, and multinational cooperation. On the other hand, GSD brings several challenges with it, which critically meet specified cost, schedule, and quality goals. One of these described challenges is social-cultural distance triggered by international team compositions from different cultural backgrounds and with different native languages. Linguistic barriers are an element affecting social-cultural distance and are mainly influenced by the team members' native language and non-native language education. Additionally, the organizational language does play an important role. As described in the section 'Linguistic barriers among international teams', linguistic barriers between NS and NNS could create misunderstandings, non-understanding, and a shift of power among the projects teams. These factors might lead to faults and to a decrease of trust. In turn, establishing and maintaining trust among team members tends to be influenced by cultural background as well as linguistic barriers.

In general, this theoretical knowledge leads to the assumption that language does play an important role in today's software development. In particular, NS vs. NNS communication among international teams does directly influence the social-cultural distance of GSD project teams and might manipulate the level of trust among these teams. Therefore, the direct influence of linguistic barriers and the possible reduced level of trust could be challenges for communication, collaboration, and control, which are central factors for the project success and for the results of the work. The structure from Figure 2 is derived from the findings above and illustrates the possible influences of linguistic barriers on DSD (GSD) success.

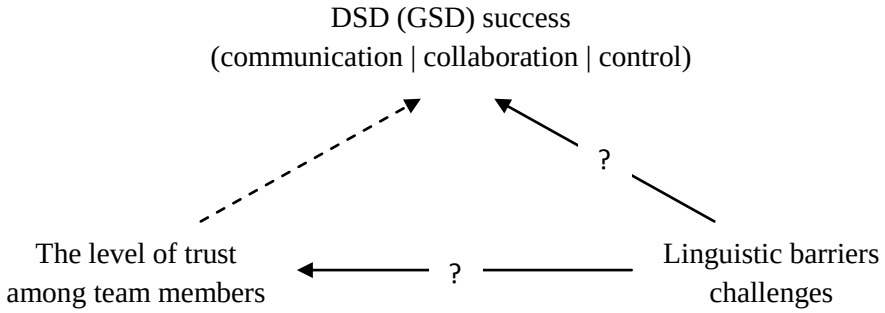


Figure 2: Motivation for the research question

Due to the knowledge that language barriers could affect the project success and might be initiated by negative NS-NSS communication correlation, this work assumes the theory that the communication in one's native language correlates positively with work result quality in software development processes. Until now, there is neither a related work available, which examines this particular research question, nor a measure of the effects of the project success influenced by different NS-NSS communication situations. Hence, this empirical research conducts an initial experiment on this area. It observes whether differences on the team performance of NS-NS, NS-NSS, or NSS-NSS teams are observable on the working results quality or not. Furthermore, correlations between linguistic barriers and trust are analyzed in this study to determine the effect of linguistic barriers on trust. Both findings might provide a basis for further research on this topic. Additionally, the study analyzes the level of personal satisfaction of the team members to determine if different team compilations of native and non-native speakers might have an effect on personal satisfaction or not. This might be an indicator for problems in the team.

Chapter 3

Research Methodology

This chapter describes the research methodology of this empirical research. First, it explains experimentation in the field of software engineering in general and the importance of conducting a controlled experiment to determine the research question. Derived from this context, the conducted experiment will be classified. Second, it provides an overview of the experimental design. This subchapter includes the characterization of the conducted experiment, the definition of the wording 'work result quality' according to the conducted experiment as well as the explanation of the major and minor hypotheses. These were generated to test the overall theory, including the dependent and independent variables. Third, it describes the instrumentation and experimentation environment as well as the organization and team settings for the empirical research. Finally, this chapter describes the data collection procedures such as the pre and final surveys and the valuation methods for proving the three hypotheses.

3.1 Experimentation in the field of software engineering

"Software engineering is a laboratory science" (Basili V. , 1996). Consequently, empirical studies are vital for software engineering to become a mature science like physics, medicine, and manufacturing (Basili, Selby, & Hutchens, 1986). A scientific

process provides a basis to increase the knowledge and understanding within a specific scientific field (Wohlin, et al., 2000). As in every discipline, experiments are also central to the scientific process in software engineering. Experiments provide a classical method to identify cause-effect relationships between components by controlled experimentation (Wohlin, et al., 2000). According to Basili, well-known for measuring, evaluating, and improving the software development process, "experimentation in software engineering helps us to better predict, understand, control, evaluate, and improve software development processes and products". He adds that "we must learn from application and improve our understating (Basili V. , 1996)."

The role of experimentation within software engineering has increased over recent decades, but there is still more scope for further experimentation in software engineering (Wohlin, et al., 2000). In addition, much research on the field of software engineering experimentation criticizes the quality of various experiments, which were processed. Basili et al. (1986) categorized and described more than 100 experiments in software engineering that have been performed in the seventies as well as in the early eighties. Based on this categorization of the research papers, he identifies various problem areas in software engineering experimentations in the sections of experiment definition, planning, operation, and interpretation. It is important to prevent these identified problems in each state of the experiment cycle in order to conduct a proper experiment (Wohlin, et al., 2000).

Basili et al. (1986) argues that in the definition section, a more precise specification of the problem under observation is required, for example, in terms of a common understanding of software quality. The reason for this is that there could be many different interpretations and perceptions of software quality deriving from different perspectives. Also, he underlines the importance of including a series of experimentation for exploration, verification, and application in the planning process to maintain a common sense of research and for further experiments. Additionally, Tichy, Lukowicz, Prechelt, & Hein (1995) recall the importance of external validation of the findings. In their quantitative study they investigated more than 400 scientific articles for the level of experimental validation and they discovered that only a few findings of conducted experiments will be validated in further studies. Therefore, they repeat that

the research in the field of software engineering should keep in mind that the empirical validation should always be an important part of an experiment; otherwise, the results are not valid if no empirical evidence has been provided. Basili et al. (1986) also found discrepancies in the experimental operations. He criticizes that the collected datasets of many examined papers have not been defined carefully to compare the records with other projects for external validation. Wohlin et al. (2000) agrees with this statement and argues that researchers should be careful about the definition, validation, and communication of the data, because they play a vital role in the experimental process. Finally, Basili et al. (1986) continues that a proper presentation of experimental results supports a correct interpretation. Therefore, a presentation of the results should include appropriated qualification feedback by developers and researchers from different platforms to support a suitable interpretation of the results.

Summarizing the literature of this subchapter, Basili et al. (1986), Wohlin, et al. (2000), and Tichy et al. (1995) agree that not only the quantity of experiments but also the quality of experiments should be improved in software engineering experimentation. Furthermore, they found out that too many software engineering experiments are performed once and stay isolated without any replication. Thus, Basili V. (1996) conducts that this kind of experiment "…does not lead to a large body of knowledge." It can be drawn from the literature that a strictly controlled design of the experiment and a detailed description for further replication is necessary for the experiment conducted in this study.

3.1.1 Humans as subjects examined in controlled experiments

Basili et al. (1986) made the point that there are many factors such as cost and quality goals, methodology, experience, problem domains, and constraints that could vary in software engineering environments. Nevertheless, he underlines the importance of human capital within the projects because each person examined can make enormous differences. Schach (1993) categorized software engineering experimentation examining persons as subjects as "experimentation-in-the-small' and "experimentation–in-the-large" to denote experimentation in the areas of "programming-in-the-small" and "programming-in-the-many", respectively.

An example of an "experiment-in-the-small" could be an experiment conducted by a professor with a class of computer science students. Therefore, a computer science assignment might be used as an example to determine different programming techniques by two pairs of student groups. The single grade of every student analyzed by appropriate statistics might be the result of the experiment. Also, professionals could be the subjects of an 'experiment-in-the-small' as described by Basili et al. (1986) and Schach (1993). In turn, an "experimentation-in-the-many" cannot be performed with a small group of people such as an "experimentation-in-the-small" (Schach, 1993). He claims that "experimentation-in-the-many" is necessary for true experimentation. Hence, at least two large teams, each with more than twenty team members, must work with a new programming method, which must be observed by an "experimentation-in-the-many". According to Schach (1993), there is apparently no trouble-free approach for a controlled experiment, because of high costs and a variety of programming skills of each participant. For example, high costs incurred by programming processes, which must be performed at least twice for each team or multiple versions of the software are created by the teams depending on the experimental design. Moreover, it is very difficult to convince the software industry to spend resources and money on testing if there is no direct value for the particular industry or customers. In addition, differences in the abilities of the software developers, as well as other disruptive factors such as language, can manipulate the outcome of the experiment. Apart from the knowledge and experience of the participants, trouble can arise if a team member resigns, is fired, or is transferred to another organization (Schach, 1993).

Many empirical studies in software engineering literature have used students as the subjects of their experiments (Carver, Jaccheri, Morasca, & Shull, 2003). Researchers often keep in mind the reduction of technical and organizational risk compared to empirical studies in the industry. Hence, Basili V. (1996) distinguishes between "novice", who are "students and individuals not experienced in the study domain" and "experts", who are "practitioners of the task or people with experience in the study domain" as subjects of the software engineering experiment. However, conducting an experiment with students is seriously discussed in the literature (Schach, 1993). Schach (1993) claimed that in student teams, the work is usually done by one or two members of each team, instead of all three team members. Often, the students have different

motivations to participate in an experiment, such as good grades or passing the course. In turn, a professional program earns money or at least fame for an open source project. He continues that the scale of a student project can never be as large as that of a professional programmer team; for the reason that in a typical ten week student project, the participants can work in average ten hours per week on the project only, because students often have part time jobs and additional courses. Therefore, he argues that this kind of project can barely be considered as being "programming-in-a-many", for example. A more practical difficulty within student projects is the designing constraints among the classrooms and the fact that students can only meet once per week. In general, Schach (1993) argues that these findings from such projects have no external validity and are not transferable to professional programmers. Whereas Basili, V. (1996) illustrates that "the novice subjects are used to "debug" the experimental design, which is then run with professional subjects."

Conversely, Runeson (2003) identifies significant differences comparing the performance of undergraduate students and graduate students, but the differences between industrial professionals and graduate students are smaller. The fact that students often become practitioners a short time later supports this theory. Another fact which indicates that the distance between students and professionals is not as high is that more and more students are working for the industry during the semester, so the lines between students and the industry are blurred. Besides this, students are fairly more accurate with the protocol of experiments compared to the professionals (Carver et al., 2003). Finally, empirical studies with students can be a helpful preliminary experiment before running an experiment in industrial settings to test the execution of studies and detect and remove problems for cost savings. Höst, Regnell, & Wohlin (2000) mentioned that pedagogical goals and research goals should be harmonized.

3.1.2 Alternatives to controlled experiment

In general, to find interesting patterns and relationships in the field of software engineering, a theoretical or experimental approach is common. A theoretical approach could be, for example, the observation of several programming methods reported in literature and a conclusion drawn from this observation (Schach, 1993). A controlled

experiment is strictly tied to certain experimental conditions like the random assignment of the subjects to different treatments and the execution of the experiment in a laboratory environment (Wohlin, et al., 2000). If these conditions of a controlled experiment cannot be fulfilled, there are several options for empirical strategies on software engineering. A quasi-experiment is used if it is not possible to perform random assignments of the group members to the quasi-experiment. Additional approaches include case studies and surveys (Wohlin, et al., 2000). Case studies are conducted for monitoring projects and assignments in a lower level of control compared to experiments. The aim is to identify relationships among different attributes and to observe specific attributes. Survey results are analyzed for descriptive and explanatory conclusions gathered by interviews and questionnaires (Wohlin, et al., 2000).

3.1.3 Classification of the conducted experiment

There are several attributes of an experiment such as type of result, participants, environment, and the level of control, which can be used to characterize an experiment according to Basili V. (1995). This empirical research was conducted within a laboratory environment (see. experimental environment), but the subjects were not randomly assigned. But due to the high level of control and the university environment, this empirical research can be classified as a quasi-experiment in "vitro". "Vitro" is an experiment under controlled conditions, for example, at the university (Basili V.,1995). The opposite is "vivo", which describes experiments in the field under realistic conditions.

Similarly, for a quasi-experiment in "vitro", statistical inferences need to be applied to determine significant cause-effects in an appreciated way of statistical analyses (Wohlin, et al., 2000). Due to the fact that the expected outcomes of H1 are two measurements for the software quality and for the requirement quality from only four teams (see. Team setting), a descriptive statistics method is adequate. Basili V. (1995) believes that "some simple combination of metrics could easily explain cost, quality, etc." Finally, the experiment is performed exclusively with graduate students, which leads to a "novice" classification according to Basili V. (1995). Basili V. (1995) also characterized an experiment on the number of teams and the projects analyzed.

According to his categorization, this empirical research does fit to the multi-variation project categorization, as it is shown in Table 1.

Table 1: Categorization of software experiments (Basili V. (1995)).

		# Projects	
		One	*More than one*
# of Teams per Project	*One*	Single Project	Multi-Project Variation
	More than one	Replicated Project	Blocked Subject-Project

3.2 Experimental design

This subchapter describes the experimental design including the characterization of the conducted experiment as well as the discussion of the hypotheses; the selection of the independent and depended variables will also be explained. It should illustrate the objective of the study and the purpose for which the experiment was conducted.

3.2.1 Definition of the experiment

In preparation for effective experimental planning, a precise definition of the study is essential (Basili V., 1995). Basili V. (1995) suggests a simplified variant of the Goal/Question/Metric (QGM) model, which characterizes the type of the performed study and moves the experimentation in a better direction (Basili & Rombach, 1988). According to Basili V. (1995), the QGM model "represents a systematic approach for tailoring and integrating goals with models from the software process, products, and quality perspectives of interest, based upon the specific needs of a project and organization." However, Basili's (1995) simplified variant only uses the goal parameters of the QGM, such as the object of the study, purpose, focus, and the point of view for the characterization approach. Adopting Basili's & Rombach's (1988) approach to this work, the objective of the study is to examine the influences of the communication language among international teams in order to improve future (virtual)

team compositions with respect to the results of the work from the point of agile software development.

Therefore, the approach of this experiment is either to confirm or disprove the theory that "communication in one's native language correlates positively with the work result quality in software development processes". This experiment ran at the University of Erlangen-Nuremberg in April and May 2012. Therefore, four multinational student teams were the core of the experiment, from which each team had to develop an application for Android devices to the same requirements specified by a customer. All teams had to perform the agile development process SCRUM to implement the customer requirements according to an agile process. According to the SCRUM alliance definition, each project team was composed of a PO group and a DEV group (The Scrum Alliance, 2012). The following section defines the wording work result quality according to the definition of the experiment.

3.2.2 Definition of the wording work result quality

Since this empirical research analyzed an experiment on a software engineering project, a typical IT-project success criterion was applied to measure the success of each project. As described in chapter two, central success criteria of an IT-project are the accomplishment of the specified cost, schedule, and quality goals according to the literature. However, this experiment only focuses on the quality of the projects' outcome. The reason for this is that the cost factors might not be adequately analyzed within a seven week student project and the scheduled factor would probably have place too much pressure on the students in respect to their grading, but the project outcome was part of the final course grade of each participant.

To determine the quality of every software engineering project, two criteria were defined: The expected work result quality of the experiment was four applications for Android devices, each with various forms of software quality (see Experimental Setup). Therefore, the first criterion of the working result outcome is the quality of each individual Android application. In addition to the software quality, the requirement quality in form of detailed user stories from the SCRUM process was part of the

working result quality. User stories are an essential part of the SCRUM product backlog and are written by the Product Owners (PO's) to describe functionalities of the Android application (see. Product Owners). Software quality as well as requirement quality were analyzed and evaluated independently. However, both results are important to draw a meaningful conclusion, because they reflect the work result quality of the entire team. In addition, the level of trust as well as the personal satisfaction of each team member were measured after the experiment and analyzed and evaluated independently from the software and requirement quality metrics. As described in chapter two, the level of trust might have direct influence on the software quality. Thereby, personal satisfaction might be an indication for problems in the team and the motivation of the team members for future joint software engineering projects.

3.2.3 Discussion of hypotheses investigated

The basis for a successful analysis of a theory is the decision regarding whether the hypotheses are valid or not. The hypotheses derived from the theory are essential for all steps of the experimental design and the included variables (Easterbrook, Singer, Storey, & Damian, 2008). Within this empirical research, the major research hypotheses should test the role of various native and non-native speaker team compositions and its influence on the defined work result quality of the experiment. Therefore, each team has dissimilar NS or/and NNS team member treatments. The detailed staffing process of the teams is described at the section "Team settings" within this chapter.

To relate well to the definition of the work result quality one section before, the hypotheses fall into two categories. The goal of the investigation is to support or reject both categories whereas the first category is the crucial factor to attempt or to reject the null hypothesis:

> **H0**: Being able to communicate in one's native languages does not correlate with the quality of the work result.

The major category includes the direct influence of linguistic barriers on the software and requirement quality. The hypothesis is as following:

Major:

H1: Being able to communicate in one's native language correlates positively with the quality of the work result

The minor category consists of language barriers and their impact on the level of trust, which in turn might have influence on the software quality according to the literature of chapter two. Furthermore, the personal satisfaction of each individual participant is measured within the minor category. The hypotheses are as following:

Minor:

H2a: Being able to communicate in one's native language correlates positively with the level of trust among virtual team members

H2b: Being able to communicate in one's native language improves personal satisfaction in the project

Hence, the empirical research tests three individual hypotheses and evaluates the result. The following describes the treatments on each independent variable to test H1 as well as the expected outcome of the dependent variables.

3.2.4 Variable definition

The experiment tests H1, whereas two additional surveys are carried out to test H2a as well as H2b after the experiment. Both surveys were related to the experiment and described in the section "Experimental data collection procedures." The experiment had three independent variables that were manipulated to measure the effect on two dependent variables.

Independent variables:

As described in the chapter "Background and Motivation", there could be three different ways of communication among NS and NNS in general. The first way is to communicate exclusively in the native language, which all team members have in common; for example, to speak German if all team members are native speakers of

German. The second possibility is that a team consists of team members with similar and dissimilar native languages, so that only some team members are able to communicate in their native language. An example for this possibility would be that a team consists of two native German speakers and the other three members speak a little or no German at all, so that the two Germans can just communicate with each other on the same level. The third alternative is that all team members speak dissimilar native languages; for example, Spanish, Turkish, French, German, and Chinese. In the second and third cases, the teams could use a so-called lingua franca such as English to communicate between a native and a non-native speaker

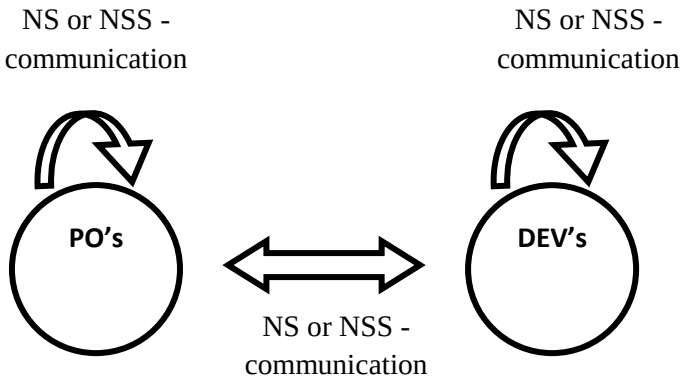


Figure 3: Communication channels among teams

Since a SCRUM team consists of a PO group and a developer (DEV) group, three different communication channels can be identified: the communication among the PO group, the communication among the DEV group, and the communication between DEV's and PO's. Each of these described communications can either take place in a native language or in a non-native language, which depends on the composition of the PO group and the composition of the DEV group. The PO group and DEV group could either be composed of group members with similar native languages or of group members who speak a dissimilar native language.

An example of a team compilation could be that the PO's were native German speakers and the DEV's speak different native languages such as Chinese, Italian, and Spanish. In this case, the method of communication between the PO's would take place in a native language, the communication between the DEV's would take place in a non-

native language, and the communication between the PO's and DEV's would also take place in a non-native language. According to this example, there are three identifiable independent variables: communication within the PO group (PO communication), communication within the DEV group (DEV communication), and communication between DEV's and PO's (Team communication), whereas all three of these independent variables could adopt the treatment's native language (NS) or non-native language (NNS). If the communication took place in a non-native language, the lingua franca used was English. The independent variables as the communication channels among a SCRUM team are illustrated in Figure 4.

Dependent variables:

Requirement quality and software quality are the main outputs of the PO's and DEV's roles in a software engineering SCRUM project. Particularly in a SCRUM project, requirement quality plays a vital role, because it is a central part of the agile SCRUM process. Many parts of the user stories will be specified in detail during the software development process and not in advance, as with the V-model. The software quality depends on the specification of the user stories, but there are also several other factors influencing the software quality, which are described in the chapter "Background and Motivation". Therefore, to measure the entire work result quality of the teams, both software quality and requirement quality are essential.

3.3 Instrumentation and development environment

This subchapter describes the experimental setup and the development process in detail, which is essential for a possible future replication of the conducted experiment. The experiment ran as part of the AMOS lab course. This course is held each semester by Prof. Dr. Riehle at the University of Erlangen-Nuremberg. The contents of the course are agile processes, more specifically SCRUM and technical (favorite XP) practices (Riehle, 2012). Theoretical knowledge combined with practical knowledge from various labs is taught during the semester. The course language is English due to the large proportion of international students.

3.3.1 Experimental setup

The general experimental setup consists of four software engineering projects based on the SCRUM agile development process, corresponding to the course content of agile processes. The goal of each software engineering project team was to develop an Android application for an open social networking portal by the same requirement. The requirements were defined and provided by a customer.

The customer role was performed by the author of this work. Agile development processes such as SCRUM are used in an environment of changing requirements and uncertainties (Green, Mazzuchi, & Sarkani, 2010). Therefore, additional information according to the requirement specifications was distributed to all teams by the customer during the sprints. Emerging questions from the teams concerning the requirement specifications were answered by the customer from a customer perspective. However, there was no SCRUM-Master provided for the teams because of a lack of resources.

Since the experiment was scheduled for seven weeks, each team held independent weekly SCRUM meetings (Figure 5). A weekly SCRUM meeting was scheduled with a time of one and a half hours and included a review and a sprint planning session according to the standard SCRUM procedure (The Scrum Alliance, 2012). Participants were responsible for the division of time during their meetings. For presenting the result or new features during the planning or review session, the university provided a projector and laptop, as well as Android Phones for the students. Each team was in charge of selecting the correct media for their individual presentation. During the planning session, the DEV's needed to play planning poker to estimate the effort for a new feature. The planning poker cards were provided by the university. The cards were on a scale from zero to thirteen effort points, whereas zero is no effort for a feature and thirteen is impossible to implement a particular feature within one week.

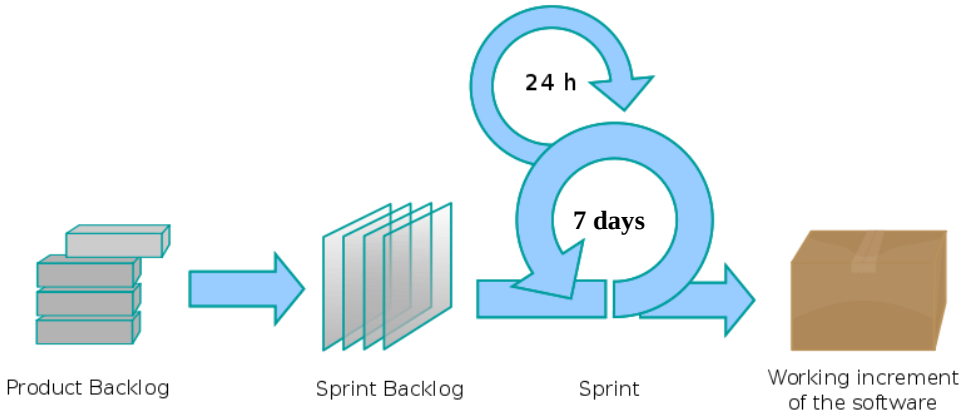


Figure 4: SCRUM Model (Adapted from scrumit.wordpress.com)

The customer observed each of the weekly SCRUM meetings; the reason for this procedure was first to answer emerging questions by the PO's concerning the requirement specifications, and second, to guarantee a sufficient laboratory process. Emerging questions and answers of one particular team were not forwarded to the other teams. The PO's also had the possibility to ask the customer questions regarding the requirements during the weekly sprints via mail. There was no restriction on asynchronous or on synchronous communication among the team members during the sprint. Additionally, daily SCRUM meetings and communication channels were not observed and analyzed.

For the weekly SCRUM meeting, all teams were located in different rooms at a different time at the University of Erlangen-Nuremberg. The meetings were held on Wednesday and Thursday. The schedule was assigned in collaboration with each team member because of overlapping university courses. The teams worked collocated and dispersed at the university or at other locations during the sprints. In addition to the four experimental teams, there was one additional team, which waslocated in Germany and China. The PO's within this specific team were located at the University of Erlangen-Nuremberg, whereas the DEV's were located at the University of Beijing. Due to this fact, the experimental results of this particular team could not be used in the overall outcome because the environment is not equal to that of the other four teams. However, this kind of experimental setup could be the next step for further research in this area.

3.3.2 Team settings

In total, there were four examined teams; two heterogeneous teams and two homogenous teams. Homogenous teams had either team members with similar native languages or team members with only dissimilar native languages. Thereby, a heterogeneous team had team members with similar and dissimilar native languages divided up in the PO group or DEV group. Additionally, there were several important factors to consider for balancing and blocking other influences on the project outcome like experience and education. Next, the step-by-step team setting process is described.

First, all participants in the experiment were graduate students from the University of Erlangen-Nuremberg. The students were from computer science courses as well as from international information systems courses. All participants were assigned to their groups according to a survey conducted at the course registration and their native languages. The course participants were divided into two groups of German NS and non-German NNS.

Second, both groups were divided into the three categories of poor, average, and professional software developers from the knowledge of a survey conducted on the course registration. There was no distinction between male and female as well as no distinction in respect to the age of the participants. However, the grouping by poor, average, and professional software developers was based on very vague information from the participants. Due to this, a further pre-survey that is described in the section “Pre-Survey”, should determine more detailed assumptions of the skills and experiences of each software developer at the beginning of the experiment.

In the third step, the students from the categories German NS and NNS were assigned to the teams (Table 2). Each team was composed of three developers with poor, average, and professional development skills and their native languages. Two participants with poor development skills were assigned as PO’s according to their native language. Therefore, the homogeneous experimental control group has scopes of five NNS with dissimilar native languages and English as their team language. Further, one heterogonous group with two native German PO’s and three non-native German DEV’s

with dissimilar native language, as well as another heterogeneous group with two non-native German PO's and three native German DEV's. Both groups had English as their team language. Lastly, there was another homogeneous team including two native German PO's and three native German DEV's. The team language of this group was German.

Table 2: Team settings

TEAM-ID	**PO's** (NS/NNS)	**DEV's** (NS/NNS)	**Team language** (NS/NNS)
FSA10	NS	NNS	NNS
FSA31	NNS	NS	NNS
FSA11	NS	NS	NS
FSA23	NNS	NNS	NNS

Leftover students were given different tasks. It is important to know that the PO's neither worked in the SCRUM development process before nor wrote a user story. Thus, there could be no advantage due to former SCRUM experiences as a PO. This survey is described in the section "Experimental data collection procedures". To gain basic knowledge about SCRUM, there was an introduction on SCRUM for all participants during the kick-off sprint meeting.

Fourth, there was no native English speaker assigned to any group of German NNS. The NNS speak little to no German. However, all of the NNS have an adequate level of English skills, which was determined in the pre-survey.

3.3.3 Development environment

The context of the study was for each team to develop an Android application for an open social networking portal called FSAhoy.com. The portal itself focuses on "…anyone interested in sharing nautical experiences and information" (Free Seas Ahoy, 2012). It provides several functions for sharing sailing experiences; for example, managing sailing trips, location tracking, and sharing individual sailing trips beyond a

social community (not fully implemented yet). The "Free Seas Ahoy!" project is an open-source initiative for the most part by the AMOS lab-course and it has been available online since summer 2011. This year it is developed in the second generation of AMOS students from the University of Erlangen-Nuremberg. In addition to the Android applications, a different group of students had continued developing the social-networking portal. Due to this, the Android Application teams used a stable copy from the first semester social networking portal to work independently. This stable copy includes additional web-service implemented for the Android application.

Each of the four experimental SCRUM-teams had to develop the Android application according to the requirements of the customer. The requirements were handed out as UML Use-Cases (see Appendix I) by the customer. Six different use cases described what the teams had to implement during the seven week sprints. However, the students had no order to develop all functionalities within the seven week sprints, period. Moreover, it was very unlikely to implement all goals of the UML Use-Cases within the seven weeks. To provide a controlled communication platform, each team was assigned to a separate Google Drive folder. The person responsible for the experiment provided all six UML Use-Cases and a SCRUM Backlog template on this folder. The teams needed to use the sprint backlog on the Google Drive folder to guarantee an equal SCRUM process. Moreover, the teams could use this folder as a document sharing platform, for example, to have a look on the product backlog during the weekly sprints or to share architectural designs.

3.3.4 Product owners

According to the SCRUM principles, the PO's were responsible for the business value of the Android application project (The Scrum Alliance, 2012). Therefore, they had to set up the product backlog including prioritized user stories for each feature of the Android application. The PO's had to generate each feature that was necessary to implement the use cases goals and to put this feature into the product backlog. The prioritization of each feature was also done by the PO's, as well as setting up a release plan. The first release of the Android application was directly after the seventh week of the experiment. A second release was at the end of the semester, which was not part of

the controlled experiment. During the review and sprint meetings, they had to manage the product and sprint backlog as well as the feature archive. The sprint backlog depended on the performance of the developers. It was also possible to remove or to split features from the product backlog if the developer had critical arguments against it. In case of any bugs found during the reviews, an additional bug feature was implemented by the product owners. In order to have a quick start to the experiment, the customer provided five user stories. Additionally, a mockup of the menu was provided from the customer at the beginning, but there was no order to use the mockup for design questions. The PO's were in charge of designing mockups for visualizing the Android Application activities.

3.3.5 Developers

The DEV's had to implement each feature provided by the PO's, but the DEV's had the opportunity for critical comments on the features and also the opportunity to call for revision during the sprint planning. In the sprint review meeting, the developer team demonstrates to the PO's what they had completed during the last sprint. Each developer was assigned as review manager at least twice within the experimental period. The review manager was responsible for the tagging of the code, the technical setup for the sprint review, and the feature presentation during the sprint review.

The DEV groups had to implement features into the Android application like reading, updating, and creating the user profile, the trips, or the logbook information. To implement these features corresponding with the context of the use cases, the developers had to use several web-services provided from FSAhoy.com. Furthermore, several mobile services such as GPS and the specific mobile menu navigation should be implemented by the DEV groups. Enabling a quick start to the projects, an Android framework was provided. The framework included an implemented RESTlet connection to the web services and an Android database interface to a DB40 client backend. But the teams were not forced to use the provided framework. Furthermore, there were no restrictions on additional frameworks.

Additionally, to avoid complications, each team had their own server instance with a copy of the FSAhoy.com sever, which provided an independent programming environment for each DEV group. In order to guarantee an equal working environment for the DEV groups, they must use the same code repository. Therefore, four repository accounts on www.bitbucket.org were created for each team.

3.4 Experimental data collection procedures

This subchapter provides an overview of the used data collection processes and methods. The major part of this subchapter is the data collection process of the software quality and the requirement quality. In total, there was one experiment and three surveys carried out to collect data. One survey was before the experiment and two surveys immediately after the experiment.

3.4.1 The 'Likert Scale' method

The survey questions consisted of open questions and questions that had to be answered with the help of Likert scales; this scale is named after its inventor, Rensis Likert (Likert, 1932). It is a psychometric scale commonly used in questionnaires and it is the most widely used scale in survey research; for example, in marketing or social economics (Web Center for Social Research Methods, 2006). The used Likert Scale in this study was a five (or seven) point scale, which allowed the participants to express themselves regarding how much they disagreed or agreed on a specific statement. The advantage of this survey method is that it is fast, cheap, and easy. Moreover, it does not allow for a simple yes or no answer from the participants, but rather a degree of answers (McLeod, 2008). By using this method, a researcher can collect large amounts of data in a short amount of time on paper or in the web. A disadvantage of this method is that participants can affect the outcome by trying to please the researcher or lying to make themselves better (McLeod, 2008); for that reason anonymity should be given to prevent this effect.

The following describes several requirements for the conducted surveys:

- Anonymity of each participant

- Easy accessibility of the questionnaire either via Internet
  or as a hand out during the lecture
- Clear introductions for the participants on how to fill out the survey
- Questions regarding personal satisfaction and the level of trust
  among a team from a proven psychological reference model

3.4.2 Pre-Survey

At the kick-off meeting, a pre-survey was conducted with the purpose of determining the software engineering experiences of each participant and blocking other influences on the experiment. As in the previously described section, the team compositions were based on questions at the course registration. Due to that, this pre-survey should determine an exact experience of each DEV and PO to weighting the experiment outcome if large differences exist in the level of experience among the teams. The pre-survey was divided into three parts. In the first part, questions were asked regarding personal characteristics such as age, nationality, and course of study. The other parts were different for DEV's and/or PO's. DEV's were asked about their level of experience in the field of programming as well as in the field of Android application developing. The programming experience ranged from pass of a software engineering course at university and no further programming skills, to working experience with more than 10,000 lines of code or three years of working experience as a programmer. In addition, the DEV's were asked whether they already developed an Android application and if they answered with yes, they should describe what kind of Android application they had developed. In turn, the PO's were asked if they have IT-project experience and if they had already written a SCRUM user story. If they answered with yes, they had to describe the context of this user story.

3.4.3 Evaluation of software quality and requirement quality

The Android applications were evaluated right after the investigation. As described in the section "Definition of work result quality", both quality criteria are strong indicators of the overall quality. However, the data are collected in different ways. Next the data collection process for both measurements is described.

Requirement Quality

This variable cannot be measured on a standardized metric scale such as the Cyclomatic Complexity Number, because each user story of the product backlog is an individual text, which can only be analyzed with the knowledge of an expert. Thus, Prof. Dr. Riehle made an expert valuation on the product backlog of each team to get a qualitative measurement of the PO's performance.

Software Quality

Good software quality is a basic goal in software engineering (Herbsleb & Mockus, 2003). However, software quality is still a complex and broad term in the literature. If software quality is discussed in the field of software engineering, there are two related but distinct notions. Software quality can be distinguished in functional and non-functional quality (Spillner & Linz, 2005). Functional quality describes how well the quality of a software product fits with the given design, based on the requirements and specifications of the customer. The non-functional quality to the contrary describes to what extent the software product meets robustness or maintainability; in general, the degree to which a software product was produced correctly. For example, the structural quality of the software product refers to non-functional quality and can be evaluated, for instance, through the software inner structure, the source code, and its effect on the architecture.

There are widespread quality models available such as McCall or ISO 9126 (today ISO 25010), but no commonly accepted evaluation model (Spillner & Linz, 2005). Perhaps the reason for that issue is that the developers of the software have another perspective on the product as managers. Software developers mainly focus on the internal structure of a software product (white box), whereas managers or customers have an external perspective (black box) on the software product. Furthermore, each software project is different from other projects. For example, projects might differ in security, performance, or replication aspects. Finally, some people rely on quality metrics and other people don't trust in metrics, but rather in fulfillment of requirements (Spillner & Linz, 2005).

Measuring the software quality of each Android application, this empirical research used an adapted version of the ISO 9126 quality model. The ISO 9126 focuses on functionality, reliability, usability, efficiency, compatibility, portability, and maintainability. Each characteristic includes several sub-characteristics, which specify the characteristics in detail. Each sub-characteristic is further divided into attributes, which are items that can be measured or verified (Spillner & Linz, 2005). This attributes are not in this quality model because of the variety of software products mentioned one paragraph before. Due to that, the paper used several metrics and qualitative evolutions to determine the software quality categorized in the ISO 9126 quality model.

To measure the software quality for the Android application in this study, an open source tool called Sonar was used, which is used in very large open source projects such as Apache or GoogleCode to control the software quality (Sonar, 2012). Sonar is a web-based application and an open platform to manage code quality, which also provides specific support for the Android application. Therefore, the Android application had to build in Maven (The Apache Software Foundation, 2012). In addition to the metrics of Sonar, which give a good internal view on each Android application, there was an evolution of the Android applications on usability and on functionality by the customer. Both characteristics cannot be adequately measured by a tool such as Sonar, for example. Table 3 shows the characterizations used from ISO 25010 and the metrics from Sonar as well as the evaluation from the customer perspective.

To determine the software quality on a comparable scale, points were awarded for each quality metric. The result of a team was divided into predefined categories which correspond to 0, 1, or 2 points. The point-categories are defined by the customer according to standardized maximum and minimum values from the software quality literature (Spillner & Linz, 2005) as well as from a customer perspective. Table 3 explains each metric and point-category used to evaluate the software quality.

Table 3: Software Quality Evaluation Model

	Functionality	Reliability	Usability	Efficiency	Maintainability
Rules compliance index with Findbugs (Gives a ratio between weighted violations by Findbugs and the number of lines of code)		2P:>= 85% 1P:>= 70% 0P: < 70%			
Comments (Number of javadoc, multi-comment and single-comment lines. Empty comment lines like, header file comments and commented out lines of code are not included)					2P: >=50% 1P: >=30% 0P: <30%
Duplication (Number of physical lines touched by a duplication)					2P: = 0 1P: <= 10 0P: >10
Method complexity (Cyclomatic Complexity Number, also known as McCabe Metric)		2P: <= 5 1P: <= 10 0P: > 10			
Package tangle index (Gives the level of tangle of the packages, The best value 0% means that there is no cycles and the worst value 100% means that packages are really tangled)					2P: <=10% 1P: <=20% 0P: > 20%
Correct implemented features (Total amount of features implemented according to the use cases compared to the other teams)	2P: >= 20 1P: >= 15 0P: < 15				
Design (Assessment of Graphic and usability from a customer perspective)			2P: UP 1P: UP 0P: UP		
Time measurement (Manually measurement of user log-in time and profile update time)				2P: <= 2s 1P: >= 3s 0P: < 3s	
Total amount of JUnit Tests (Total amount of features tested)		2P: >= 10 1P: > 0 0P: = 0			

3.4.4 Final Surveys

The level of trust as well as the personal satisfaction among teams has often been the object of the research in the field of psychology and sociology. Hence, there are already various approaches to measure both attributes. Therefore, two reference models from the literature were used to create the questionnaires by meeting the scientific demand. To find out the level of trust among the teams, a modified 5-point scale measure of trustworthiness from Pearce, Sommer, Morris, & Frideger (1992) was distributed to the team members. It was constructed to provide a general statement of trust among teams. The survey questions were modified by Leidner & Jarvenpaa, (1998) who explored the challenges of creating and maintaining trust in a global, virtual team. These questions resulted in responses on a seven points scale and measured as intervals according to the reference model of Leidner & Jarvenpaa (1998). Thereby, personal satisfaction was measured according to a reference model of Wageman, Hackman, & Lehman (2005). This measurement was used by Rogelberg, Allen, Shanock, Scott, & Shuffler (2010) to identify the satisfaction of team members. These questions were responded to on a five points scale and measured as intervals according to the reference model

Chapter 4

Experimental Operation

Software engineering is a process strongly influenced by human interactions and communication, certain differences can occur, especially in agile software development. Hence, this experiment was conducted to measure those differences with a focus on similar and dissimilar native languages among software engineering teams. To obtain comparable results of each team, a controlled experimental setup was essential to prevent other influences as described in the previous chapter. Each team was also observed by the customer to guarantee an equal SCRUM process as well as to offer the opportunity to answer upcoming questions from a customer perspective. The outcomes of this observation are several notes concerning the research question. Additionally, sound records were made during each session.

The personal observation, explained in the following findings does not contribute to the answer to the research question. Instead, it only provides an inside view of the experimental operation and might help in finding additional starting points regarding this topic.

4.1 Personal observation

During the six review and planning sessions, several characteristics of the heterogeneous as well as of the homogenous teams could be observed corresponding to the theoretical knowledge from chapter two. Here, there are only patterns described which had often been repeated during the weekly SCRUM meetings.

The communication in English as lingua franca led to several misunderstandings or even non-understanding during the sprint meetings. These groups who used a lingua franca had many more uncertainties during the review sessions concerning the user stories or the division of labor between PO's and DEV's. For example, there were misunderstandings regarding how functionalities should have to look, which led to revised versions of the functionalities until the next sprint meeting. Also, general discussions occur on who has had to define constraints or error messages (PO's or DEV's). In comparison, there were much more discussions about the technical implementation of functionalities and much more cycles in planning poker if the DEV groups spoke in their native language.

The native language team in particular needed more time for the planning session than for the review session, whereas the heterogeneous native language team spent more time on the review session than on the planning session to talk about ambiguities and implementation mistakes. In addition, there was a noticeable shift of the hierarchies within the heterogeneous groups through the native language speaker group members. One reason for the shift was that the native German speakers within heterogonous groups often switched to German, although they had agreed on English as a lingua franca. In general, the roles of PO's and DEV's were separated more in the heterogeneous than in homogeneous groups. The latter group made a harmonious impression, whereas the heterogeneous groups had developed their monument on their own.

The observations during the sprint sessions show dissimilar communication behaviors in the teams. Summarizing the observation findings, the teams that communicated on a lingua franca had more linguistic barriers, which led to a shift of hierarchy and many

more misunderstandings compared to the native language team. In turn, the native language team had longer discussions in the planning session, which might have positively influenced the working results. The observations currently suggest that the native language SCRUM team might have performed better than the other teams during the sprint meetings.

Chapter 5

Analyses and Results

This chapter analyzes the quality data of the conducted experiment and the responses of the subsequent surveys to extract the most important results. These results are vital in order to support or to reject the major and minor hypotheses. A summary of the experimental results as well as of the survey results are given in the last subchapter.

5.1 Pre-survey

With the results of the pre-survey, two weightings were calculated for each team to compensate differences in the abilities of software development or software project management of the participants. The weightings were calculated independently for the PO-group as well as for the DEV-group in a team. The total points of each group member were aggregated to calculate the total points of each DEV or PO group. Finally, the total group points were scaled from zero to one. In general, the nominated values were calculated using this formula:

$$x = \frac{total\ points\ -\ min.points}{(max.points\ -\ min.\ point)}$$

With the values of this scale, an average value among all teams was calculated for the DEV as well as for the PO groups. This average value was divided by the nominated value of each group to define the weightings of each PO and each DEV group. In section 5.2.2, both weightings were used to weight the software quality results and requirement quality results. Besides the collected data concerning the experience of each participant, further personal attributes were collected, which were not used for any evaluation of the experiment.

5.1.1 Calculation of the DEV group weighting

Each DEV group could receive between zero and twenty-seven experience points from the responses of the pre-survey. Each participant could collect points for his or her "development experience in general" (DEIG), "specific experience in Android development" (SEIAD), and for his or her "general work experience in international teams" (GWEIIT).

For GWEIIT each developer received zero or one point primarily depending on his or her qualitative answer. If a developer had GWEIIT resulting from a profession in a company or projects at the university, he or she received one point. Further, if a developer had experience in SEIAD, he or she received either one point for experience from theoretical courses at the university or three points from a developer profession in a company or projects at the university. For DEIG, each developer received zero, one (Passed the university course "Algorithmen und Datenstrukturen"), two (University IT-project(s)), three (Private IT-project or working experience < 10000 lines of Code (~ 0,5 years)), four (Private IT-project or working experience > 10000 lines of Code (~ 1,5 years)), or five (Working experience > 10000 lines of Code (~ 3 years)) experience points. Some of the participants marked two or three possible DEIG answers, but it was always rated the highest value of experience. For example, a student with experience in "University IT-project(s)" and "Private IT-project or working experience > 10000 lines of Code (~ 1,5 years))" received four points.

The pre-survey results in Table 4 show a nearly balanced presetting of the developer groups. FSA11, FSA31, and FSA23 had initial experience in Android development,

resulting from previous lectures or a profession in a company. In contrast, the GWEIT and DEIG values are nearly balanced for all teams. Therefore, the software quality results in particular from team FSA10 will be positively weighted, whereas the results from teams FSA11 and FSA23 will be negatively weighted, which will be explained in detail in section 5.2.2.

Table 4: Illustrates the results from the developer groups of each team

team-id	n	total experience points (GWEIIT, DEIG, SEIAD)	nominated	weighting
FSA10	3	13 (3, 10, 0)	,481	1,173
FSA11	3	17 (2, 11, 4)	,630	,897
FSA31	3	15 (2, 9, 4)	,556	1,017
FSA23	3	16 (2, 11, 3)	,593	,953
		average nom.:	,565	

5.1.2 Calculation of the PO group weighting

The PO group of each team could receive between two and twenty total experience points. They received one (poor), two (fair), three (good), four (very good), or five (excellent) points depending on their general IT-Project experience (GITPE). Further, zero, one, or two extra points were given if the PO's already had general work experience in international teams (GWEIIT) depending on the qualitative answer. If the PO had GWEIIT resulting from a profession in a company, he or she could receive two points. But for GWEIIT or specific SCRE from theoretical courses at the university, only one point was given. For initial SCRUM experience (SCRE), each PO could also receive zero, one, or three points depending on the qualitative answer; either one point for experience from courses at the university in SCRE or three points from a profession in a company.

Table 5 shows that the experience of the PO's varies among the teams due to dissimilar experiences in GITPE and GWEIIT of each PO group. However, none of the PO's had any initial SCRUM experiences.

Table 5: Illustrates the results from the product owner groups of each team

team	n	total experience points (GWEIIT, GITPE, SCRE)	nominated	weighting
FSA10	2	8 (2, 6, 0)	,300	,958
FSA11	2	7 (2, 5, 0)	,250	1,150
FSA31	2	10 (3, 7, 0)	,400	,719
FSA23	2	6 (1, 5, 0)	,200	1,438
		average nominated:	,288	

5.2 H1: Software quality and requirement quality

The following sections show the quantitative results of the conducted experiment. As described in chapter three, the software quality data as well as the requirement quality data was evaluated based on the Android Applications and the product's backlogs produced by the project teams. In particular, the software quality data was calculated on a software quality evaluation model corresponding with the ISO 91261 standards, whereas the requirement quality was collected by an expert valuation on the product backlog of Prof Dr. Riehle.

5.2.1 Requirement quality results

To determine the requirement quality, the results were calculated on the quality of each user story in the product backlog as described in chapter three. The results were determined for each PO within a team. Therefore, the formula below was used to calculate the quality of the user story within each sprint week he or she performed. Afterwards, the results were aggregated for every PO group and scaled from zero to one.

$A_i = Worked\ (1)\ or\ not\ (0)$

$F_i = \ Function\ completed\ (1)\ or\ not\ (0)$

$Q_i \ = \ Quality\ of\ the\ functionality\ (1\ or\ 0)$

$$\frac{\sum_{i=1}^{6} A_i * (F_i + Q_i)\text{^}2}{6}$$

Table 6: Illustrates the results of the requirement quality

	Team FSA10	**Team FSA11**	**Team FSA31**	**Team FSA23**
Requirement Quality	2,5	2,83	1	1,5
nominated	0,625	0,708	0,25	0,375

5.2.2 Software Quality evaluation

All results illustrated in Table 7 were evaluated on the basis of software metrics and a qualitative validation from a customer perspective according to the use cases. The software metrics ranging from number 1) to 5) and 9) were calculated by Sonar, which is an open source platform to manage code quality, whereas the qualitative validation of number 6), 7) and 8) were assigned by the customer. The customer was already convinced of the right implementation (6) of the features during the sprint review sessions. The calculation of the results of number 7) and 8) will be described in detail to justify the customer validation.

All results were classified into predefined categories ranging from zero to two points, as described in chapter three. In total, every team could gain between zero and eighteen software quality points. Afterwards, the total points of each team were scaled from zero to one for each team.

Table 7: Software quality evaluation model

Metrics & Qualitaive validation	Team FSA10	Team FSA11	Team FSA31	Team FSA23
Lines of Code	3899	7696	6564	1771
1) Rules compliance with Findbugs	1 (76,5%)	2 (89,3%)	2 (90,0%)	1 (82,4%)
2) Comments	0 (5,5%)	0 (5,4%)	0 (5,2%)	0 (16,4%)
3) Duplication	1 (5,1%)	1 (8,0%)	1 (5,9%)	2 (0%)
4) Method complexity	2 (2,6%)	2 (2,7%)	2 (1,8%)	2 (1,8%)
5) Package tangle index	2 (2,1%)	0 (45,9%)	1 (12,3%)	2 (6,3%)
6) Right implemented features	1 (20)	2 (25)	1 (21)	0 (14)
7) Design *	1	2	1	0
8) Time measurement **	0	2	2	1
9) JUnit tests	0 (0)	1 (3)	0 (0)	1 (5)
total points	8	12	10	9
nominated	0,444	0,667	0,556	0,500

* **FSA10:** (-) Textlabels and textfields cannot be read on profile add/update (-) no Android usability feature implemented (+) Application design goes hand in hand with the webpage (-) no menu navigation **FSA11:** (+) Uses Android specific functionalities such as a slider to navigate through the app (+) Clear and attractive design according to the webpage (+) Menu navigation **FSA31**: (+) Menu navigation (+) Clear and attractive design (-) no android usability feature implemented **FSA23**: (-) no android usability feature implemented (-) no menu navigation (-) Application design does not fit to the webpage

** **FSA10:** User-LogIn: 6,21 sec. & Profile-Update:5,46 sec. **FSA11:** User-LogIn: 1,92 sec. & Profile-Update: 2,32 sec. **FSA31:** User-LogIn: 1,65 sec. & Profile-Update: 1,20 sec. **FSA23:** User-LogIn: 4,41 sec. & Profile-Update: 2,46 sec.

The results of the software quality evaluation show that team FSA11 had performed best with twelve out of eighteen points. However, there was no very large distance in the results from FSA31, FSA23, FSA10, and the other three teams even performed better in some software quality metrics. It is notable that team FSA23 as well as team

FSA10 had numerous good software quality metrics but a poor performance, for example, in the application design or rules compliance, which might have been affected by the smaller amount of implemented features by these teams. In turn, team FSA11 and team FSA31 only had a fairly good performance on maintainability metrics such as the package tangle index or duplication. Furthermore, none of the four teams wrote a sufficient amount of comments on their code or sufficient test cases which might have been influenced by the small programming period. Summarizing the results of Table 7, team FSA11 and team FSA31 performed better in this model compared to the other teams because of their well-balanced development of good code quality and the fulfillment of the customer requirements.

Table 8 shows the results of the conducted experiment weighted with the results of the pre-survey as described in subchapter 5.1. For each team, the result of their software quality was multiplied with the weighting of their particular DEV group. Moreover, the result of the requirement quality of each team was multiplied with the weighting of their PO group.

Table 8: Weighted Software Quality and Requirement Quality results

TEAM-ID	Software Quality	weigh.	SQ weigh.	Requirements Quality	weigh.	RQ weigh.
FSA10	,444	1,173	,521	,625	,958	,599
FSA11	,667	,897	,598	,708	1,150	,814
FSA31	,556	1,017	,565	,250	,719	,180
FSA23	,500	,953	,477	,375	1,438	,539

As shown in Table 8, the software quality results of FSA10 moved closer to FSA11 and FSA31 after the weighting with the team member experience. Also, FSA11 moved closer to the other groups because of their negative weighting. But still, team FSA11 performed best on software quality followed by team FSA10 and FSA31, as it is shown in Table 7. Team FSA23 had poor performance in software quality compared to the other teams. On the other side, the requirement quality of the PO’s of FSA11 and

FSA23 increased through the positive weightings, whereas the requirement quality of team FSA31 and FSA10 decreased.

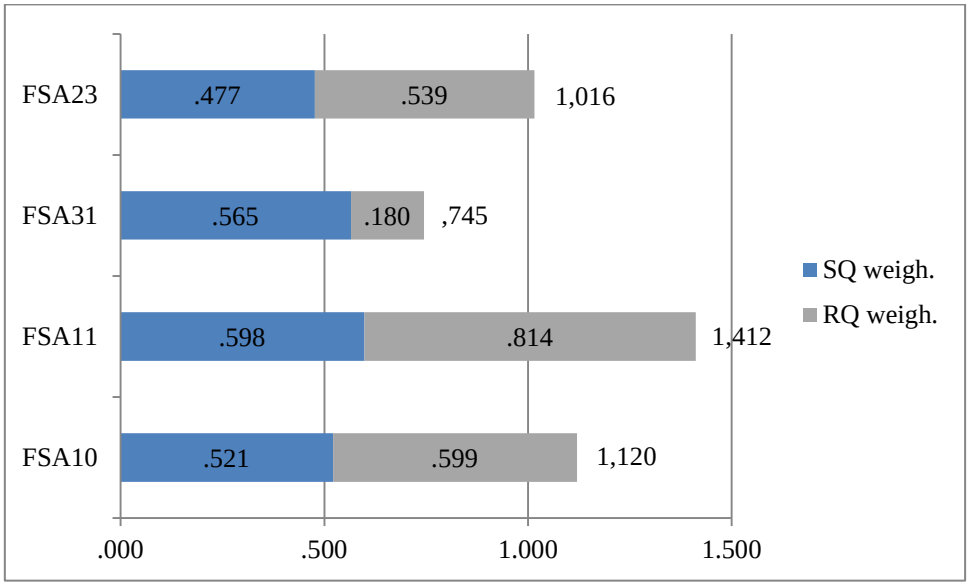


Figure 5: Total software and requirement quality

In general, the best software and requirement quality was delivered by team FSA11, followed by team FSA10. FSA31 delivered good software quality but poor requirement quality. Thereby, team FSA23 delivered good requirement quality but the lowest level of software quality compared to the other teams.

On the basis of the work result quality, H1 need to be rejected and H0 remains because no significant differences among the teams concerning correlations of their native language with the work result quality could be identified. However, on a descriptive level of statistics, the direct comparison of the average values shows that FSA11 had the best quality results of the work. Chapter six will discuss this outcome in detail.

5.3 H2a: Analysis of the level of trust survey

This survey was distributed in paper form to all team members after the conducted experiment. The survey was answered by 100% of the participants. Eight questions were evaluated using this analysis, which could be answered on a seven-point scale. The level of trust survey questions are attached as appendix 3.

Before analyzing the answers, a conducted correlation analysis showed that all questions are independent of each other. Hence, there was no need to conduct a factor analysis to group some of the questions. Due to that, an average value could be calculated for all responses of each individual team member. Therefore, the answers of question six and question seven had to be inverted due to a negative formulation of these questions. For example, if the question "There is no 'team spirit' in my group" was answered with 'strongly disagree", the answer was graded with five points instead of one point. Afterwards, a one-way Analyses of Variance (ANOVA) was applied to the calculated team member average values to determine significant differences among the teams. An ANOVA analysis can be used to test differences among at least three groups (Analyses among two groups can be covered by a t-test). The analysis shows that there are differences in the level of trust between the team FSA10 and the other teams with significance of $p < .004$ (see Appendix IV).

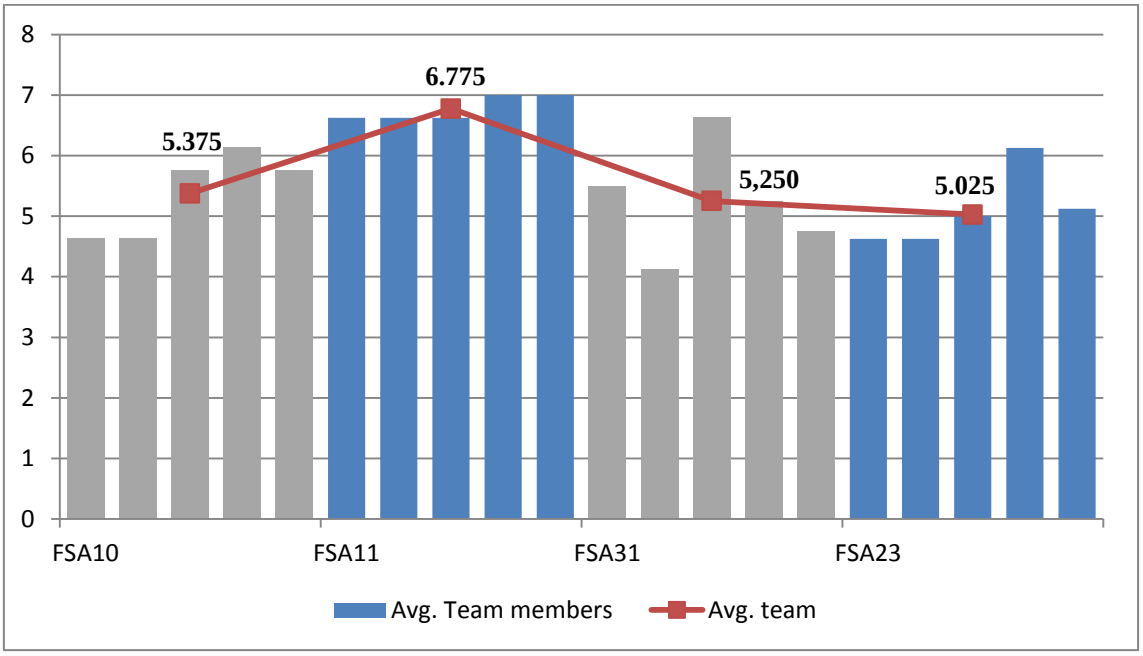


Figure 6: Show the average value of each team member as well as the average value of each team (ANOVA p > .004)

The result of the level of trust survey and the adopted ANOVA analysis is that the level of trust is significantly most pronounced in team FSA11, which is followed by team FSA11 and team FSA31. The lowest level of trust is among team FSA23. Therefore, H2a can be confirmed by several limitations as described in chapter six.

5.4 H2b: Analysis of the personal satisfaction of the team members

This survey about the personal satisfaction of the team members was conducted via internet to all team members. There was a total response of 100 percent. In total, ten questions were asked to all team members, whereas two questions had a negative formulation. These negative question results were inverted as described in the previous subchapter. The participants had to answer the questions on a five point scale. The survey is attached as appendix four. Similar to the analysis of the level of trust survey, a correlation analysis was conducted to show that all questions of this survey are independent to each other. Also, no further factor analyses had to be conducted. Afterwards, the average values for the responses of each team member were calculated, and furthermore, a one-way ANOVA was applied to determine possible differences in the personal satisfaction of the team members. However, the ANOVA analysis shows no significant $p > .183$ differences and no tendency on a better personal satisfaction of the teams with a common native language. Therefore, H2b needs to be rejected (see Appendix V).

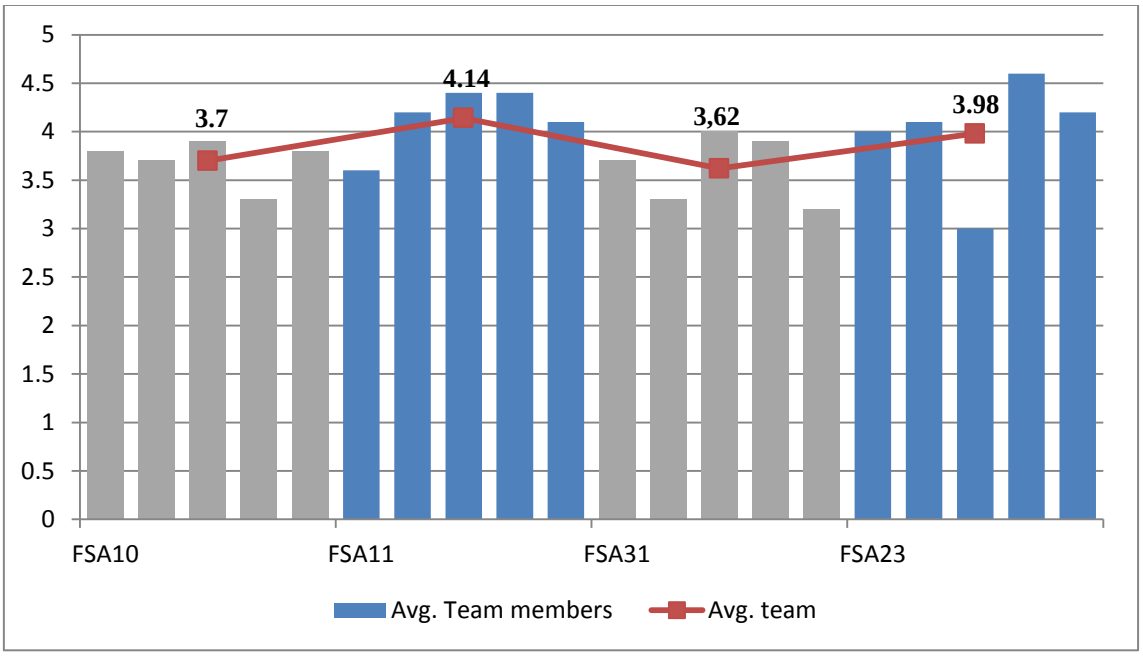


Figure 7: Show the average value of each team member as well as the average value of each team (ANOVA p.003)

5.5 Summary

On the radar chart below, all results of the conducted experiment as well as of the level of trust survey and the personal satisfaction survey are illustrated. Therefore, the results of the level of trust and personal satisfaction surveys were scaled from zero to one. This radar chart is not content of the following discussion because the results of each investigation are not comparable. Nevertheless, it shows a tendency which is conformed in the next chapter.

Clearly, team FSA11 performed best in the conducted experiment as well as in the level of trust survey. Team FSA23 and FSA10 had equal performances. Team FSA31 had similar performances compared to FSA23 and FSA10 and even performed slightly better in Software Quality, but the overall performances were negatively affected due to bad requirement quality.

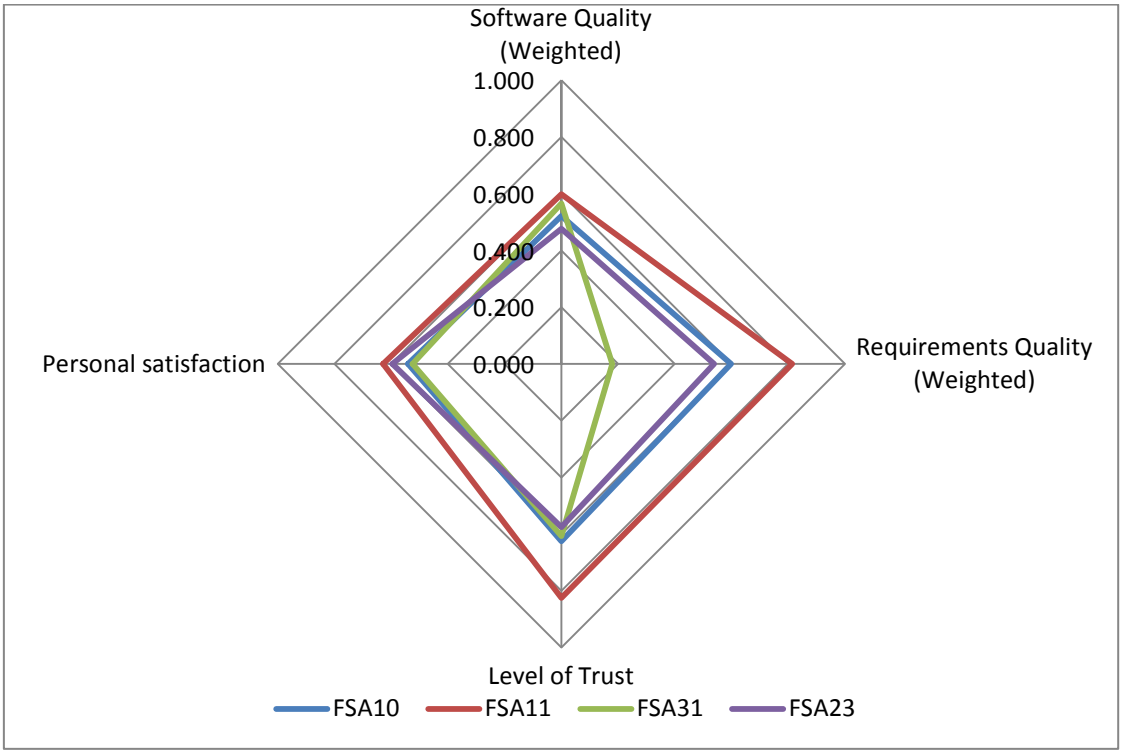


Figure 8: Result summary radar chart

Chapter 6

Discussion

The last chapter analyzed and described the results of the conducted experiment as well as of the surveys. Taking up this point, the following chapter discusses the findings from chapter five in order to resolve the hypothesis under observation. Additionally, some limitations of this empirical research are discussed. Overall, in addition to the discussion of the results, this chapter provides further information regarding a replication of the conducted experiment as well as improvements for future studies on this topic.

6.1 Limitations

First, there are some limitations on the study which need to be made before discussing the results. The experimental setup as a quasi-experiment has lower external validity due to the presetting of the teams instead of a randomized arrangement of each participant (Wohlin, et al., 2000). Randomized arrangement of the treatments (In this case the native or non-native language of the students) is a prerequisite for a controlled experimental setup (Wohlin, et al., 2000). Furthermore, the experiment was conducted with graduate students as the subjects under observation, which is discussed seriously in the literature concerning the external validity because this experimental design might

bias the reality (Schach, 1993, Höst, Regnell, & Wohlin, 2000). Further, the overall experimental model is too small due to the amount of test persons within the teams as well as the amount of teams in general. This amount was naturally limited by the resources of the AMOS course, which is not sufficient to calculate significant cause-effects among the teams by statistical analyses. Therefore, only descriptive methods are used to analyze the findings of the software and requirement quality. Another limitation is that all native speakers are from Germany, which is also naturally limited by the recourses available. This suggests that the results might have been influenced by other reasons; for example, the German education system. Finally, the results are restricted to the SCRUM agile development. Perhaps other agile or non-agile development models such as Extreme Programming or the waterfall development model would have led to different results

6.2 Discussion of the results

At the beginning of this discussion, it should be underlined that no general assumption can be drawn on external validity of the experimental results, which is a consequence of the limitations discussed one subchapter before. However, interesting tendencies are discussed in this subchapter that may be confirmed in further research building upon this investigation. On the basis of the findings in chapter five, this discussion reflects the rejection of H1 and H2b, as well as the internal validity of H2, because of the significant testing of the level of trust survey.

H1 was rejected for the reason that the small amount of data did not allow any statistical method that could confirm rational significant differences among the teams. Nevertheless, the descriptive analyses of H1 show a tendency of a better performance of the team FSA11 that could be derived from the experimental outcome with respect to the limitations. The results of the experiment show that this native language team had, in total (software and requirement quality), the best result of work quality (1,412) compared to the other teams. As shown in Figure 6 in chapter five, the team is followed by the heterogeneous team FSA10 (1,120) and the homogenous control group FSA23 (1,016), with team FSA31 following shortly after (,745). Interestingly, the native speaking DEV group in team FSA31 had the second-best software quality results (,565)

right after the DEV group of FSA11 (,598), whereas the PO group of team FSA31 had the lowest requirement quality result compared to the other PO groups. It can be asked on the basis of these findings, if the developers do speak a common native language in an agile SCRUM development process, and whether the languages barrier between PO and DEV group or within the PO group matters or not? Also, the heterogeneous team FSA10 shows this effect, vice versa. The native language PO group of team FSA10 performed second best (,599) right after the native language PO group of team FSA11 (,814). In contrast, the software quality of the non-native language DEV group of team FSA11 was not as good as the software quality of the native language DEV groups of team FSA11 and team FSA31. But either way, in turn it can be asked if low performance of the PO's in SCRUM projects does have an effect on the DEV group in general?

Over-all, the tendency is towards a better outcome of the native language groups. Maybe the reasons lie in the findings from chapter two, which describes the challenges of linguistic barriers; but this cannot be significantly confirmed by these findings. Due to that, this tendency should be viewed with caution as already mentioned in the limitation subchapter. Additional research needs to exclude a "German effect" by a replication of this study with native speaker groups from non-German speaking countries.

Referring to the software quality evaluation model in general, this tendency regarding the software quality can be discussed whether it reflects a reasonable examination on the software quality or not. As described in chapter three, there are several ways of measuring software quality in the literature. However, several previously conducted experiments on software quality only based their appraisals on the formula: $\text{Defect Rate} = \frac{\text{Errors}}{\text{KLOC}}$ (Green et al. 2010, Bird et al. 2009). Adapting this type of software quality measurement to the software quality outcome of this experiment, the results indicate a similar shape. This might confirm a positive correlation tendency of the native-speaking DEV groups with the work result outcome, as shown in Table 7. The values in Table 9 were calculated on the critical and minor errors identified by Sonar and the inverted weighting of software development experience by the conducted pre-survey.

Table 9: Software quality based on Defect Rate

	Team FSA10	**Team FSA11**	**Team FSA31**	**Team FSA23**
Lines of Code	3899	7696	6564	1771
Code size (KLOC)	3,899	7,696	6,564	1,771
Critical errors	19	23	39	16
Major Errors	224	176	126	68
Defect Rate	62,32	25,86	25,14	47,43
weigh.	0,827	1,103	0,983	1,047
Defect Rate (weighted)	51,537	28,519	24,718	49,654

Going along with the tendencies of the experimental findings, H2a was confirmed by the survey outcome with respect to the limitations of the study. The result shows that the level of trust within the pure native speaker team is significantly higher compared to the other teams. Interestingly, the survey outcome of the other teams has no high divergence among these teams. This result corresponds with the theoretical background of chapter two, which tends to the theory that a high level of trust among teams might support the software quality outcome. With regard to the personal satisfaction of the team members tested by H2b, it is interesting that all teams are strongly satisfied during the sprint review. Due to that, no significant differences among the teams could be calculated. The results might have differed if the experiment would have been executed for more than seven weeks or would have been tested in more than two different projects.

Adding to this discussion, the theory that the communication in one's native language correlates positively with work result quality in software development processes cannot be confirmed due to the fact that the major hypothesis, H1, had to be rejected. But as described in this subchapter, there is a tendency that the communication in one’s native language correlates positively with the work result outcome, which is supported by the outcome of the conducted trust survey. The next chapter describes some suggestions for further exploration on this particular research question.

Chapter 7

Conclusion

The theoretical background of this empirical research underlines that globally distributed software development has grown over recent decades, but challenges such as temporal, geographical, and social-cultural distances have still remained. Technical advances did support this growth and further technical innovations such as digital face-to-face meetings had reduced geographical distances. However, social-distance such as linguistic barriers cannot easily be bridged by a new technological innovation (at least not today). The cultural and educational background of the employees, previous experiences in virtual teams, and training are essential to reduce the social distance among teams. Hence, challenges such as linguistic barriers are still feasible.

This study investigated an experiment to determine the effects of linguistic barriers on the work result quality of two homogenous and two heterogeneous teams with dissimilar team member compositions regarding their native languages. The experiment was conducted in a controlled environment to prevent other influences such as geographical and temporal distances to determine the effects of native language on the work result quality of a software product. This investigation will be interesting for future compositions of software development teams as well as for minimizing the risk of low quality on future software products. The field of interests could vary from global

IT projects of organizations to global open source projects, on which international teams need to find and work together. Certainly, a significant confirmation of these results won't change and should not change existing and future team compositions, but it might help regarding the sensitivity of team members as well as the management.

Interestingly, the findings show a tendency that linguistic barriers might affect results of the work quality. This tendency is reasoned by the good performance of one homogenous team, which was composed of no linguistic barriers as well as the good performance of two subgroups of the heterogeneous teams, also without linguistic barriers. Although these results have some limitations, a replication of the experiment is necessary to confirm this tendency as well as the theory which was not confirmed during the discussion of the results.

7.1 Suggestion for future research

On the basis of the previous discussion and the limitations of this study, some suggestions for future work in this field of research are described in this subchapter. To confirm the tendency, this quasi-experiment needs to be replicated with native language speaking teams or groups from countries other than Germany. Additional research needs to exclude a "German effect" by a replication of this empirical research with native speaker groups from other countries. It is important that these team compositions are from at least two countries with similar educational systems and industry infrastructure as Germany. In addition, there should be another team from countries with a lower educational system and industry infrastructure. Of course, a control group with an international team composition is essential, as well.

In the case of similar findings as found in this empirical research, a true controlled experiment should be conducted. For example, randomized software development projects at the university with similar and dissimilar native language team compositions could be analyzed in a long-term study. The third step should be to conduct an experiment on this research question to gain significant results about professional developers in this field.

Regarding the measurement of the software quality, different measurement techniques as well as different development processes should be applied on further investigations. Besides the four teams under observation in this empirical research, there was another team developing an Android application in collaboration with a team in Beijing via face-to-face online communication. Therefore, in a fourth step, the linguistic barriers should be tested with virtual team composition instead of weekly face-to-face meetings.

Acknowledgment

I would like to thank Prof. Dr. Dirk Riehle for the thesis opportunity, Hannes Dohrn and Frank Denninger for their ongoing software engineering support.

Literature

Ågerfalk, P., & Fitzgerald, B. (2006). Flexible and distributed software processes: old petunias in new bowls? *Communications of the ACM 49(10)*, pp. 26-34.

Ågerfalk, P., Fitzgerald, B., Holmström, H., Lings, B., Lundell, B., & Ó Conchúir, E. (2005). A Framework for Considering Opportunities and Threats in Distributed Software Development, In. *In Proceedings of the International Workshop on Distributed Software Development (DiSD 2005), A* (pp. 47-61). Austrian Computer Society.

Basili, V. (7th. July 1996). The Role of Experimentation in Software Engineering: Past, Current, and Future. *8th international conference on Software engineering* (pp. 442 - 449). Washington, DC, USA: EEE Computer Society.

Basili, V. R., & Rombach, D. H. (June 1988). TAME Projecti Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering*, Vol. 14, No. 6.

Basili, V. R., Selby, R. W., & Hutchens, D. H. (July 1986). Experimentation in Software Engineering. *IEEE Interactions of Software Engineering*, No. 7, pp. 733-743.

Battin, R., Crocker, R., Kreidler, J., & Subramanian, K. (April 2001). Leveraging resources in global software development. *IEEE Software*, pp. 70-77.

Bird, C., Nachiappan, N., Premkumar, D., Gall, H., & Brendan, M. (August 2009). Does Distributed Development Affect Software Quality? An empirical case study of Windows Vista. *Communication of ACM*, Vol. 52, No. 8, pp. 85-93.

Bloomfield, L. (1995). *Language.* London: Motilal Banarsidass.

Carmel, E. (1999). *Global software teams: collaborating across borders and time zones.* Upper Saddle River, NJ, USA: Prentice Hall PTR.

Carmel, E., & Agarwal, R. (March/April 2001). Tactical Approaches for Alleviating Distance in Global Software Development. *IEEE SOFTWARE*, pp. 22-29.

Carver, J., Jaccheri, L., Morasca, S., & Shull, F. (2003). Issues in Using Students in Empirical Studies in Software Engineering Education. *Ninth International Software Metrics Symposium.* IEEE Computer Science.

Damian, D. (2002). Workshop on Global Software Development. *In Proceedings of International Conference on Software Engineering (ICSE).* Orlando, Florida, USA.

Damian, D., & Moitra, D. (September/October 2006). Global Software Development: How far we have come? *IEEE SOFTWARE*, pp. 17-19.

Easterbrook, S., Singer, J., Storey, M., & Damian, D. (2008). *Selecting Empirical Methods for Software Engineering Researc.* London: Springer London.

Ebert, C., & Neve, D. P. (March/April 2001). Surviving Global Software Development. *IEEE SOFTWARE*, pp. 62-69.

Favela, J., & Pena-Mora, F. (March/April 2001). An Experience in Collaborative Software Engineering Education. *IEEE Software*, pp. 47-53.

Free Seas Ahoy. (11. 08 2012). *Free Seas Ahoy!* Von Home: http://www.fsahoy.com/ abgerufen

Gartner Outsourcing & Strategic Partnerships . (2012). *Key Issues Facing ITO Industry.* Stamford.

Geay, C., McNally, S., & Telhaj, S. (March 2012). *Non-Native Speakers Of English In The Classroom: What Are The Effects On Pupil Performance?* London: Centre for the Economics of Education.

Green, R., Mazzuchi, T., & Sarkani, S. (2010). Communication and Quality in Distributed Agile Development: An Empirical Case Study. *World Academy of Science, Engineering and Technology*, pp. 61: 322-328.

Green, R., Mazzuchi, T., & Sarkani, S. (2010). Understanding the role of synchronous & asynchronous communication in agile software development and its effects on quality. *Journal of Information Technology Management*, S. Volume XXI, No. 2.

Herbsleb, J. D., & Moitra, D. (2001, March/April). Global Software Development. *IEEE Software*, pp. 16-20.

Herbsleb, J. D., Mockus, A., Finholt, T. A., & Grinter, R. E. (2001). An Empirical Study of Global Software Development: Distance and Speed. *23rd International Conference on Software Engineering* (S. 81 - 90). Washington, DC, USA: IEEE Computer Society.

Herbsleb, J., & Mockus, A. (June 2003). An empirical study of speed and communication in globally distributed software development . *IEEE Transactions on Software Engineering* , Vol. 29 , No. 6, pp. 481 - 494 .

Holmstrom, H., Ó Conchúir, E., Ågerfalk, P. J., & Fitzgerald, B. (2006). Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance. *IEEE International Conference on Globale Software Engineering.* IEEE Computer Society.

Höst, M., Regnell, B., & Wohlin, C. (2000). Using Students as Subjects - A Comparative Study of Students and Profession-als in Lead-Time Impact Assessment. *Empirical Software Engineering*, Vol. 5, pp. 201-214.

Leidner, D. E., & Jarvenpaa, S. L. (June 1998). Communication and Trust in Global Virtual Teams. *Journal of Computer-Mediated Communication*, Vol. 3, No 4.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, Vol. 22, No. 140.

Long, M. (1982). Native speaker/non-native speaker conversation and the negotiation of comprehensible input. *Applied Linguistics*, Vol. 4, No. 2, pp. 126 - 141.

Lutz, B. (2009). Linguistic Challenges in Global Software Development: Lessons Learned in an International SW Development Division. *Fourth IEEE International Conference on Global Software Engineering* (pp. 249-253). IEEE Computer Society.

Manifesto for Agile Software Development. (2001). *Agile Manifesto.* Von Principles behind the Agile Manifesto: http://agilemanifesto.org/principles.html abgerufen

McLeod, S. (2008). *Simply Psychology*. Von Likert Scale: http://www.simplypsychology.org/likert-scale.html abgerufen

Moe, N. B., & Smite, D. (2008). Understanding a lack of trust in gloabl software teams: a multiple-case study. *Softw. Process Improve. Pract.*, No. 13, pp. 217-231.

Muhammad, A. B., June, M. V., & Phong, T. N. (2007). Establishing and maintaining trust in software outsourcing relationships: An empirical investigation. *The Journal of Systems and Software* , Vol. 80, pp. 1438–1449.

Noll, J., Beecham, S., & Richardson, I. (September 2010). Global software development and collaboration: barriers and sollutions. *ACM Inroads*, Vol. 1, No. 3, pp. 66-78.

Oza, N., Hall, T., Rainer, A., & Grey, S. (2005). Trust in software outsourcing relationships: an empirical investigation of Indian software companies. *9th International Conference on Empirical Assessment in Software Engineering.* Keele, UK.

Pearce, J. L., Sommer, S. M., Morris, A., & Frideger, M. (1992). *A configurational approach to interpersonal relations: Profiles of workplace social relations and task interdependence.* Irvine: Graduate School of Management, University of California.

Powell, A., Piccoli, G., & Ives, B. (2004). Virtual teams: A review of current literature and direction for future research. *The DATA BASE for Advances in Information Systems,* Vol. 35, No. 1, pp. 6-36.

Prikladnicki, R., Marczak, S., & Audy, J. L. (2006). MuNDDoS: A Research Group on Global Software Development. *International Conference on Global Software Engineering* (S. 2). IEEE Computer Science.

Riehle, P. D. (10. 08 2012). *Software Research and the Industry*. Von Courses: http://dirkriehle.com/courses/agile-methods/ abgerufen

Rogelberg, S., Allen, J., Shanock, L., Scott, C., & Shuffler, M. (March–April 2010). Employee satisfaction with meetings: A contemporary facet of job satisfaction. *Wiley InterScience,* No. 2, pp. 149–172.

Runeson, P. (2003). Using Students as Experiment Subjects - An Analysis on Graduate and Freshmen PSP Student Data. *7th International Conference on Empirical Assessment & Evaluation in Software Engineering* , (pp. 95 - 102).

Sahay, S. (2003). Global software alliances: the challenge of 'standardization'. *Scandinavian Journal of Information Systems*, pp. 15: 3-21.

Sarker, S., & Sahay, s. (2004). Implications of space and time for distributed work: an interpretive study of US-Norwegian systems development teams. *European Journal of Information Systems*, Vol. 13, pp. 3-20.

Schach, S. R. (1993). *Software Engineering - Second Edition.* Chicago: Irwin Professional Publishing.

Smite, D., & Borzovs, J. (2006). *A framework for overcoming supplier related threats in global projects.* Joensuu, Finland: Springer Verlag.

Sonar. (12. 08 2012). *SonarSource*. Von Home: http://www.sonarsource.org/ abgerufen

Spillner, A., & Linz, T. (2005). *Basiswissen Softwaretest.* Heidelberg: dpunkt.verlag GmbH.

The Apache Software Foundation. (19. 08 2012). *Apache Maven Project*. Von Home: http://maven.apache.org/ abgerufen

The Scrum Alliance, T. (08. 08 2012). *Scrum Alliance*. Von Scrum 101: Scrum Alliance: http://www.scrumalliance.org/pages/scrum_101 abgerufen

Tichy, W. F., Lukowicz, P., Prechelt, L., & Heinz, E. A. (1995). Experimental evaluation in computer science: a quantitative study. *J. Syst. Softw.*, pp. 9–18,.

Wageman, R., Hackman, R., & Lehman, E. (2005). Team Diagnostic Survey-Development of an Instrument. *The Journal of Applied Behavioral Science*, S. Vol. 41, pp. 373.

Walenta, T. (April 2004). Managing cross-cultural issues in global software engineering. *Communication of the ACM*, S. Vol. 47, No 4, pp. 62-66.

Web Center for Social Research Methods. (11. 08 2006). *Social Research Methods*. Von Likert Scale: http://www.socialresearchmethods.net/kb/scallik.php abgerufen

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2000). *Experimentation in Software Engineering.* Boston: Kluwer Academic Publishers.

Versicherung

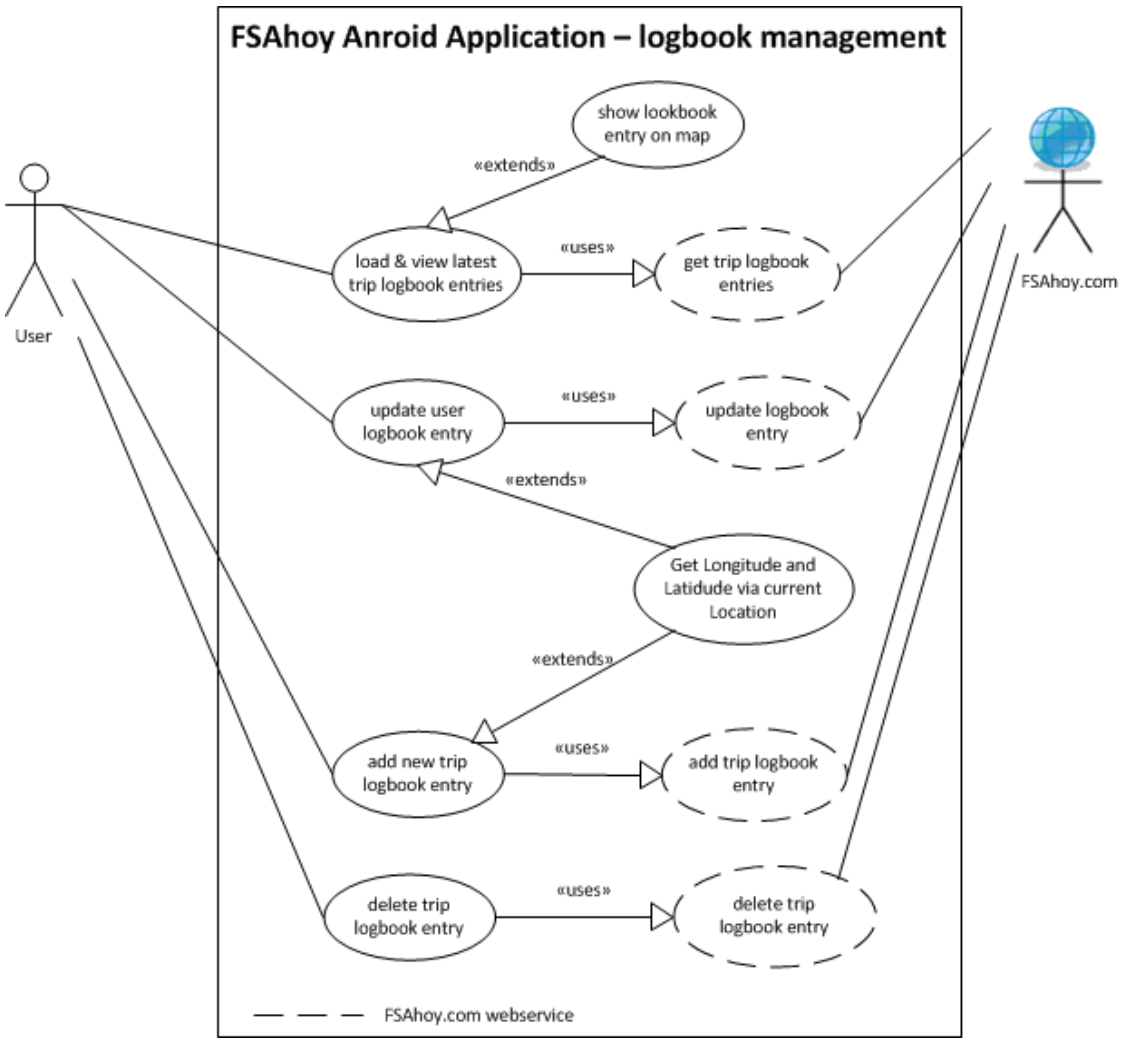
Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat. Alle Ausführungen der Arbeit, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Nürnberg, den ....

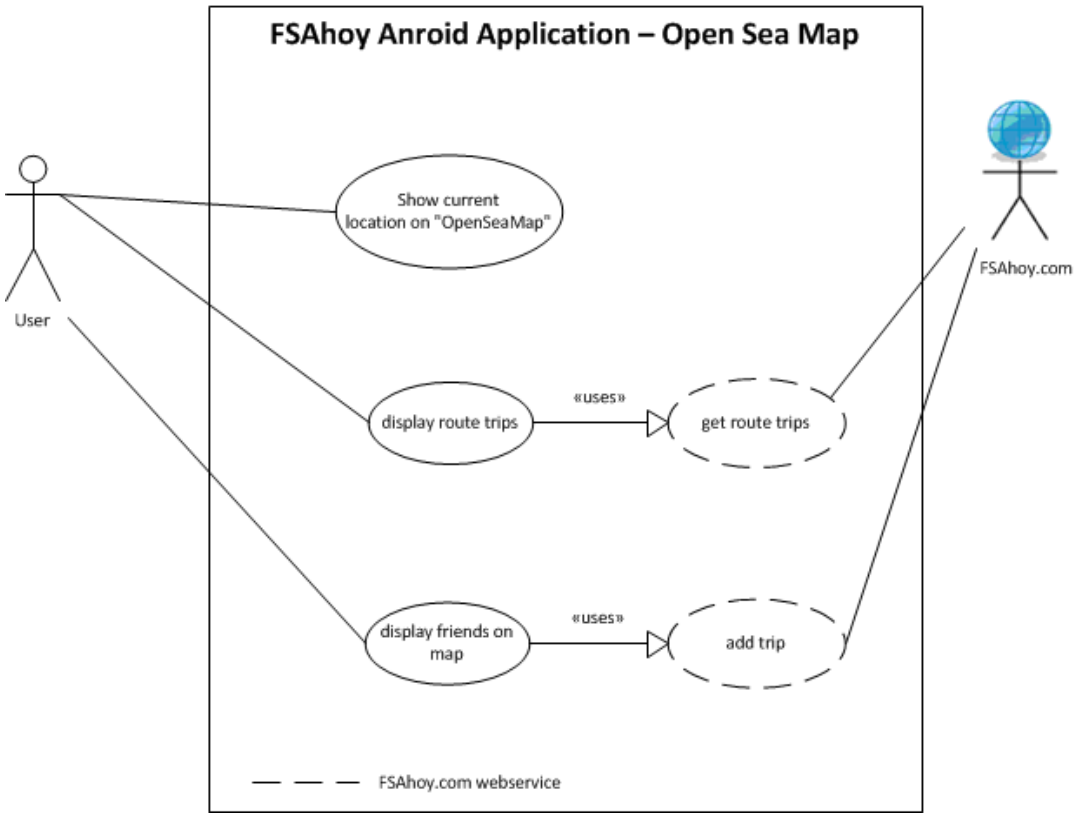
(Unterschrift)

Appendix I – Use Cases

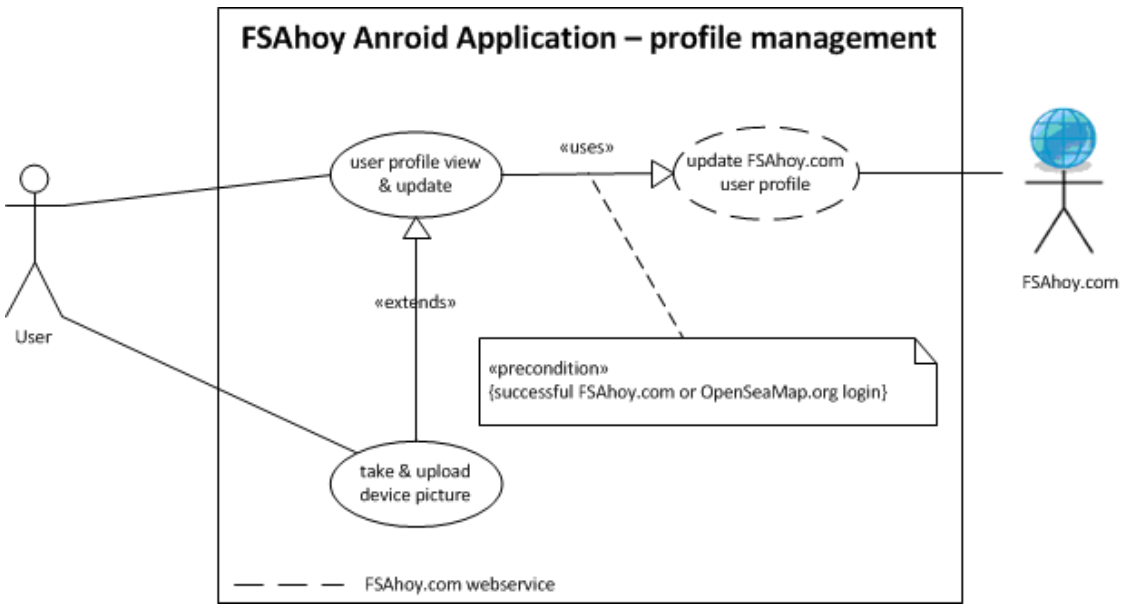
A)



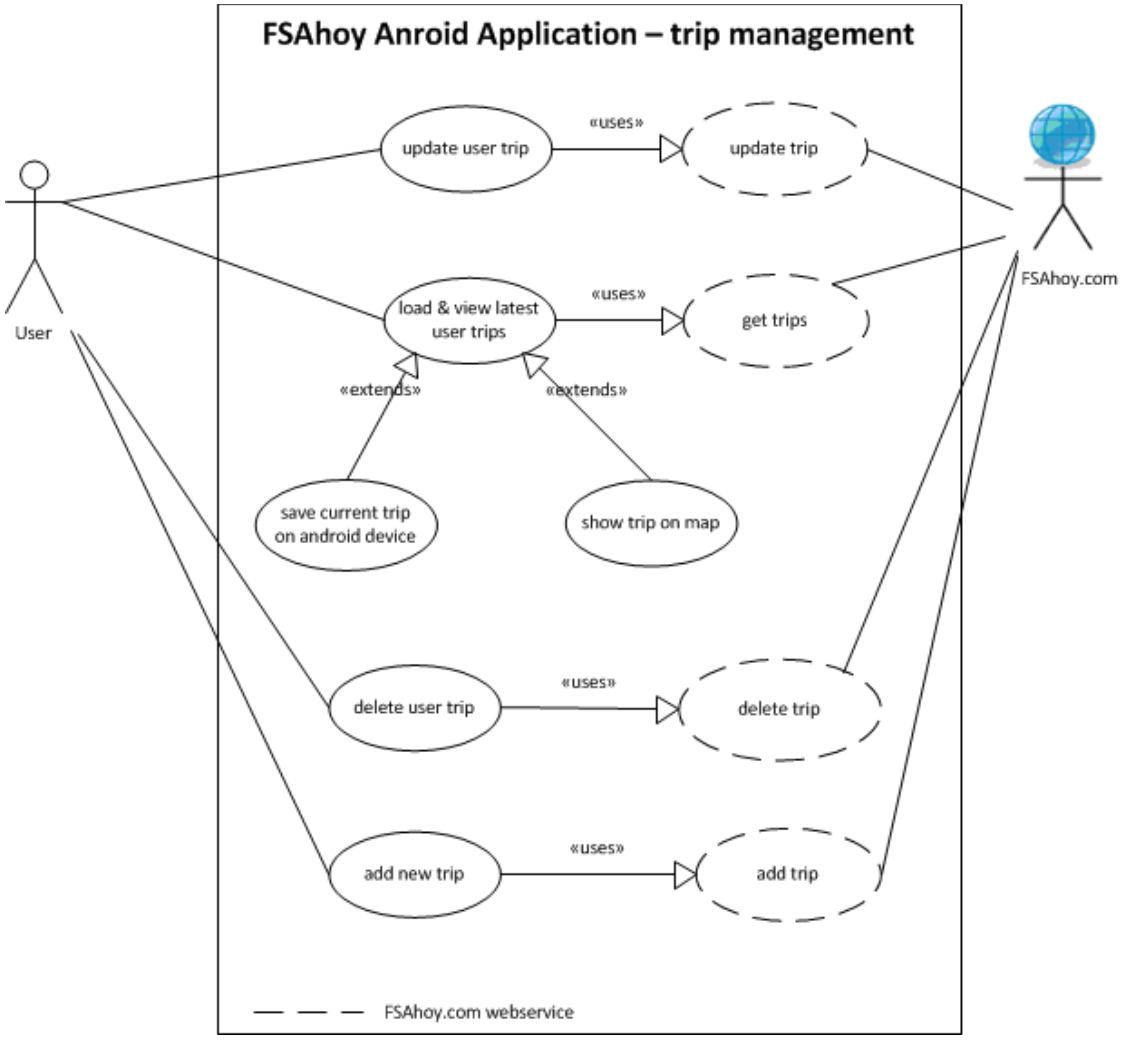
B)


FSAhoy Anroid Application – Open Sea Map
User
Show current location on "OpenSeaMap"
display route trips
«uses»
get route trips
display friends on map
«uses»
add trip
FSAhoy.com
FSAhoy.com webservice

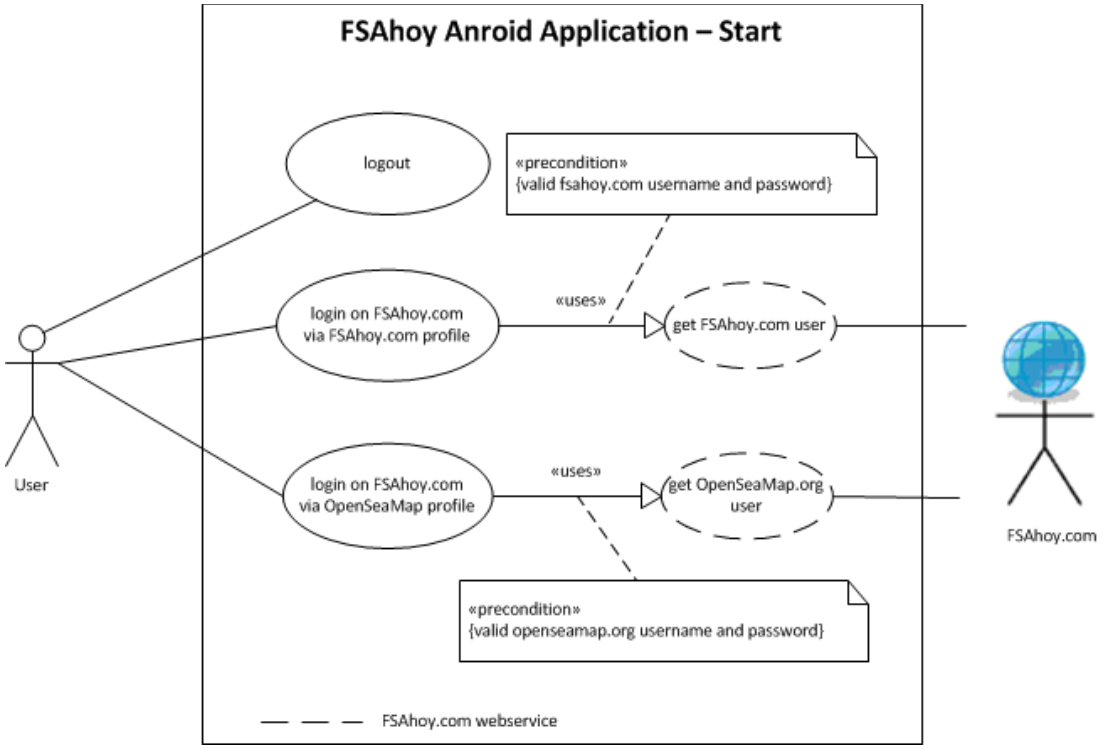
C)


FSAhoy Anroid Application – profile management
User
user profile view & update
«uses»
update FSAhoy.com user profile
«extends»
«precondition»
{successful FSAhoy.com or OpenSeaMap.org login}
take & upload device picture
FSAhoy.com
FSAhoy.com webservice

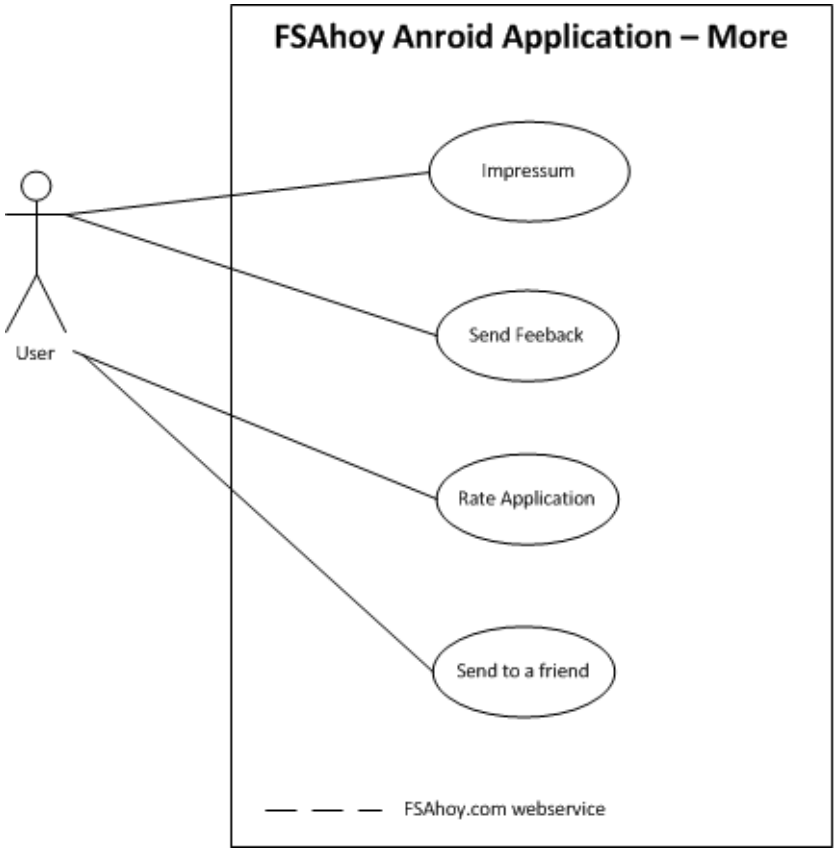
D)


FSAhoy Anroid Application – trip management
update user trip
«uses»
update trip
load & view latest user trips
«uses»
get trips
«extends»
«extends»
save current trip on android device
show trip on map
delete user trip
«uses»
delete trip
add new trip
«uses»
add trip
FSAhoy.com webservice
User
FSAhoy.com

E)


FSAhoy Anroid Application – Start
logout
«precondition»
{valid fsahoy.com username and password}
login on FSAhoy.com
via FSAhoy.com profile
«uses»
get FSAhoy.com user
User
login on FSAhoy.com
via OpenSeaMap profile
«uses»
get OpenSeaMap.org
user
FSAhoy.com
«precondition»
{valid openseamap.org username and password}
FSAhoy.com webservice

F)


FSAhoy Anroid Application – More
Impressum
User
Send Feeback
Rate Application
Send to a friend
FSAhoy.com webservice

Appendix II – Personal Satisfaction Survey Questions

1. I feel a real sense of personal satisfaction when our team does well
2. I feel bad and unhappy when our team has performed poorly
3. My own feelings are not affected one way or the other
   by how well our team performs
4. When our team has done well, I have done well.
5. I learn a great deal from my work on this team
6. My own creativity and initiative are suppressed by this team
7. Working on this team stretches my personal knowledge and skills
8. I enjoy the kind of work we do in this team
9. Working on this team is an exercise in frustration
10. Generally speaking, I am very satisfied with this team

Appendix III – Level of Trust Survey Questions

1. Members of my work group show a great deal of integrity
2. I can rely on those with whom I work in this group
3. Overall, the people in my group are very trustworthy
4. We are usually considerate of one another’s feelings in this work group
5. The people in my group are friendly
6. There is no "team spirit" in my group
7. There is a noticeable lack of confidence among those with whom I work
8. We have confidence in one another in this group

Appendix IV – ANOVA Personal Satisfaction

ONEWAY deskriptive Statistiken

AVERAGE

	N	Mittelwert	Standardabweichung	Standardfehler	95%-Konfidenzintervall für den Mittelwert		Minimum	Maximum
					Untergrenze	Obergrenze		
FSA10	5	3,7000	,23452	,10488	3,4088	3,9912	3,30	3,90
FSA11	5	4,1400	,32863	,14697	3,7319	4,5481	3,60	4,40
FSA31	5	3,6200	,35637	,15937	3,1775	4,0625	3,20	4,00
FSA23	5	3,9800	,59330	,26533	3,2433	4,7167	3,00	4,60
Gesamt	20	3,8600	,42600	,09526	3,6606	4,0594	3,00	4,60

Test der Homogenität der Varianzen

AVERAGE

Levene-Statistik	df1	df2	Signifikanz
,805	3	16	,509

ONEWAY ANOVA

AVERAGE

	Quadratsumme	df	Mittel der Quadrate	F	Signifikanz
Zwischen den Gruppen	,880	3	,293	1,828	,183
Innerhalb der Gruppen	2,568	16	,161		
Gesamt	3,448	19			

Appendix V – ANOVA Level of Trust

ONEWAY deskriptive Statistiken

AVERAGE

	N	Mittelwert	Standardabweichung	Standardfehler	95%-Konfidenzintervall für den Mittelwert		Minimum	Maximum
					Untergrenze	Obergrenze		
FSA10	5	5,3750	,70156	,31375	4,5039	6,2461	4,63	6,13
FSA11	5	6,7750	,20540	,09186	6,5200	7,0300	6,63	7,00
FSA32	5	5,2500	,93123	,41646	4,0937	6,4063	4,13	6,63
FSA23	5	5,1000	,61492	,27500	4,3365	5,8635	4,63	6,13
Gesamt	20	5,6250	,92124	,20600	5,1938	6,0562	4,13	7,00

Test der Homogenität der Varianzen

AVERAGE

Levene-Statistik	df1	df2	Signifikanz
1,673	3	16	,213

ONEWAY ANOVA

AVERAGE

	Quadratsumme	df	Mittel der Quadrate	F	Signifikanz
Zwischen den Gruppen	9,006	3	3,002	6,747	,004
Innerhalb der Gruppen	7,119	16	,445		
Gesamt	16,125	19			

Post-Hoc-Tests

Mehrfachvergleiche

AVERAGE
Tukey-HSD

(I) TEAM	(J) TEAM	Mittlere Differenz (I-J)	Standardfehler	Signifikanz	95%-Konfidenzintervall	
					Untergrenze	Obergrenze
FSA10	FSA11	-1,40000*	,42186	,020	-2,6070	-,1930
	FSA32	,12500	,42186	,991	-1,0820	1,3320
	FSA23	,27500	,42186	,913	-,9320	1,4820
FSA11	FSA10	1,40000*	,42186	,020	,1930	2,6070
	FSA32	1,52500*	,42186	,011	,3180	2,7320
	FSA23	1,67500*	,42186	,005	,4680	2,8820
FSA32	FSA10	-,12500	,42186	,991	-1,3320	1,0820
	FSA11	-1,52500*	,42186	,011	-2,7320	-,3180
	FSA23	,15000	,42186	,984	-1,0570	1,3570
FSA23	FSA10	-,27500	,42186	,913	-1,4820	,9320
	FSA11	-1,67500*	,42186	,005	-2,8820	-,4680
	FSA32	-,15000	,42186	,984	-1,3570	1,0570

*. Die Differenz der Mittelwerte ist auf dem Niveau 0.05 signifikant.