Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

FAKULTÄT FÜR !NFORMATIK

A controlled experiment on team meeting effectiveness in software architecture evaluation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Christoph Seemann
Matrikelnummer 9826968

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Prof. Dr. Stefan Biffl
Mitwirkung: Dipl.-Ing. Dietmar Winkler

Wien, 28.09.2009 _____ _____

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Christoph Seemann

1170 Wien, Leopold-Ernst-Gasse 34-36 / 41

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien am 28.09.2009

Kurzfassung

Die Architektur eines Softwareprodukts ist ein sehr kritischer Punkt in der Software Entwicklung, weil die Qualität der Architektur sehr eng mit der Qualität des fertigen Produktes zusammenhängt. Aus diesem Grund ist es sehr wichtig sicherzustellen, dass die Architektur sehr sorgfältig definiert und überprüft wird – und das in frühen Stadien des Software-Entwicklungsprozesses. Der Aufwand und die Kosten steigen, je später Fehler bemerkt werden oder Änderungen auftreten.

Architektur-Reviews unterstützen Entwickler dabei die Qualität der Architektur in frühen Entwicklungsphasen sicherzustellen und diese genau zu evaluieren. Szenarien sind bekannte Ansätze um einen guten Überblick zu bekommen, was von einer Software-Architektur verlangt wird und wie diese aussehen sollte, um den verlangten Qualitätsmerkmalen (auch nicht-funktionale Anforderungen genannt) des Produkts zu entsprechen. Solche Merkmale können zukünftige Anforderungen, wie z.B. in den Bereichen der Modifizierbarkeit, Wartbarkeit, etc., aufzeigen. Solche möglichen Änderungen sollten bereits früh in der Software-Entwicklung bedacht werden.

Szenarien können durch Brainstorming-Sessions einzelner Personen aber auch in Teams identifiziert werden. Durch Team-Meetings können bessere und wertvollere Szenarios durch Diskussion und Interaktion der Teilnehmer entdeckt werden. Richtlinien können den Teilnehmern dabei helfen sich auf eine spezielle Szenario-Kategorie zu konzentrieren. Es ist sehr wichtig die bestmögliche Methode für das Erstellen solcher Szenarien zu finden, da die Ergebnisse solcher Evaluierungen mit der Qualität des Endprodukts zusammenhängen.

Empirische Studien helfen herauszufinden, welche Methoden am besten geeignet sind, indem reale Bedingungen von Architektur-Evaluierungen in einem kontrollierten Experiment nachgestellt werden. In unserer Untersuchung wurden 54 Studenten, mit unterschiedlicher Qualifikation, gebeten Architektur-Szenarien für zwei unterschiedliche Systeme zu finden, welche jeweils unterschiedliche Aufgabengebiete und Funktionen hatten. Durch die Evaluierung zweier Applikationen haben wir einen besseren Überblick bekommen und eine bessere Vorstellung davon welche Anforderungen verlangt werden. Zu Beginn mussten die Studenten Szenarien alleine finden und anschließend daran in einer Gruppe von drei Leuten. Während dieser Gruppenphase gab es zwei Möglichkeiten wie die Studenten miteinander kommunizieren konnten: Die eine Hälfte der Teams kommunizierte und diskutierte die Szenarien von Angesicht zu Angesicht, während die andere Hälfte der Teams über das Internet mit einer textbasierten Chat-Applikation kommunizierte, ohne physisch beieinander zu sitzen. Um zu untersuchen, welche gefundenen Szenarien „die besten“ sind, wurde ein Referenz-Profil erstellt, welches auf allen gefundenen Szenarien basierte.

In dieser Arbeit wollten wir die beste Möglichkeit identifizieren, wie Szenarien für Architektur-Reviews in Teams gefunden werden können. Aber welche Art von Team-Meeting wäre optimal – physische, reale Meetings oder Meetings mit elektronischen Hilfsmitteln, wie z.B. über das Internet mit einer Chat-Applikation? Das ist eine sehr wichtige Frage, da reale Meetings viel teurer sind, weil alle Teilnehmer physisch am selben Ort zusammenkommen müssen. Wäre die Qualität der Szenarien beider Team-Meeting Möglichkeiten gleich, würde dies reale Meetings obsolet machen.

Wir haben herausgefunden, dass Teams, die sich real getroffen haben, mehrere wichtige Szenarien gefunden haben, als Teams die über eine textbasierte Chat-Applikation kommuniziert haben. Wir haben auch herausgefunden, dass real-treffende Teams im Verhältnis mehr wichtige Szenarien in ihren Szenario-Profilen, im Vergleich zu ihrer individuellen Phase, hatten als Teams, die über das Internet kommunizierten. Ein Zusammenhang zwischen der Erfahrung der Teilnehmer und der Anzahl an gefunden, wichtigen Szenarien konnte nicht festgestellt werden.

Diese Ergebnisse können Entscheidungsträgern dabei helfen, welche Team-Meeting Methode bei der Evaluierung von Software Architekturen die effizienteste ist, um eine möglichst hohe Qualität des Softwareprodukts zu gewährleisten.

Abstract

The architecture of a software product is a success critical issue in software development, because the quality of the architecture coheres very strongly with the quality of the final product. Thus, it is very important to ensure that the architecture is set up and reviewed very carefully early in the development life cycle of a software product. Rework effort and cost increases, the later defects are identified or changes occur.

Architecture reviews support engineers to ensure the quality of the selected architecture early in the software development process and help engineers in efficiently evaluating the underlying architecture of a software product. Scenarios are well-known approaches to get a well-defined focus on what is needed and how the architecture of a software product should look like to meet the quality (or non-functional) requirements of the product. Quality requirements can address possible upcoming needs, e.g., modifiability and maintainability, during development and maintenance. These changes should be addressed early in the development life-cycle.

Scenarios can be developed through individual brainstorming sessions or in architecture evaluation team meetings to identify better and more valuable scenarios due discussion and interaction. Brainstorming guidelines can help reviewers to focus on a defined scenario category. Thus, the results of such brainstorming sessions cohere with the quality of the final product, so it is very important to find the most convincing method of creating these scenarios.

Empirical studies help to find those most convincing methods by rebuilding real circumstances of architecture evaluation in a controlled experiment. In our research 54 students, who had different levels of qualification, were asked to identify architecture scenarios for two different software systems – each with different focuses and functions. Evaluating two applications gives us a better and more general view on what requirements are requested. First the students had to find some scenarios individually and then in a group of three people. While in a group there were two different ways, how the students could communicate with each other. One half of the teams was communicating and discussing the scenarios face to face and the other half was communicating over the internet with a text-based chat application. To tell which scenarios are “the best”, a reference-scenario profile of all found scenarios was created.

In this work we wanted to investigate the most effective way for creating architecture scenarios with the team processes for scenario identification. But what kind of team meeting would be optimal - Face to face or tool-supported communication? This is a very important question - doing face to face meetings is much more expensive than doing tool-supported communication, because all participants have to come together physically – maybe even coming from different continents. Comparable quality levels of architecture scenarios from both meeting styles would make face to face meetings obsolete.

We found out that groups, that were communicating face to face, have found more important scenarios than groups communicating with the text-based chat-application. We also saw that face to face teams had more important scenarios in their scenario-profiles compared to the individual phase, than tool-support teams. We also had a look at the relationship between experience and the number of important scenarios found, but we could not see a connection between the meeting style of the teams and their experience as we measured it.

These results can help decision makers and stakeholders to choose, which team-meeting method of evaluating software architecture scenarios will be more efficient to ensure a high quality of the software product. Another interesting aspect of analyzing which way of communication is more effective is, that it can also be learned which method might be more effective for evaluation, meetings, etc. in general.

Table of Contents

List of Figures

List of Tables

1 Introduction

Project Managers who are working in the IT-sector have the task to ensure that their projects are finished in time and that the quality is high. This task fits to every Project Manager, not only in IT business. In other businesses it is quite easy to say, if a product and its quality is good. An ingot made of steel for example is not good, when it is not straight but crooked or it cannot carry the weight it should. The process of manufacture has grown and developed further and further over the centuries. Today it is no problem producing good ingots of steal. There are methods of doing this and also methods of testing its quality.

In IT this is different. Computers are not present in our society for a few hundred years but only for a few decades. That means, that there are not so many well working methods of quality management or production. Such methods are still coming and going, some are developed further, some are not.

For a Project Manager it is a very difficult process to choose, how IT projects should be handled and realized. They have to choose which model they want to use to bring the project to a good end. How can be said, if an IT project is good or successful in the end? When it is finished in time and the customer is satisfied. To keep a project in time it is very important to do the planning right. Everything that is not planned or defined well in the beginning will become a problem in later project phases and can lead to a longer continuance of the project - maybe too long for the customer.

There are many ways to lead a software project to a good end. The key of doing this is to do good planning in the beginning of a project. If the first phases of a project are done well, fewer surprises should arise in later phases of the project. One major part is to develop a good software architecture which coheres with the needs and the functionality of the final product – as size and complexity of software increases, the design of software architecture gets more difficult [54, 66]. Good software architecture is not built easily. There are many software projects that do not have a good architecture or no defined architecture at all and fail because of this.

In this thesis we will have a look on how good software architecture can be build and what can be done to evaluate it. There are many ways of doing this. We will focus on scenario software architecture evaluation, which will be explained in detail later.

First it is good to know what it is all about. To understand the need of a fundamental architecture in software projects you have to understand what software engineering is in general and how software is built. In this chapter we will have a look at software engineering and an overview of modern software processes. After you have a feeling of how large and complex software projects are handled nowadays you can understand what software architecture is all about and why it is important to spend enough time building it in detail.

1.1 Software Engineering

Software is all around us today. It is not only running on computers where we notice it directly. It is included in nearly every electronic part or device. Some programs are large and complex and some are small and easy. There is software running in traffic lights, elevators, but also in huge factories, where machines and robots are working.

But all these different types of software have a different claim on quality, costs for production, life-time, etc. If you look at software that is running in a cheap digital alarm-clock, the focus won't be on quality at first place. It will be on short development time so the production of the software is as cheap as possible. If you look at elevators, traffic lights or atomic power plants on the other hand the focus will be (or should be!) on quality. The software has to work perfectly and must not have any bugs, because human life relies on the functionality of software.

So there are different types of software which have different focuses during development. Software Engineering [62] is used to produce software with good quality as cost effective as possible. The larger the software gets, the more complex it is. A small program, like the one of the alarm-clock mentioned before, might not need that much quality management or management at all. But if you look at larger and more complex software like in an atomic power plant it is very important to do good software engineering. In such huge programs million lines of code are written, thousands of objects, methods, interfaces, etc. work together, exchange data and so on. All these interactions and procedures in these programs have to work right. Good software engineering means, that there is an overview, what which part does and how it communicates with other parts. There is a schedule that defines when which part and when the whole project has to be finished. Software has to be tested, revised, recoded and in the end work as it should. Not only human life is a big factor why software should work perfectly – also money. If you look at space-travel for example, not only the quality factor is important, but also the cost factor. Sending satellites to other planets costs a lot of money, so it is evident that the production cost of software has to be as low as possible. Sometimes quality management is not executed well enough, like it happened in 1999 where the 'Mars Climate Orbiter' crashed because of a software bug: Some scientist were using the metric system, some the U.S. system in their programs.

Software engineering was not there when the first computers and programs appeared. In the beginning programs were not on time and have been completed months later. Others were way over budget. So methods were created to help develop software with planning. But the more complex and bigger programs became, the more complex and better techniques had to be found. So software engineering grew with the size and complexity of software itself.

But even nowadays, where good software engineering techniques are available, not all software produced is using them, or is not using them correctly. There is as well no 'right' way to develop software. It depends on what you want to produce – how complex it is, if cost and / or quality are more important than other factors, if you develop the program alone or in a team of a few hundred people. It depends on what you want to create.

According to Sommerville software engineering is an engineering discipline that is concerned with all aspects of software production from early stages of system specification to maintaining the system

after it has gone into use, which means that it includes things like programming, but also project and quality management, testing, etc. [1]

To ensure good software-quality a thought trough software process is necessary. A software process in general is divided in various phases. It is a rough plan for developing software which helps managing software projects. Using a software process as a base, more detailed project plans, including milestones for example, can be created and more detailed processes like the V-Model XT or the Rational Unified Process (RUP) can be used.

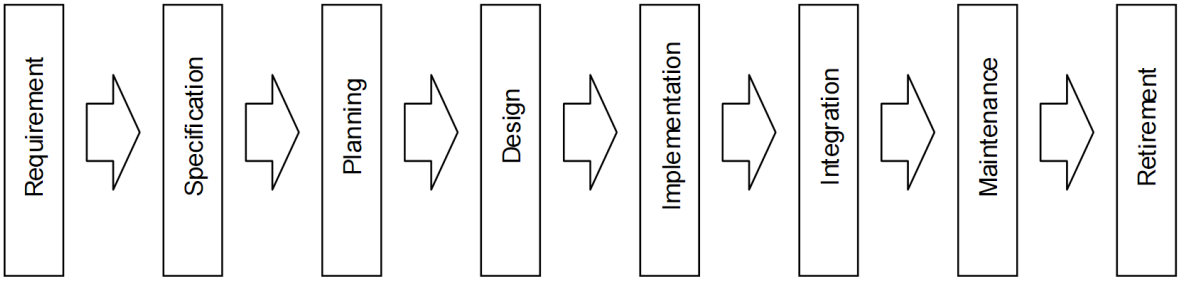


Figure 1-1 Software Life-Cycle [2]

Schach describes eight phases of a software process which will be described here shortly [2]:

- *Requirement*: The customer and the developer should define very clearly what the requirements of the software should be. The customer has to make clear what he expects from the final product and the developer defines what he is going to develop. All these things have to be written down to avoid differences of opinion in later phases of the project.

- *Specification*: Detailed functionality of the software product is written down. It is also defined, how documentation should be handled and so on. It is important to define the specification very clearly to avoid ambiguity which could lead to problems between customer and developer when the project has to come to an end.

- *Planning*: The project management has to do the planning for the entire project minding the length, the cost and resources of the project.

- *Design*: All parts of the software product are defined in detail. Inputs and outputs between components, the components itself, interfaces, etc. in other words the whole architecture of the software product.

- *Implementation*: All things that have been defined in the design phase are programmed and tested.

- *Integration*: All parts are put together. All modules that have been programmed and tested are now connected to one big (hopefully) working system. In this stage design flaws will become visible. That is why good planning and a good architecture are very important. If basic failures coming from the design are discovered now, it becomes very expensive to fix them.

- *Maintenance*: The software is finished and running. Bugs that are discovered during operation are fixed and also the architecture is expanded and modified as well, if the system is going to be adapted or extended.

- *Retirement*: At some point the software has to reach the end of its life-cycle, because it may be better to retire the perhaps many years old software and to produce a new one.

Many projects do not spend much time for the first two or three phases, which may result in huge problems later on. The Requirement-, Specification- and Planning-Phases of software projects are very important. When building software architecture you have to take time and do the planning very carefully. Also evaluating the architecture has to take place in the beginning phases. So you have the chance to find problems or design mistakes in the basic fundament of your software.

There are a lot of different software process models which do have this software process as a base - some more, some less. We will have a look at one, often used, software process to see, how software engineering is done.

1.1.1 Software Architecture

Software architectures a very important, if not the most important, factor of a software product. This chapter will give you a short description of the nature of architecture in software engineering and why it is so important. We will go into detail in the next chapters.

Software architecture is often not done deficient. It is more than just a short textual description and some diagrams. Kruchten describes a few fundamental properties of software architectures [5]:

- The organization of a software system

- The selection of structural elements and their interfaces of which the system is composed of, together with their behavior, as specified in the collaboration among those elements

- The composition of these elements into progressively larger subsystems

- The architectural style that guides this organization, these elements and their interfaces, their collaborations and their composition

As it can be seen, if the architecture of a software product is well-conceived it is a good basement for the product and not only a brief sketch. Just having a bad Quality-Management or no Quality-Management at all, but also the lack of a good software architecture can lead a software project to fail and does many times in real business-life. Software architecture is often not described very well and does not fulfill the wishes of all stakeholders [28], but only of some of them, which could lead to a software product that does not include all non-functional requirements (also called quality attributes) it should [58].

It is very important to define a method, how the architecture should be described, to have the possibility to review it and work on it in a team. Otherwise the development of the software architecture will be very difficult and may not be very effective. In the next chapter we will have a

look at software architecture in detail, at different views you can have on it and how it can be reviewed.

Software Architecture is not done quickly as one part of many in one phase of the software process. Furthermore it grows and grows continuously through the whole project. Many project managers think that the architecture design only takes place in the design phase where all the other parts of the software are designed too – but that is wrong.

The design of software architecture should start at the very beginning in the requirements-phase. As soon as the customer requests are known, the architecture should be built – not in complete detail of course, but as a sketch. Like the scheme of a house you want to built, you begin with a rough sketch, what your house should look like and over time this sketch gets more detailed. It is the same with software architecture. In the requirements phase you design the basic functionality of your product to fulfill the customer's basic needs.

In the specification-phase, where the functionality of the software is written down, the architecture grows too. To say it more clearly, this defined functionality is part of the architecture. It describes how the system should work and how different parts should work together.

In the planning- and the design-phase the architecture should be mostly complete. After this two phases it has to be clear how the system should function, all modules and interfaces should work together and also how documentation is done. It also should be clear how modules and other components have to be tested. These are all parts of software architecture. I said "mostly complete" before because the architecture of a software product is not finished at this stage of the project. The majority of it should be completed, but it must not be frozen at this certain point.

While in the implementation phase of the project, where all parts are put together and the functionality of the whole system is tested, it could be the case that design flaws of the architecture appear, that have not been discovered before. If so, it has to be adapted and corrected. Of course, it is much more expensive to change fundamental things in later design phases as we will discuss in upcoming chapters, but if failures appear, they have to be corrected.

Still, when the system is up and running in real-life, being in the maintenance-phase, the architecture of the software can change and develop. Imagine the original software should be expanded and more functionality should be added. With these changing requirements the architecture has to change too, because it was not built for these new functions initially.

So, the development of software architecture is always in movement. It has to be done very accurate in the first phases of a software development process, but it must not stop there and has to evolve over time as well. To ensure that the design and specifications of a software project are done well, good quality management is needed.

1.1.2 Quality Management

There are many different software processes in use nowadays. But they all have one thing in common: They need good Quality-Management. Quality-Management is a very important factor in software engineering [64, 65, 69]. There are many software projects that do have little or no Quality-Management at all, which is one important reason why software projects fail. Often people think that it is not necessary, especially for smaller projects, and that it would be too expensive to do Quality-Management or even to have an own team doing it.

It is very important to do Quality-Management during the whole software development process, from beginning to end. Failures and design flaws have to be found as soon as possible and if some failures are found in later phases of development, they must not be ignored. As can be seen in Figure 1-2 it is comparative cheap to fix a problem in early software life-cycle phases. But, the later they are found, the more expensive it gets to fix them. Imagine you have an error of reasoning in the basic design of your software architecture, which may have come from false understood requirements from the customer. If you discover this failure in the design phase you can easily adopt the architecture by changing some documentation and code fragments, but if you discover it in the implementation or even the integration phase, where the software should be finished, it gets very expensive to change the architecture, because the whole system is built on it.

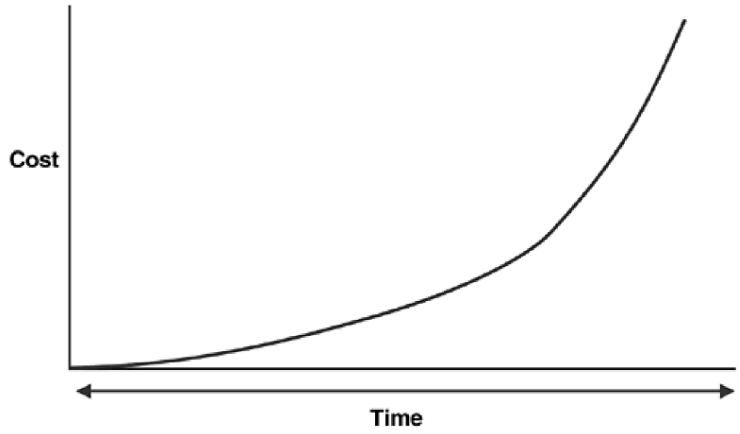


Figure 1-2 Problems and their fixing costs [5]

But when does a software product have “good” quality? There are many quality attributes that define, if a product has good or bad quality. IEEE describes a basic concept of quality attributes [6]:

- *Correctness*: Describes how the final software product works compared to the requirements.
- *Reliability*: Describes the ability of the software to provide its functionality over time.
- *Efficiency*: Describes how efficient a system is working, which means how long it takes to perform an operation, etc.
- *Integrity*: Ability of preventing unauthorized access.

- *Usability*: The ability of how easy (or difficult) it is for a user to work with the system.
- *Maintainability*: Describes how a system can be changed or expanded and how easy bugs can be fixed.
- *Flexibility*: The ability of a software system to function in an environment it was initially not meant to work in.
- *Testability*: Describes the ability of how easy it is to perform tests and test-routines (manual or automated) with the system.
- *Portability*: Describes the ability of the software to define the effort to transfer the system from one environment to another.
- *Reusability*: Means how easy a program can be reused in another system or environment.
- *Interoperability*: The ability of the system to interact with other systems to exchange information for example.

So Quality-Management is a very important and often underestimated factor, while doing software engineering.

It is essential that software architecture not only includes functional requirements of the software, but also non-functional requirements [41, 68], which describe how software should be and should act. Things like network bandwidth or a fast database server are non-functional requirements, which have nothing to do how the program works but how the software is running. If a web-server for example is designed for 10 users a day and suddenly 100 users are visiting the site a day, the site will load and code, that may run on this site, will load and run, but it may not be as fast as if only 10 users are visiting this site.

Software architecture also has to deal with such elements of software. It is not only about program code and modules. It is about the whole performance of the system. You have to think what your software should be like, and how the environment it is running in will be. These non-functional requirements have to be implemented in the software architecture with care. Functional requirements are often easier to find than non-functional ones. Functional requirements often reveal themselves through software descriptions in general. Non-functional requirements do not, because, as their name says, they have nothing to do with the function of software. Their implementation into the software architecture is not very easy and therefore very hard to change later on [29]. A good way to get a picture of what non-functional requirements a program can and must have is to evaluate the architecture of the software. There are different ways to evaluate software architectures, which will be looked at in the next chapters. You have to find the “best” or the “right” architecture for <u>your</u> software-project. There is no one, ‘right’ architecture that builds the fundament of software – there are many possible ones. To find one that suits best you have to review it – if possible from different views [5, 27, 40]. When you look at the architecture from different points of view – that may be the project managers view, the programmers view, the user-interface-designer view, etc. – many ideas may arise that would not be thought of when only looked at it from only one direction. There are various ways architecture reviews are held. They can be done by individuals – everyone reviewing on their own from a different point of view and then comparing the results – or by groups of people building a team. While in a team, every team-member has a different role. The

goal is to review the architecture together and compare the results with, maybe existing, other teams. To achieve the best result possible, the communication between the team-members must be optimal.

There are two different possibilities for team-members to communicate: sitting together physically or not sitting together physically. If not sitting together the team-members have to use some kind of electronic communication technique. But which way of communication would be the most efficient? While sitting together physically you may be able to discuss in a more efficient way, but you may save time by meeting electronically, because you do not have to go somewhere. Trying to find an answer to this question will be the main goal of this thesis.

1.2 Motivation and goal of thesis

The motivation of this work is to survey what the best possibility is to evaluate software architectures using architecture scenarios in teams. There are a lot of possibilities to do such evaluations. But one big question is: how can it be most effective. We will have a look at how such scenario evaluations can be done and if it is more effective to meet in reality or just electronically, like over a chat-application in virtual space for example.

To ensure that a software architecture is as good and detailed as possible is a major goal in software engineering. Many projects fail because of missing or bad constructed architectures. Our goal is to produce good software in projects that are in time and budget. Not only good programming is needed, but rather more important is good design-work at the beginning. If a software architecture is built well, the whole project will be finished much easier. It is like building a house: With just a small sketch on a piece of paper it would not be easy to build a house. But with exact plans and drawings it can be done with minimal problems.

In this research 54 students, who had different levels of qualification, were asked to find architecture scenarios for two different software systems. We used two different systems, because every application has different focuses and functions. Using two (and not only one) applications gives us a better and more general view on what requirements are requested. First the students had to find some scenarios individually and then in a group of three people. While in a group, there were two different ways how the students could communicate with each other. One half of the teams was communicating and discussing the scenarios face to face and the other half was communicating over the internet with a text-based chat application. To tell which scenarios are “the best”, three different reference-lists of all found scenarios were created, each based on another technique.

In this thesis we want to find the most effective way for creating architecture scenarios. We will focus on finding scenarios in teams. But what kind of team meeting would be optimal? Face to face, or communicating electronically? It also may have become difficult discussing scenarios within the team. So maybe scenarios that have been found prior from each individual are more in number and better in quality than that ones that were found by the team.

The main hypotheses that are constructed are 1) That groups that were communicating face to face have found more important scenarios than groups communicating with the chat-application, (2) Face to face teams have more important scenarios in their profiles compared to the individual phase, than

tool-support teams, because face to face communication is better and not good scenarios are left out and 3) More experienced face to face teams have better results than equally experienced tool-support teams, because knowledge and experience can be contributed better if the communication is better.

1.3 Structure of thesis

This thesis will consist of various chapters that will discuss different topics.

Introduction

The introduction should give the reader an overview what to expect from this thesis. It will give a quick look on the topics discussed and a brief introduction to the topics.

Related Work

The following chapters (2-5), that could be called “Related Work”, will give the reader a deeper and more detailed explanation on the topic. Techniques and vocabulary in general are discussed in detail and should give a good knowledge of the topic to help understand and comprehend the results and its discussion. The “Related Work” chapters should give the reader the state of the art information on the discussed topic.

Research Approach

In this chapter the hypotheses are explained, that will be discussed later in this thesis.

Experiment Description

The experiment we did will be explained in detail. It will provide enough information to understand, how the experiment was run and how we collected the data. It should also provide enough information to reproduce the experiment for further investigations.

Results

In this chapter the results of the experiment will be presented. It will deal with the main question what kind of communication leads to better and more effective architecture evaluation. With these results answers to the hypotheses postulated in the chapter “Research Approach” should be found.

Discussion

In this chapter the results will be discussed, focusing on the hypotheses postulated.

Conclusion

This chapter will hold the conclusion of the thesis and it will also give an outlook of future work on this topic.

2 Architecture in Software Processes

Software architectures a very important, if not the most important, factor of a software product and, unfortunately, it is often not done accurate. With this architecture quality goals like security, reliability, usability, modifiability, stability and real-time performance are addressed for the first time as a design artifact. [7]

This architecture has to be designed in the earlier software engineering stages. There are many different software process models in use like the waterfall-model [3], the spiral-model [4], the Rational Unified Process [5, 70], the Unified Software Development Process [71] or the V-Model.

2.1 The V-Model

The V-Model is called that way because of its V-shaped form as can be seen in Figure 2-1. It was first developed 1986 in Germany for public IT-projects but was also used in the private industry later on.

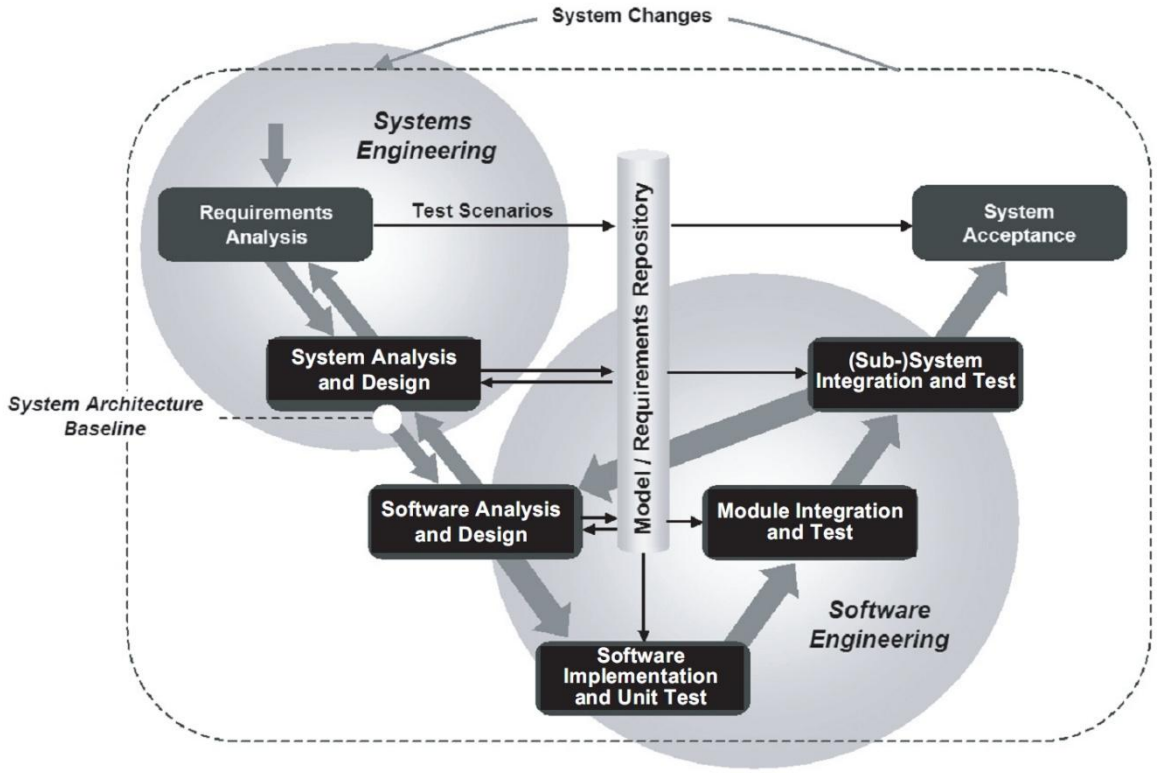


Figure 2-1 The V-Modell [51]

The idea is to start with a pre-investigation or requirements analysis of the project. Then go a little more into detail to the analysis-phase, then to the design-phase and then to the implement-phase - going all down the V getting more and more in detail. After the implement-phase has been completed, you go up the V again to the test-phase and the integration-phase. Every phase that goes

up is checked against its opposite phase. While you are testing the software-integration, you check if it is really the way you wanted it to be – or in other words, like it has been analyzed and designed before. While in the integration-phase, you check if the system is really the system you planned in the beginning, while doing requirement-analysis in the requirements analysis-phase.

2.2 The V-Model XT

The latest version of the V-Model is the V-Model XT. It has been released in 2005 and is under continuous development. The actual version is 1.3. One main advantage is that the V-Model XT can be tailored to individual needs – that's why it is called V-Model XT, XT stands for "extreme tailoring". It also brings better support for adaptability, scalability and extensibility and is also compatible to standards like ISO 9000. New in the V-Model XT is the use of various modules that are used differently depending on who is involved.

The V-Model XT also includes phases for the customer and not only for the developer, like in the old V-Model. As you can see in Figure 2-2 the so called decision gates have been extended with phases for the customer.

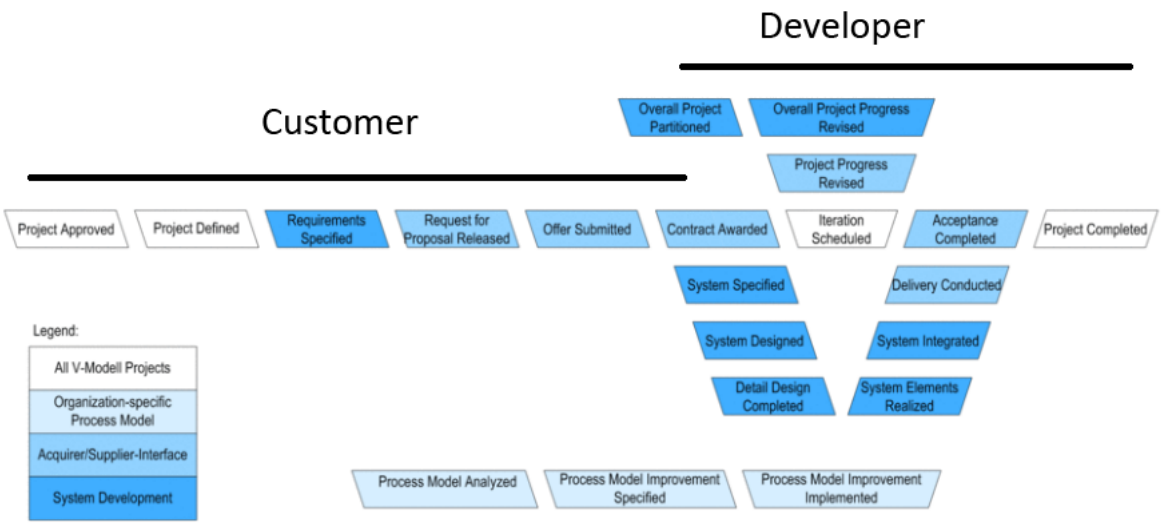


Figure 2-2 V-Model XT Decision Gates [35]

The V-Model XT is still in use for public IT projects in Germany. It can be used for small and also for big and complex IT projects, what is one main advantage of the V-Model.

When looking at the example of the V-Model, the architecture of a software project will be designed on the left side of the V. It already begins in the requirements analysis-phase, where the programmer tries to get as much information as possible from the customer. At this stage non-functional

requirements, like performance or usability, may arise, which are major and fundamental parts of a software architecture.

In the analysis phase hard- and software-requirements are evaluated and defined – in this phase the software architecture gets more detail in non-functional and functional requirements.

In the design phase code is produced, modules created, interfaces defined, etc. As we go down the V in the model – which means that we are going more and more into detail – the software architecture is getting more detailed too. The construct of the whole software product is also getting a part of the architecture.

In the implementation phase all modules are brought together and the software as a whole will be built. From this moment on we go up the V again and we begin to check continuously whether the requirements that have been set in the opposite phases have been met. If there are any design flaws in the architecture they will start to reveal from this moment on.

The test- and implementation phases of the V-Model are the last two phases. If there are fundamental errors found in the design of the architecture it is getting very expensive and time-intensive to correct these issues. Because the software is finished, little changes in the architecture, which can lead the software to a complete different direction, cause a lot of rework.

3 Architecture Evaluation Approaches

As we have learned so far it is very important to try to build the best architecture possible for a software project. Many project-managers are planning the specification- and design phases of a software-project too short, which means that there is not enough time to think about the fundamental parts in detail.

3.1 Architectural Views

Many stakeholders have to use the architecture – for different purposes and with different levels of experience. A programmer may want to see the connection between single modules, a project manager may want to get information for planning the project and the customer wants to know what to buy.

As Kruchten says [5] software architecture must not be flat – it has to be a multi-dimensional thing. All the possible stakeholders must have the opportunity to discuss, communicate and reason over the architecture.

So there have to be certain views on an architecture for various stakeholders. Each view is an abstract view from the architecture which gives the viewer the kind of information he needs. A project manager for example may not be very interested in the interfaces of software modules, nor may a programmer care for the project plan as a whole.

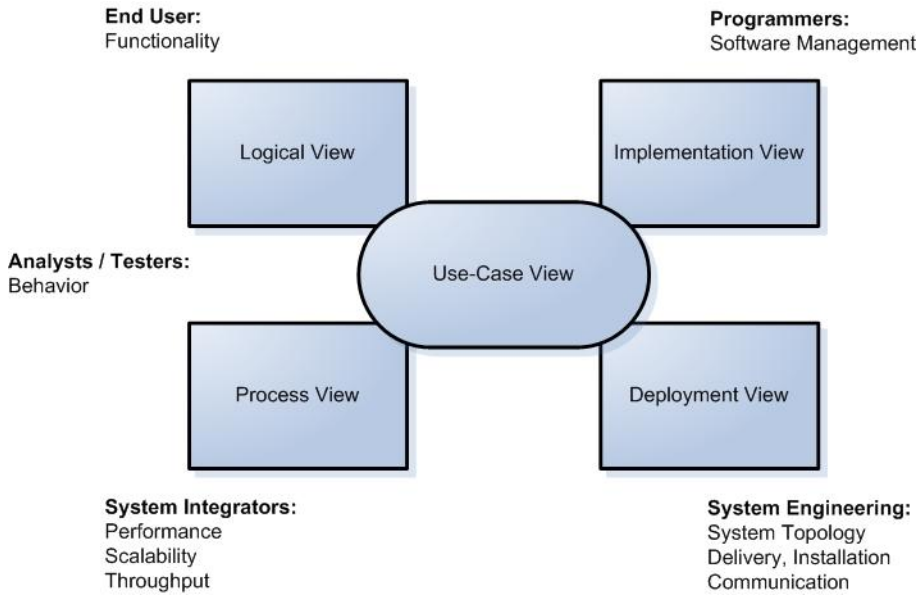


Figure 3-1 The 4+1 View Model [5]

Kruchten describes the 4+1 view model on architecture, as can be seen in Figure 3-1, as follows [5, 27]:

- *The Logical View:* This view of the architecture addresses the functional requirements of the system, in other words, what the system should do for its end users. It is an abstraction of the design model and identifies major design packages, subsystems, and classes.

- *The Implementation View:* This view describes the organization of static software modules (source code, data files, components, executables, and other accompanying artifacts) in the development environment in terms of packaging and layering and in terms of configuration management (ownership, release strategy, and so on). It addresses the issues of ease of development, management of software assets, reuse, subcontracting, and off-the-shelf components.

- *The Process View:* This view addresses the concurrent aspects of the system at runtime—tasks, threats, or processes as well as their interactions. It addresses issues such as concurrency and parallelism, system start-up and shutdown, fault tolerance, and object distribution. It deals with issues such as deadlock, response time, throughput, and isolation of functions and faults. It is concerned with scalability.

- *The Deployment View:* This view shows how the various executables and other runtime components are mapped to the underlying platforms or computing nodes. It is here that software engineering meets system engineering. It addresses issues such as deployment, installation, and performance.

- *The Use-Case View:* This view plays a special role with regard to the architecture. It contains a few key scenarios or use cases. Initially, these are used to drive the discovery and design of the architecture in the inception and elaboration phases, but later they will be used to validate the different views. These few scenarios act to illustrate in the software architecture document how the other views work.

These different views on software architecture build the base on modern architecture reviews. Parnas and Weiss have defined a way to review designs generally – Active Design Reviews (ADRs) [8]. ADRs make use of the roles different persons have in a software project and use the various views they have on the architecture. Participants of a review are chosen because of their role. In ADRs every participant is given a questionnaire and / or some exercises to complete. The result is, that the artifact being reviewed, is actually exercised. [7].

The Software Engineering Institute CMU has decided to include Parnas and Weiss' idea of ADRs in terms of architecture reviews to the software life-cycle process. [9] There are now various methods of architecture design by the Software Engineering Institute:

- Architecture Tradeoff Analysis Method (ATAM) [10, 37]

- Software Architecture Analysis Model (SAAM) [11, 36]

- Architecture-level Modifiability analysis (ALMA) [15]

- Performance Assessment of Software Architectures (PASA) [16]

- Active Review for Intermediate Designs (ARID) [12]
- Attribute-Based Architectural Styles (ABAS) [13]
- Cost Benefit Analysis Model (CBAM) [14]

We will have a look at some of these models in this chapter. Most methods have in common that they try to find out what the non-functional requirements of the software are - or should be - and then to check whether they are covered by the architecture or not. Also the Software Performance Engineering (SPE) [30, 31, 32] and ArchOptions [33, 34] are very active in this area of software architecture.

Architecture reviews can be seen in various ways. They can be experiments, modeling, walk-through scenarios [7], etc or simply questioning- or measuring-techniques (or hybrid) as Abowed et al. formulates [17]: Questioning-techniques are questionnaires, scenarios or checklists and measuring-techniques are simulations, prototypes or experiments on software systems – things where you have something to measure. Hybrid-techniques combine both techniques. Most architecture review methods are hybrid-techniques – there are questions asked and results measured.

3.2 Scenarios

Scenarios are well-known approaches to get a well-defined focus on what is needed and how the architecture of a software product should look like to meet the non-functional requirements of the product.

A scenario is a brief description of a single interaction of a stakeholder with a system. Scenarios are used to precisely define quality (also called non-functional) attributes, as mentioned in [39].

There are usually two classes of scenarios:

- General scenarios, which are system independent scenarios to guide the specification of quality attribute requirements.
- Concrete scenarios, which are system specific scenarios to guide the specification of quality attribute requirements for a particular system. They are usually instances of General scenarios.

Scenarios can fit to various areas of software engineering or viewpoints. There can be scenarios that may fit to the user-interface area or scenarios that fit to performance or security. When doing architecture evaluation with change categories provided the participants have to find scenarios for the various categories given, like performance, security, user-interface.

In the following there are a few examples on architecture evaluation scenarios:

- The user wants to examine budgetary under different fiscal years without reentering project data (usability)
- The user changes a graph layout from horizontal to vertical and the graph is redrawn in one second (performance)

- A remote user requests a database report via the Web during a peak period and receives it with-in five seconds (performance)
- A data exception occurs and the system notifies a defined list of recipients by email and displays the offending conditions in red on data screens (reliability)

3.3 Processes of Architecture Evaluation

There are many different approaches for software architecture evaluation. They all have a different approach on the topic (scenario-based [67], use case maps [55, 56, 57, 63], etc.). Some techniques do base on others and some are developed independently. We will have a look at common techniques in this area and describe where the focuses lie at the different approaches. Most of them use scenarios for architecture evaluation and that is why we chose to use change categories for our controlled experiment as well. Scenarios are collected differently in these techniques. They can be found through brainstorming-sessions, collected from stakeholders through interviews and so on. The focus of all techniques is the same: finding the most useable architecture for a software product to fulfill all functional and non-functional requirements.

3.3.1 ATAM

ATAM gives an overview of how an architecture coheres with non-functional requirements and it also shows how these requirements are related to each other. ATAM is a method that uses scenarios for architecture evaluation. ATAM uses three types of scenarios [7]:

- Use case scenarios: These scenarios describe the behavior of and the interaction with the final software.
- Growth scenarios: These scenarios describe growing / scaling of the software or system.
- Exploratory scenarios: These scenarios show the limits of the system. How far it can go and where problems occur (stress testing).

The ATAM uses various viewpoints of different stakeholders, the architecture itself and business goals of the software system as base for evaluation. During analysis of the evaluation non-functional requirements are combined with scenarios that have been and found and rated. So problems and risks that may exist can be found much more easily. ATAM can be done during every stage of developing though, but it is most useable in early project phases.

ATAM is influenced by three areas:

- The Software Architecture Analysis Method (SAAM)
- Quality Attribute Communities
- The notion of architectural styles [38, 47, 48, 49, 53]

Quality Attribute models help to understand the architecture in a better way, while the concept of architectural styles, which are descriptions of component types, patterns of run-time control, etc., define a set of architectures [39]. The architectural style ATAM uses is called ABAS.

The ATAM method consists of nine steps (step 0 to step 8), which are as follows [52]:

0) *Planning / Information exchange*: Method is described to stakeholders and their main non-functional requirements are learned.

1) *Scenario Brainstorming*: Most important stakeholders do scenario-brainstorming.

2) *Architecture presentation*: The architecture is explained in detail and found scenarios are mapped.

3) *Scenario coverage checking*: Questions about non-functional requirements are used to check the coverage by found scenarios.

4) *Scenario grouping and prioritization*: Stakeholders vote the most important scenarios.

5) *Map high priority scenarios onto architecture*: For every high priority scenario it is shown how it affects the architecture.

6) *Perform quality attribute-specific analysis*: Models are built based on non-functional requirements. Then input parameters are changed and so sensitive points are evaluated.

7) *Identify trade-off points*: All architecture elements with multiple sensitivities are located.

8) *Consolidate findings and develop action plan*: A set of recommendations to improve the architecture are given.

When all these steps have been gone through, a decision is made whether or not the architecture will be changed. If the architecture is changed the circle begins again at step 1 “Scenario Brainstorming”.

ATAM attributes

The results of ATAM are dependent on the experience and skills of the participants and also from the quality of the scenarios. The better the scenarios, the better the outcome. ATAM delivers good documentation and helps stakeholders to better understand the architecture and the goals of the software.

3.3.2 SAAM

The SAAM needs a clear description of the architecture before evaluation can be done. It defines three perspectives (which are recognized perspectives in literature [1, 28, 42, 43]) for understanding and describing architecture – functionality, structure, allocation (not like ATAM which uses much more different quality attributes [7]) – and also provides a simply language to describe the structure [11] :

- *Functionality*: Functionality describes what a system is doing, how it is operating and how it is functioning. Depending on the size of the system it may be one function or a bundle of different functions that describe the functionality of a system. Kazman et al. mention that normally a system's functionality is decomposed through structured analysis or object oriented analysis, but in their case a domain (databases, user interfaces, etc.) analysis is used for partitioning of functioning.

- *Structure*: The structure of a system describes of which parts it consists of. It describes the components like a module or an object and the connection between them. In the SAAM special icons are used to describe these structure elements as can be seen in Figure 3-2.

- *Allocation*: The allocation of the functionality of the software-system to its structure shows how the functionality is realized in that certain case. There are many ways how the functionality of a system can be realized and to show which way is used to reach the functionality demanded, the allocation-perspective is used.

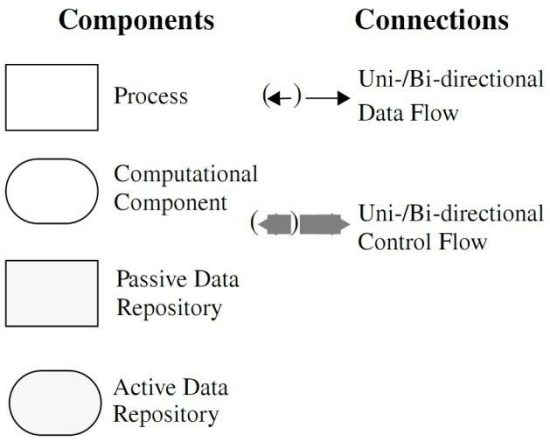


Figure 3-2 SAAM elements of structure description [11]

After the architecture of the software system is described the following analyzing steps are applied for evaluation [11]:

- Characterize a canonical functional partitioning for the domain.

- Map the functional partitioning onto the architecture's structural decomposition.

- Choose a set of quality attributes with which to assess the architecture.

- Choose a set of concrete tasks which test the desired quality attributes.

- Evaluate the degree to which each architecture provides support for each task.

SAAM is scenario-based, as is ATAM, but in SAAM scenarios are grouped into direct and indirect scenarios [7]: Direct scenarios are scenarios that are supported by the architecture and need no modification while indirect scenarios need modification on the architecture to be supported. For

evaluation SAAM uses change scenarios from various stakeholders which are rated and then mapped onto the architecture. While mapping, risks or problems in the architecture show up.

SAAM attributes

SAAM has a relative flexible process and, like ATAM, is dependent on the experience and the skills of the participants. With SAAM you also get a better view and description of a software architecture.

3.3.3 ALMA

ALMA also uses change scenarios for evaluating software architectures, which help to get a picture of how big the effort might be to make changes to the architecture once the system is completed. In other words it shows how flexible the architecture in terms of modifiability is [18].

ALMA uses the following five steps [15]:

1) Set the goal.
2) Describe the software architecture.
3) Elicit scenarios.
4) Evaluate the scenarios.
5) Interpret the results.

Ad 1: In this step the questions is asked what goal to reach with the evaluation. There are three goals that can be met:

- *Prediction of maintenance cost*: This goal shows the how difficult it will be to change the software-system in the future to fulfill future requirements.
- *Risk assessment*: This goal shows where the software architecture will be inflexible so that changes cannot be made.
- *Selection of software architecture*: This goal helps to find the best architecture available when comparing different architectures.

Ad 2: How well a software architecture can be described depends on the time evaluation is done. As the architecture grows with the process of the software-project some effects, relationships or dependencies may not visible in earlier design stages – which does not mean that architecture evaluation should be done later in a project. It means that some scenarios might not show the full effect on the architecture because some relationships will become visible at later design or implementation stages [15]. In ALMA different views on the architecture are used, like Kruchten explains [5][19]. Architectures are often described in architecture description languages (ADLs) [50]. The views that are used should give information about the following [15]:

- The decomposition of the system in components,

- The relationship between those components and
- The relationship to the system's environment

Ad 3: In this step scenarios are found by various stakeholders. To limit the number of different scenarios two techniques are used: equivalence classes and / or classification of change categories. While equivalence classes group similar scenarios that mean or describe the same thing in a broader perspective, change categories are defined categories where scenarios can fit in. There are two ways to define those categories: First they can be defined at the beginning and found scenarios are then assigned to a category (top-down approach) or second, scenarios are found in the beginning and categories are defined later using the information of the found scenarios (bottom-up approach).

Ad 4: In this step the effect of the scenarios on the architecture are discussed. This analysis consists of three steps: [15]:

- *Identify the affected components*: In this step the components that are affected through scenarios have to be identified.
- *Determine the effect on the components*: After the components are identified it has to be made clear how and which functions of the components are affected.
- *Determine the ripple effects*: In this final step the ripple effects a scenario causes have to be identified. There may be little ripple effects, so that only a few components are affected, or there may be large ones where every affected component has an effect on other components too.

Finally every scenario can be ranked on how large its impact on the architecture is which helps to compare the effects of different scenarios.

Ad 5: After evaluation interpretation has to be done which depends completely on the goal that has been set and the requirements of the software.

ALMA attributes

ALMA only focuses on modifiability attributes, which may be not enough in some cases. The skills and experience of the participants is very important, like seen before at other methods.

3.3.4 PASA

PASA, as the name gestures, focuses on performance. Williams and Smith [16, 45] say that performance cannot be retrofitted: it must be designed into software from the beginning. They say that not false coding but inappropriate architectural choices cause performance issues. PASA is based on software performance engineering (SPE) [44] and may be used for new systems in development or for existing systems which should be upgraded. PASA is also scenario-based. Scenarios are found for use cases that have been identified before and are then documented using UML sequence diagrams [46]. These use cases address critical circumstances in terms of responsiveness or scalability

of a system [7]. Usually key developers and project managers are participants in PASA. For every key scenario performance objectives have to be found and then it is checked whether the architecture does support those objectives or not. As mentioned by Williams and Smith [16] the PASA process consists of ten steps:

1) *Process overview*: Presentation of evaluation process, the outcomes and the reason in general to participants (managers and developers).
2) *Architecture overview*: Developers present the architecture of the software.
3) *Identification of critical use cases*: Behaviors of software are identified.
4) *Selection of key performance scenarios*: For every important use case, performance critical scenarios are identified.
5) *Identification of performance objectives*: For every scenario, objectives are identified.
6) *Architecture clarification and discussion*: Key performance scenarios are discussed in context with the architecture and specific features in more detail.
7) *Architectural analysis*: The architecture is analyzed to check if performance objectives are supported.
8) *Identification of alternatives*: If a problem is identified, alternatives are searched.
9) *Presentation of results*: Results are presented and recommendations are given.
10) *Economic analysis*: Cost and benefits are presented and improvements are shown.

PASA attributes

PASA focuses on performance, which is a very important thing, but may be not enough for various software projects. PASA works with use-cases, which is a good approach, because actual interaction with the software is analyzed.

4 Team-Meeting collaboration

The most important focus in this thesis is: team-meetings. We will compare different methods of holding team-meetings and evaluate which one delivers the best results.

Team-meetings are held in various contexts. In software development for example they are held while doing inspections for finding failures in programs. In an inspection phase, reviewers try to find failures in the software product individually and then come together in a meeting to discuss the failures they have found in a group. Fagan showed the benefit of doing inspections to find failures in the phase, where they have been created and not in later phases, where it is much more expensive to correct that failure [26].

But doing team-meetings is not always more efficient than doing things individually – it depends on how team-meetings are held. Many stakeholders (if not the most) think that team-meetings, especially those done during software inspections, are a good thing, but as Lawrence and Votta [23] showed, those meetings are not as beneficial as most participants think they are and mostly they even cost more than their benefit is.

There are not many evaluations about team-meeting collaboration in a software engineering surrounding. That's why we have a look at Lawrence and Votta's work. They evaluated the usefulness of team-meetings in inspections of software, which is a similar approach to doing team-meetings for software architecture evaluation.

4.1 Inspections in software engineering

Inspections are done throughout a software developing process. First various reviewers try to find failures in different kinds of elements of the software project. These could be lines of code, diagrams, use-cases, etc. Then a meeting is held, where all people involved participate. In this meeting all failures found are discussed and afterwards all those failures have to be fixed.

Lawrence and Votta [23] showed that doing small team-meetings with 2 or 3 people is much more efficient than doing meetings with more participants (and more roles). The more people should attend to a meeting the more likely it is that the meeting will start late. It is also very difficult to find an appointment where everybody can attend. 20% percent of the time of one inspection cycle is waiting for all participants to meet [24].

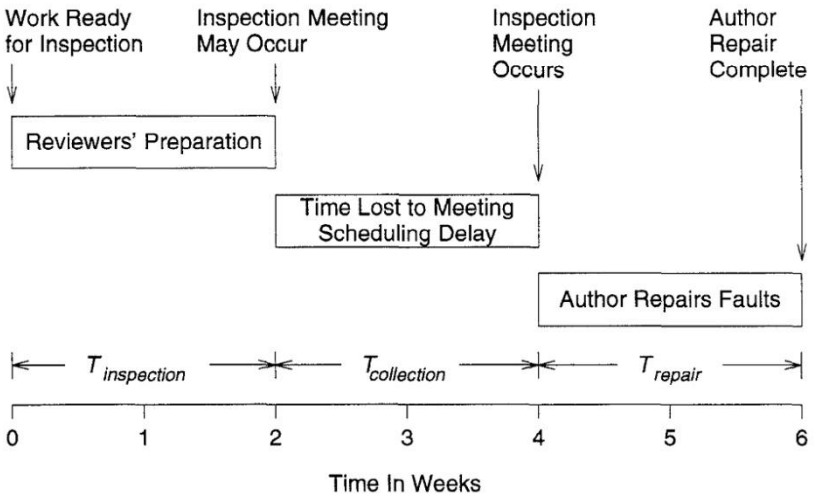


Figure 4-1 Inspection Meetings – Timetable [23]

In Figure 4-1 it can be seen how long it actually takes to complete one inspection iteration on average, based on the evaluation done by Lawrence and Votta. The interesting part is $T_{collection}$ which shows the time-span from where the meeting could be held to when the meeting actually occurs. In their study it is 2 weeks in average. In these 2 weeks nothing happens in terms of inspection work. The problem is that too many people attend to classical inspection meetings. There are many reviewers, a moderator, a scriber, author, etc. And as said before: The more people should attend, the more difficult it is to find a suitable appointment. So the 2 weeks is a time-span where all participants wait for the meeting to occur.

But does that mean that team-meetings are not effective and not useable in general? No. It depends on how team-meetings are held. Inspection meetings also do have positive aspects as they provide a deadline when everything has to be finished and follows a thought through process [25]. Lawrence and Votta provide two different alternatives that do fit to our approach of doing team-meetings:

- Small meetings (face to face) done by 2 or three people
- Do meetings using verbal or written media (telephone, chat, notes, etc.)

Their results are that doing smaller meetings reduces inspection cycle times dramatically, reduces administration overhead and has little or no loss in quality. They found out that larger meetings may be 4% more effective maximum, but because of longer cycle times and more overhead this advantage is reduced and even exceeded by the higher costs.

4.2 Team-meetings in Architecture Evaluation

In our study team-meetings were held in groups of three people. Every team-member had a specific role: Moderator, Scenario-Scriber and Time-Keeper. According to the study explained in this chapter, our group size of 3 three people has to be very good. We will have a look on how the groups succeeded compared to the individuals to see, if the size of 3 people in a group is a useable one. We also will have a look at the second alternative mentioned: doing meetings using tool-supported communication. We will compare teams doing communication face to face, sitting together physically, with teams doing tool-supported communication.

While discussing scenarios in teams a better result can be expected than trying to find scenarios alone. It is very important to find the best scenarios possible for architecture evaluation. When doing scenario brainstorming alone, some scenarios may occur, that are not very useful at all. By discussing them in teams, useless scenarios are identified and left out and only good and useful scenarios will make it through the team phase of evaluation.

5 Different approaches on Architecture Evaluation

As can be seen in the methods described in chapter 3, scenarios are used very often to do architecture evaluation and to qualify non-functional requirements. Every method has its own definition of scenarios (direct, indirect, growth, etc.), but they all have in common that it is very important to ensure that the quality of those scenarios is as good as possible. If the scenarios found, or the reference-list of scenarios to which they are compared to, are not good or not meaningful enough then the possible resulting adaption of the architecture may be a step back and not forward.

So, the outcome of an evaluation is bound tightly to the quality of the scenarios used [15, 20].

5.1 Methods of evaluation

There are various ways how scenarios can be found in various architecture evaluation methods. Stakeholders can do individual brainstorming-sessions where they try to think about scenarios alone. Later all found scenarios are brought together and are evaluated. Another option is to form groups of people and let them discuss in teams which scenarios might be useful or not [21]. When thinking about scenarios alone, stakeholders may write down scenarios that may not be very useful or obsolete if inspected more in detail. While discussing in a group such scenarios may not be written down, because other team members might start a discussion about the usefulness of a scenario. As far as we are concerned there has been no empirical study if discussing scenarios in a team reveals better results than trying to find scenarios alone – this is one point that will be discussed in this thesis.

5.2 Methods of communication in teams

While communicating in a team, various ways of communication can be chosen. Team-members may come together at the same place physically and discuss scenarios face to face. Another way of communication might be over longer distances, so that team-members may not come together physically but with some technical help. Not meeting in reality has some advantages: Time that is usually spent for arrival and departure is no lost time, like it is when meeting in reality, because there is no arrival and departure. When stakeholders of companies meet, such travels can become expensive – time is money. But if the participants of architecture evaluation are communicating over phone, chat, etc. time can be used more efficient. This will be another topic of this these – which kind of communication will bring better results: Meeting face to face or communicating with some kind of tool-support.

5.3 Experience and guidelines

It is also at hand that stakeholders, contributing to an architecture evaluation, may have different levels of knowledge and experience. This can have several reasons. Some may have a better education or just have a better insight in the software-product, because of the role they have in development. For example a programmer might have more information on the architecture – or with architecture evaluation in general – than a "normal" user who will be using the software product in the future. Babar and Biffl mention [22] that different experience levels of participants do have little differences that can be seen, but did not have significant influences on the results. Babar and Biffl were trying to find differences between evaluation participants, who had some guide (change categories to fit scenarios in) finding scenarios, and participants who did not. An important question, that will be the third and final question of this thesis, will be if the experience level of the participants has some influence in the communication technique. It may be possible that more experienced participants can contribute their wisdom better when communicating face to face.

Babar and Biffl were evaluating the difference between two different groups of participants. One half of the participants was handed out a set of change categories which should help them find scenarios for a specific system [22]:

- User-Interface changes
- Security-policy changes
- Performance & scalability changes
- Workflow management changes
- Content management changes

These categories should help the participants to find suitable scenarios. Maybe they would not have thought of scenarios for the content-management part of the application for example. The other half of the participants did not have these categories and had to find scenarios without them. The idea was to find out which method would have better results for architecture evaluation – should participants be guided, so they can focus on what scenarios they have to find, or not, letting them find scenarios freely. The set of categories was carefully chosen of course so they covered the topics of the software evaluated.

The result was that participants, who had a set of change categories provided, did have better results than the other ones.

5.4 Evaluation in our study

Our study not only did distinguish between different meeting-styles (Face to face and tool-support) but also handed out categories to one half of the participants too. Because of the fact that 'change categories' provided leads to other results than 'no categories provided', we will distinct the results from each other as well. Discussing the hypotheses will also be separated in those groups who used categories and those who did not, if necessary. Because the results in this matter are different, it is good to have a view on the results from different viewpoints.

The quality of a scenario or a scenario set has to be classified to say if it is useable or not. For that matter one method would be to compare a scenario set to a reference scenario set. This method was used by Bengtsson [20] and Babar and Biffl [22]. There are many ways to create a reference scenario set. One possibility may be to let experts create a list of scenarios that are rated by importance and use this list as a reference set, to which other sets (the scenarios individuals or groups had found before) are compared to. Another method is to rate scenarios because of their frequency – that is what Babar and Biffl did for example [22]: If a scenario is found by a participant, he thinks that it is important somehow. If this scenario now has been found by all participants it can be considered to be the most important scenario of all – the more it appears in different scenario profiles, the more important it will be.

6 Research Approach

There are many ways of doing software architecture evaluation – it can be done individually by its participants, or in groups of people for example. It is important to find the best method possible to ensure perfect quality of the software product with appropriate costs and finalization in time. Many approaches have been analyzed, but some have not yet.

Doing evaluation in teams has not been evaluated extensively. For doing architecture evaluations in teams, its members may come together at the same place physically and discuss scenarios face to face, or they might do tool-supported communication. When communicating over phone, chat, etc. time can be used more efficient. It is also cheaper, because travelling to and from the meeting-place costs money. Doing tool-supported communication also means that participants all over the world can be part of the evaluation process. The question which kind of communication will bring better results - meeting face to face or communicating with some kind of tool-support – has not been discussed or evaluated in detail. It is also a very important one, because doing tool-supported communication can save a lot of money, if the quality level of the results is at least as good as results from evaluations doing face to face meetings.

But architecture evaluation must not be done in teams. It can also be done by stakeholders individually. But, when thinking about architecture scenarios alone, stakeholders may write down scenarios that may be not very useful or obsolete if inspected more in detail. While discussing in a group, such scenarios may not be written down, because other team members might start a discussion about the usefulness of a scenario. A comparison of results from evaluations done by individuals and evaluations done by teams is a very interesting task.

Another important question is if the experience level of the participants has some influence in the communication technique. Evaluating, if experience has an impact on architecture evaluation in general, has been done before [22, 61], but there has been no focus on different meeting-styles, like face to face and tool-supported communication. It may be possible that more experienced participants can contribute their wisdom better when communicating face to face.

To analyze such questions a controlled experiment is needed [59, 60]. In this experiment, that was held in the summer term 2008 at the Vienna University of Technology, real circumstances of software architecture evaluation were rebuilt. 54 students, who had different levels of qualification and experience, participated in the experiment. They had to do architecture evaluation individually and in teams. While doing evaluation in teams one half of the students did face to face communication and the other half was doing tool-supported communication. In this experiment two different applications have been evaluated. This gave us the chance to let teams change the meeting style in the second round. So every group of participants was doing face to face and tool-supported communication. This controlled experiment gave us the necessary data needed to do research in the following topics.

6.1 Comparing different methods of team-meetings

The first experimental hypothesis is that teams that communicate face to face find more important scenarios, than teams communicating with some kind of tool-supported communication. In our controlled experiment each participant was trying to find architecture scenarios alone first in an individual phase. Then, in the team phase, they came together in teams to discuss those scenarios in groups. During this phase one half of the groups was sitting together physically, face to face, in the same room, while the other half was not sitting together – every participant of a tool-support team was sitting in front of his own computer, communicating with his team members over a chat-application. Both teams had 60 minutes to discuss the architecture scenarios found in the individual phase (and also to find new ones) and wrote those down, which the whole group thought of being important.

More formally, the first null hypothesis is:

H0: The face to face discussion of software architecture scenarios for architecture evaluation in teams will not have better results than discussing them over tool-supported communication.

The first alternative hypothesis is:

H1: The face to face discussion of software architecture scenarios for architecture evaluation in teams will have better results than discussing them over tool-supported communication.

If the face to face communication has better results than the tool-supported communication in our experiment, we will reject H0.

6.2 Comparing individuals and teams

The second experimental hypothesis is that teams find more important scenarios, than individuals. In our experiment every participant was trying to find scenarios in an individual phase alone first. In this phase he might have written down everything that came to his mind. In the team phase afterwards every scenario should have been discussed to write it down or not. That means that the results in the team-phase should have better results, than those in the individual phase, because useless scenarios should have been left out. But the main thing here is that every single scenario must have been discussed. So we will have a look on both communication styles, face to face and tool-support, to evaluate which form of communication had better results compared to the individual phase. If the scenarios have been discussed well, the amount of more important scenarios in the specific scenario profile has to be higher as if the scenarios have not been discussed well.

The second null hypothesis is:

H0: Teams communicating face to face cannot do better communication and so cannot discuss scenarios for software architecture evaluation more in detail than tool-supported communicating teams, which does not result in a higher amount of more important scenarios in their scenario profiles compared to the individual phase.

The second alternate hypothesis is:

H1: Teams communicating face to face can do better communication and so can discuss scenarios for software architecture evaluation more in detail than tool-supported communicating teams, which results in a higher amount of more important scenarios in their scenario profiles compared to the individual phase, because not useful scenarios are left out.

If the face to face teams have better results compared to the individual phase than tool-supported teams, H0 will be rejected.

6.3 Importance of experience in team-meetings

The third and final experimental hypothesis is that teams with more experience have better results, meaning that they have found more important scenarios, than teams with less experience. In our experiment every participant had to fill out an Experience Questionaire at the beginning. Later the participants were grouped to teams randomly (with the aspect of getting as equal face to face and tool-support-teams and “categories used” and “categories not used” teams as possible). Because the experience level of every participant was evaluated, the experience level of every team can be calculated. By comparing the experience-level of every team with their results it can be seen what influence experience has on the results. We will compare face to face and tool-support teams in that matter to see if there are different results. While communicating face to face every team-member might be able to contribute his experience better than team-members doing tool-supported communication. Therefore more experienced face to face teams would have better results than more experienced tool-support teams.

The third null hypothesis is:

H0: More experienced teams communicating face to face have not found more important scenarios for software architecture evaluation than teams doing tool-supported communication, because participants cannot contribute their experience better when communicating face to face.

The third alternate hypothesis is:

H1: More experienced teams communicating face to face have found more important scenarios for software architecture evaluation than teams doing tool-supported communication, because participants can contribute their experience better when communicating face to face.

If face to face teams have better results than tool-support teams with the same experience level we will reject H0.

7 Experiment Description

In this chapter the experiment, which is the base of our research and of this thesis, is explained. In the beginning a short description and overview of the experiment design is given. Then all important terms of the experiment are explained in detail. After knowing all necessary parts of the experiment its procedure is explained in detail.

7.1 Experiment Process overview

The data that is used and discussed in this thesis was collected through an experiment in the summer term of 2008 at the Vienna University of Technology. 54 students participated in this experiment.

The students had different levels of education. One part of the participants were bachelor-students, at the beginning of their study, the other part were master-students nearly at the end of their study. To measure the level of qualification we handed out an experience questionnaire to them to fill out – but more on the various questionnaires will follow later. Before the experiment started there was a two-hour introduction on software architecture evaluation, finding evaluation scenarios, etc. and the participants received some information a week before the experiment so they knew what to do during the experiment.

There were two dates on which the experiment was held, because not everyone had time on the same day. So the number of people was split and the experiment took place on two different days under the same circumstances. In the following, the sequence of one day will be explained in detail.

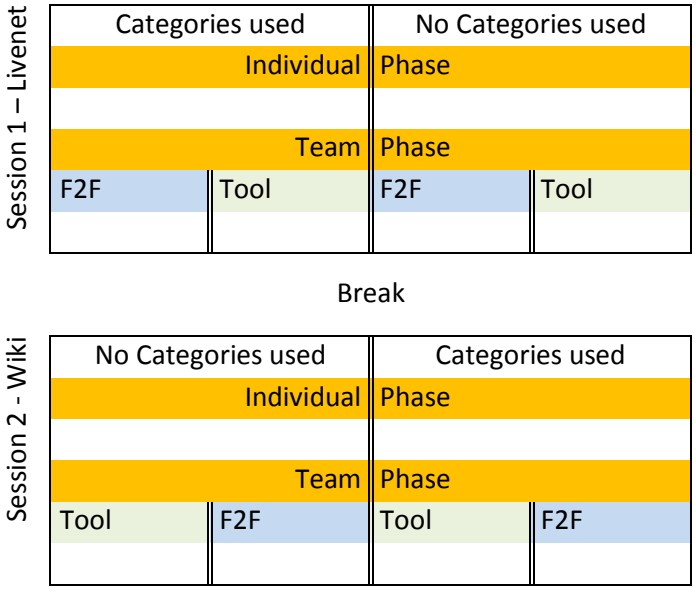


Figure 7-1 Experiment basic setup

Basically the students had to pass two different sessions as can be seen in Figure 7-1. In the first session the focus was held on an application called Livenet and in the second on a Wiki-system. In both of these sessions the students had to find scenarios for the given application on their own first – that was called the individual phase. After that they came together in groups of 3 people to discuss their scenarios and find scenarios as a team. In this so called team phase two different team-meeting styles were held: The one half of the groups was meeting directly, face to face, to discuss their scenarios. The other half could only communicate over a chat application and was not sitting together.

Finally there was a differentiation of all the students: One half of them had categories of scenarios given to help them finding scenarios. The other half did not have these categories. Certainly the groups that were built in the team-phase also were consisting of either students that had categories or did not have them – there was no mixing.

In the second session, where the focus was set on the Wiki-System, categories and team-meeting styles were changed, which means that students who had categories in session 1 did not have any in session 2 and vice versa and groups that were communicating face to face in session 1 did communicate with tool support in session 2 and vice versa.

7.2 Participants

As can be seen in Table 7-1, 54 students did participate on the experiment in total. There were 19 students from a master-study and 35 students from a bachelor-study participating. On June 3rd 31 students were participating in total and on June 6th 23. On both days the experiment was held in exactly the same way.

June 3rd		June 6th		Total
Master-Students	16	Master-Students	3	19
Bachelor-Students	15	Bachelor-Students	20	35
Total	31		23	54

Table 7-1 Number of participants

Before the experiment started, the participants were divided in groups. This had several reasons. The first one was, that students had to come together in teams during their experiment discussing their scenarios. But the even more important reason was to ensure that nearly the same number of students were using and not using change categories and that nearly the same number of students and groups were communicating face to face where others were communicating electronically. This was very important for the research afterwards. We needed nearly the same amount of participants in every possible constellation to have good and meaningful data for later analysis.

Livenet	Categories used	No Categories used	Total
Toolsupport	14	16	30
F2F	13	11	24
Total	27	27	54

Table 7-2 Number of students in Session 1

Wiki	Categories used	No Categories used	Total
Toolsupport	11	11	22
F2F	16	14	30
Total	27	25	52

Table 7-3 Number of students in Session 2

Table 7-2 and Table 7-3 give you an overview of how many participants were put into which group. They give you the numbers of session 1 (Livenet) and session 2 (Wiki). In session 2 the teams switched:

- Students that used categories in session 1 did not in session 2 and vice versa.
- Students that communicated face to face in session 1 communicated with tool-support in session 2 and vice versa.

For session 1 (Livenet), as you can see in Table 7-2, there are 27 students in one categories-group and also 27 in the other. Also, 30 students are in one meeting-style group and 24 in the other. This gave us a very good base to start with a good focus on meaningful data. As can be seen in Table 7-3 two participants left after session 1. These two participants formed a 2-man team, so it was no problem for us to leave these individuals / this team out in our evaluation of session 2.

Considering that we had to group all participants into “categories used” and “categories not used” and into “face to face” and “tool-support” teams we have a very good average number of all different group-styles possible.

There have been two different phases in every session too in which the participants had to find architecture scenarios – an individual and a team phase. In the individual phase they had to find scenarios on their own and in the team phase the participants discussed their scenarios in groups of 3 people.

7.3 Material

In this chapter we will describe all kinds of materials that have been used during our controlled experiment, like the applications we evaluated, Data Capturing Sheets and other Questionaires, etc.

7.3.1 The two applications evaluated

In this experiment two different applications were evaluated. The participants tried to find scenarios for each of the two applications. We thought that using two different applications will give us a better overall picture of the results.

7.3.1.1 Livenet

Livenet (http://livenet4.it.uts.edu.au) is a collaborative application which is web-based and supports synchronous (same-time, different places) and asynchronous (different time, different places) activities. In this application a user is able to communicate with others via discussion-forums and real-time chat. Every user is assigned to a different role, which means that every user has different rights within the system. A team leader might schedule tasks for his team members, or a "normal" user may create a workspace where he can give some users access to. In such workspaces documents can be stored. Depending on the access rights of the user, documents can be read, modified, deleted and uploaded to the workspace.

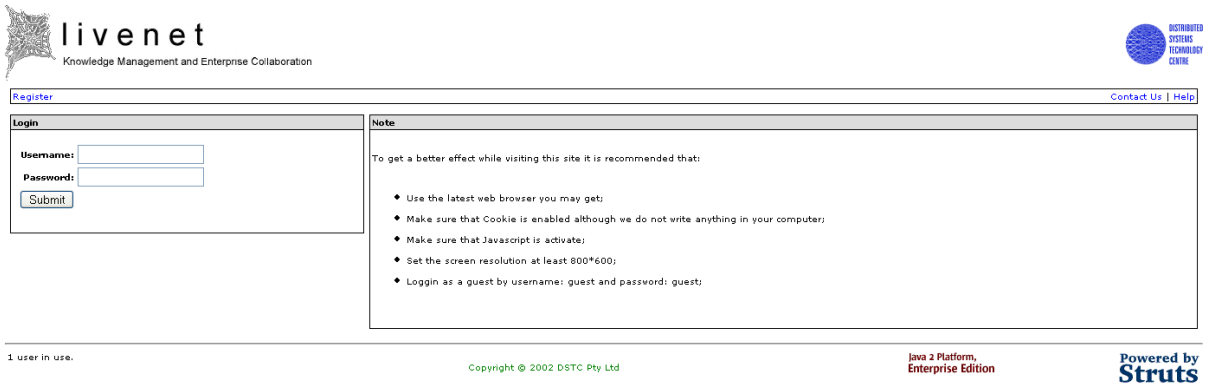


Figure 7-2 Livenet log-in screen

The Livenet application which has been evaluated by the participants had the following features and a guide describing them was handed out at the beginning of the experiment (taken from the experiment description):

- A user can register with the system. Once registered, a user is assigned a portal to perform a number of collaborative and knowledge management activities. For example, a user can create a number of workspaces, one for each unique activity that needs to be performed, a

number of roles can be created and these roles can be assigned to various activities and individual participants of those activities can be placed in those groups for ease of privilege and security management.

- Once registered and assigned to a particular portal or workspace, a user can use a web browser to access and view an online version of the documents prepared and uploaded to the workspaces.
- All the users have full editing privileges to any artifact assigned to them. A user needs to select the desired artifact and click "edit" to make changes to that artifact.
- A user can provide a brief description of the artifact created in a workspace. The system makes this brief description available to all the users who can access that particular artifact.
- If this tool is used to support a particular activity of software development process, a large number of documents can be made available to the members of the team by uploading those documents. For example, design diagrams, checklists to ensure the quality of the process, an online version of any standards that must be complied with.
- A user can send text messages to other members of the team on various issues of importance. A user can also use discussion forums to discuss various issues and discussion forums can serve like organizational memory to be accumulated over the life time of the project.
- A user can assign various artifacts to different team members.
- The system provides an online chat room that can be used for synchronous meetings, while discussion forums can be used for asynchronous meetings.
- A user can send an email to other members involved in the same activity.
- A user can invite new members to join a group or activity.

The participants were told that Livenet will be used in a company to support a distributed software architecture evaluation process. Their task was to think about changes that could be necessary to fulfill those requirements in the present state and in the next three years of time. These changes have been described by the participants as change scenarios.

7.3.1.2 Wiki

The Wiki-system is a web-based collaborative content management system in which content can be organized by the participants, like you may know from Wikipedia.org.

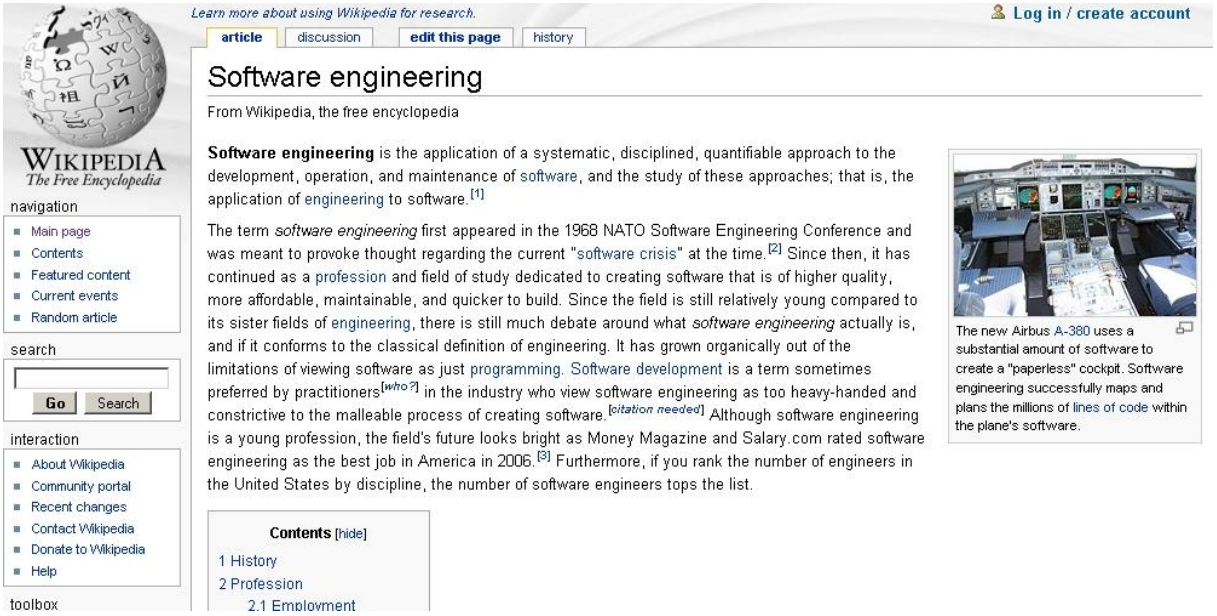


Figure 7-3 Wikipedia.org

The Wiki-system is accessed through a web browser. There a user can create new articles and edit existing ones. All the articles in the Wiki can link to each other using a given naming convention. To format text it uses a simple tagging system, which is not compatible to HTML, to remain as simple as possible.

The Wiki system (based on an open source framework), that had to be evaluated by the participants, had the following features and a guide describing them was handed out prior at the beginning of the experiment (taken from the experiment description):

- All the users have full editing privileges to any page they want to edit. A user can click "edit" link on the desired page to change the text on that page using an editing form.
- The Wiki system follows a naming convention for a page name. The naming convention is: capitalized words joined together. For example: APageName, SoftwareEngineering, IssuesOnDesignQuality etc.
- The Wiki system formats and links the text content, according to the simple structured text formatting rules.
- When the Wiki system finds a text content following its naming convention, it makes that text a hyperlink for a target page. If the target page does not exist, a question mark is placed at the end of the hyperlink to remind the user that this page needs to be created.

- The Wiki system keeps the log of changes made in a page. This log can be displayed in unix diff format.
- Users can subscribe to the page change through email.
- The Wiki system allows the users to make certain changes in the environment through an option setting page. E.g. name, email, time zone etc.

The students were asked to imagine that they are a regular Wiki user that manages and shares his content using this system. They should think about changes they would like to see over the next three years. These changes were documented as change scenarios by them.

7.3.2 Questionaires and other sheets

To get all the information needed from the participants various questionnaires were handed out during the experiment.

Experience Questionaire

The Experience Questionaire was handed out at the beginning of the experiment. It was meant to get a picture of the experience level of every participant. This was important to classify the knowledge and experience level of the participants. On this sheet there were questions related to software engineering, software projects, architecture evaluation, experience in this areas and so on. The goal was to have the ability to compare the experience level of every student with the number and quality of scenarios they have found.

Feedback Individual Brainstorming

The Feedback Individual-Brainstorming Questionaire was handed out after the first individual phase (Livenet) to every student. It contained questions about the organization of the individual phase, like if there was enough time, if the instructions handed out were understandable and helpful and so on. This questionaire was meant to give us feedback what might have gone wrong beforehand and what we should consider the next time such an experiment is planned. This Feedback-Individual-Brainstorming Questionaire was not handed out after the second individual phase, because the questions would not have made much sense again after the second individual phase.

Feedback Team-Meeting

There were two different Team-Meeting Feedback Questionaires: One for the face to face meeting teams and one for the tool-support meeting teams. Some questions of these two questionaires were the same and some were different, focusing on the meeting style of the team.

In this questionaire the focus was on how well the team-members could communicate with each other. In the face to face sheet the questions were based on direct communication and in the tool-support sheet some more questions about the communication with the chat-application were asked.

This questionaire was handed out to every team twice, after every team-meeting – not like the individual feedback sheet, which was only handed out once. That's because in every team-meeting the way the team-members were communication with each-other was different. A group could have communicated very well while sitting together, talking face to face, but may have had big problems communicating over a chat application. To evaluate that, two team-meeting questionaires were handed out.

Final Questionaire Overall

The Final Questionaire was handed out at the end of the experiment, meaning after the second team-meeting. This questionaire was focusing on what meeting-style the participants thought they were better in, where the group has found more and better scenarios, where the group was better working together and so on.

This Final Questionaire was handed out to every single student and was meant to help us compare the actual results of every team-meeting style with the self-assessment of every student.

Data-Capturing Sheet

On the Data-Capturing Sheets the students had to write down the scenarios they found. Every scenario found had to be classified by the students in the terms of importance and likelihood. Importance means how important the scenario is and likelihood means with which probability the scenario might occur. The students could rate both from A to C, A meaning very important or very likely, and C meaning, not important or unlikely. There have been different Data-Capturing Sheets for the individual phase and the team-phase.

7.3.3 Guiding Material

We also handed out various guiding material to the participants which should help them getting an overview of the task.

Application guideline

In the application guideline we handed out, the actual application for the session was explained in detail. The participants got an overview of what the software does, what it is able to do at the moment and what their task will be in the upcoming phase.

Individual / Team-meeting guidelines

We also handed out guidelines for individual brainstorming in the individual phase separated into guidelines for participants who used change categories and those who did not. In the team phase we handed out team meeting guidelines to the teams. Those guidelines were separated into face to face and tool-support guidelines. These guidelines gave the participants information about what to do in the actual phase and how they should proceed. Information how to fill out various questionnaires was included too, so the participants should not have any problems filling out the sheets correctly in time. The team guidelines also included the description of the three roles of moderator, scenario scriber and time keeper as can be seen in Table 7-5.

With the guidelines handed out all necessary elements have been explained, so the participants knew what to do and could check back every time in the phase running without waiting for a tutor coming to them for help.

7.3.4 Categories

The categories we provided to one half of the participants should help them to find – maybe "better" - scenarios. With these sets of categories the participants should try to find scenarios for each category. By knowing that scenarios have to be found for the user interface, for performance changes, etc. it may be easier for the participants to find architecture scenarios.

Livenet	Wiki
1. User interface changes (UI)	1. User interface changes (UI)
2. Security policy changes (SP)	2. Security policy changes (SP)
3. Performance changes (PC)	3. Performance changes (PC)
4. Communication channels and/or mechanism changes (CO)	4. Notification policy changes (NP)
5. Workflow features changes (WF)	5. Content editing rules changes (CE)
6. Content management requirements changes (CM)	6. Meta data related changes (MD)

Table 7-4 Change categories

7.4 Phases

The participants had to find architecture scenarios in an individual and a team phase. In the individual phase they had to find scenarios on their own and in the team phase the participants discussed their scenarios in groups of three people.

7.4.1 Individual Phase

In the individual phase every participant had to find architecture scenarios on his own. In the beginning of this phase a guideline was handed out to each student, with instructions what they had to do. Also a description of the application (Livenet or Wiki) was handed out. And to those students who were using categories as support, the category-set was handed out too.

Then, based on the requirements specification, the participants had to brainstorm a number of functions for the current application and possible future functionality. They also had to consider various roles, e.g., participants team leaders, management, etc. which should have helped them to find scenarios. The task was to think about the changes that would be required in the web-application currently as well as within the next three years of its existence.

7.4.2 Team-Phase

After the individual phase the participants came together in groups of three people. Every group either consists of students using categories or of students not using them. Every team-member had a role within the team:

Role	Responsibilities	Desirable Characteristics
Moderator	Facilitates generation of scenarios; keeps the group focused on the task; makes the group following the process	Good facilitation skills, good observer, able to intervene when discussion is pointless, feels comfortable interacting with people
Scenario Scriber	Writes scenarios during scenario generation process. Carefully captures agreed wording of each scenario and doesn't allow moving to the next scenario until exact wording of the accepted scenario is written down.	Reasonable speed to write or type. Willingness to be a stickler about not moving on before one scenarios is carefully captured
Timekeeper	Helps the team manage the allocated time. Give a polite warning if a discussion on one scenario goes on more than 3 minutes.	Willingness to interrupt prolonged discussion. Doesn't want to keep his/her nice image at the cost of group's time.

Table 7-5 Roles in team phase

In the team-phase, there were also two different meeting-styles: Face to face and tool-support.

Face To Face

Groups that were communication face to face were sitting together physically in the lecture room. They were discussing the scenarios they had found as individuals. For each scenario they had to decide whether it is "good" and should be written on the Team-Capturing-Sheet or if it should be left out.

Tool-Support

The tool-support groups did not sit together physically face to face. They were sitting in an informatics-room – every student in front of a computer on his own. They were not allowed to talk to each other, but only to communicate with their team-members over a chat-application. The task was the same as for the face to face groups: Discussing the scenarios found in the individual phase to take them onto the group-scenario-list or to left them out.

7.5 Setup in detail and execution

The experiment was held on two days, to give all students interested the chance to participate. On both days the experiment was held exactly the same way. The procedure of the controlled experiment can be seen in Figure 7-4.

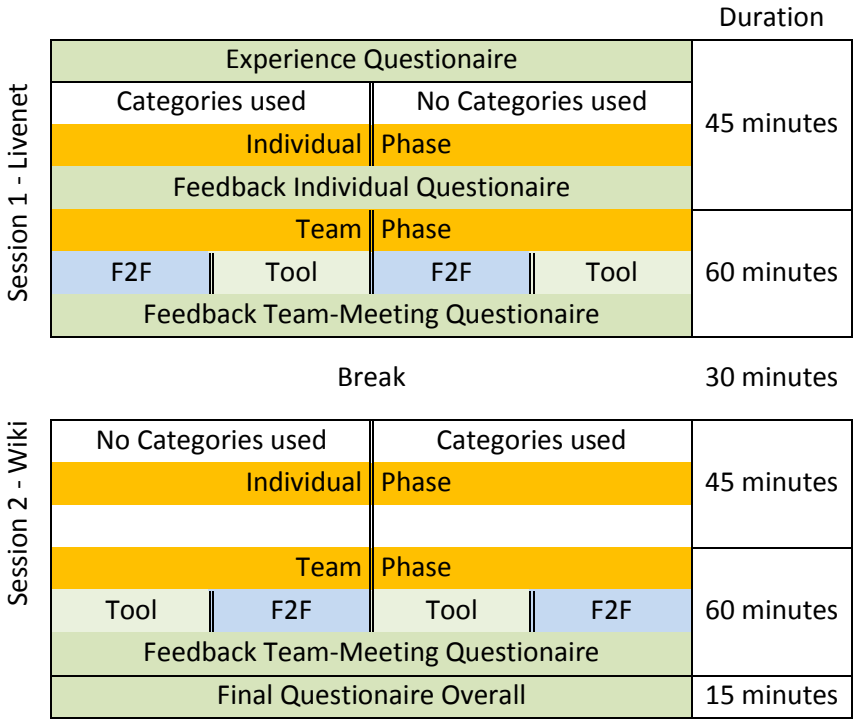


Figure 7-4 Experiment design in detail

7.5.1 Session 1 – Livenet

After everyone was in assigned to a group every student got information material on what to do in the upcoming, the individual, phase. They got information on the application they had to find scenarios for, which has been Livenet in session 1, and which change categories to use, if they were assigned to do so in session 1. At the beginning every student had to fill out the Experience Questionaire which took them a few minutes. After that they started to find scenarios for Livenet – some used categories, some did not – and wrote them down on the Data Capturing Sheet. After 45 minutes the individual phase ended and the students had to fill out a Feedback Individual Questionaire. Then the Experience Questionaire, the Individual Feedback Questionaire and the information material were collected. The students only kept their Data Capturing Sheets with the scenarios they have found, because they needed them in the upcoming team-phase.

Then groups who used tool-support communication in session 1 went to the informatics lab and the groups that were communication face to face stayed in the room. Again, the information material necessary was handed out to the groups. Then every team had the task to discuss all scenarios they had found in the individual phase and decide whether to write them down onto the Team Data Capturing Sheet or not. The face to face groups were sitting together physically discussing their scenarios. The tool-support teams did not sit together in the informatics lab. They were communication over a chat application and or forum as they'd liked to. They also had to discuss their individual scenarios and decide to put them on the team sheet or not. At the end of the team-phase, which lasts 60 minutes, every team had to fill out a Team Feedback Questionaire. Finally every material was collected from the students.

After that session 1 ended and there was a 30 minute break.

7.5.2 Session 2 – Wiki

In session 2 the teams switched in terms of categories and meeting-style. Session 2 was executed nearly like session 1, but with some adaptions:

- No second Experience Questionaire was handed out.
- No second Individual Feedback Questionaire was handed out, because the second individual phase was executed as the first one, only with the difference that the system in focus has been a Wiki-system and not Livenet. A second Team Experience Questionaire was handed out tough, because the second team phase was different to the first one, because the groups had to use another communication technique.
- In the end a Final Questionaire was handed out.

7.6 Experiment Validity

In this section we discuss the threats to internal and external validity that every empirical study has.

The **independent variable** in this study and thesis is the meeting-style of the teams during the team-phase of the experiment. The teams did either face to face communication sitting together physically or tool-supported communication not sitting together physically.

The **dependent variable** of this study and thesis is the quality of the scenarios found by the participants in the individual and team-phase.

7.6.1 Threats to internal validity

Internal validity describes how well the dependant variable can be referable to the experimental variables. To ensure that the composition of the different groups is as randomly as possible and the participants do know as less group-members as possible a list of possible groups – distinguishing between face to face and tool-support groups and groups that use categories and that do not – was created beforehand. At the beginning of the first individual phase, one student after another was assigned to either “categories used” or “categories not used” for scenario evaluation – this ensured that participants sitting together and knowing each other did not come together in the same group, because one used categories and one did not and groups also were separated considering this: “Categories used”-participants were grouped to face to face and tool-support teams and also the “categories not used”-participants were grouped to face to face and tool-support teams.

Also the method of creating a reference-scenario set is a possibility where the internal validity can be violated. Though this method has been used by Babar and Biffl [22] and Bengtsson [20], it still may be a threat. But we think that this method will not affect the results of the experiment.

A possible threat may be the team that created the reference-scenario set. All three members of the this team have been in the last term of their study in business informatics, dealt with the topic over a few months, have prepared and executed the experiment and collected all the data. So there is a very little possibility that this internal validity has effect on the results.

7.6.2 Threats to external validity

The external validity stands for the ability to compare the results to other situations, which means how general the results can be interpreted.

The major problem of the external validity may be the participants as representatives of “real” stakeholders participating in software architecture evaluation. Stakeholders usually have more experience in software development. The participants of our study had different levels of experience and as good as no experience in architecture evaluation – but stakeholders in companies might not have experience in architecture evaluation too. Before the experiment, a two-hour introduction on architecture evaluation, scenarios brainstorming, etc. was given to the students, so they knew what to do. They also had some information material one week before the study. Stakeholders may have

different levels of experience too and this in different topics. There are stakeholders who have a technical background, a project-management background, etc. Our participants can mostly be compared to stakeholders with technical background, because of their technical study at the Vienna University of Technology.

Also, stakeholders usually know the application, or the surrounding the application is build for, so they have another approach to it than people who do not know the application at all. Our participants did not know the application Livenet beforehand, but they might have known the Wiki System better, because of their regular use of Wikipedia.org. So there might be differences in the results of Livenet and the Wiki System.

Finally it can be said, that the process of finding scenarios for software architecture evaluation might be supported by more information on the application in focus and so the participants might be “nearer” to the application than our participants.

8 Results

In this chapter we will present results from our controlled experiment. In this experiment 54 students had to evaluate software architectures of two different systems. They had to do this individually and in teams.

8.1 TopRated-Scenarios reference profile

To analyze the results of the controlled experiment, the quality of a found scenario has to be classified to say if it is useable or not. A method, used by Bengtsson [20] and Babar and Biffl [22], is to look at every scenario profile of all individuals and teams. If a scenario is found by a participant or team, they think that it is important somehow. If this scenario now has been found by all participants it can be considered to be the most important scenario of all – the more it appears in different scenario profiles, the more important it will be. That's how a reference scenario profile is created (there are other methods to create such reference profiles too). Based on this reference scenario profile that we called "TopRated-Scenarios" further evaluation can be done. Scenario-profiles of individuals and / or teams can be compared with each other in reference to our TopRated-Scenarios.

After we have collected all scenarios found by individuals and teams, we had a look at how often a scenario was found by individuals and teams. If an individual has found a certain scenario, it got 1 point. If a certain scenario was found by a team, it got 2 points. We decided to give away 2 points for a scenario when found by teams because of the following fact: Every scenario that is found in a team is discussed by the team-members. Only if the team as a whole agrees to the scenario it is written down into the team's scenario-profile. So, if a scenarios appears in a team list it is "more valuable" as if it appeared in an individual list "only". That's why we decided to rate such scenarios with 2 points.

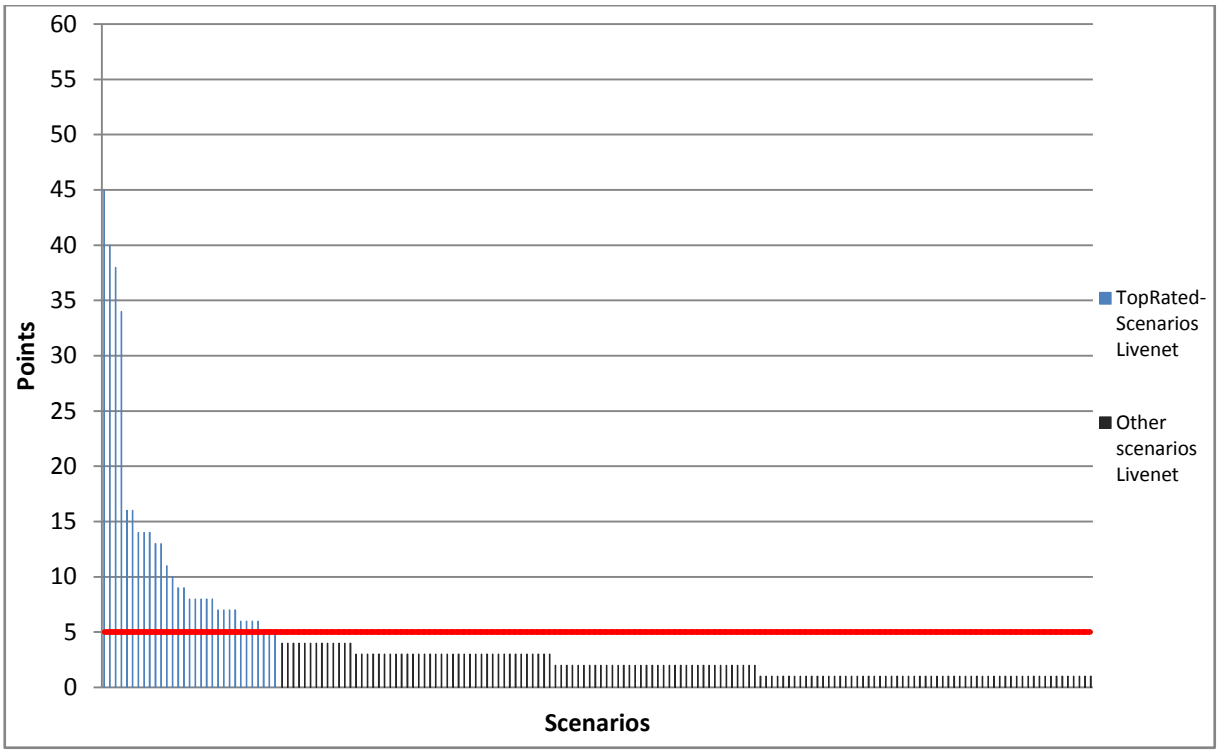


Figure 8-1 TopRated-Scenarios Livenet

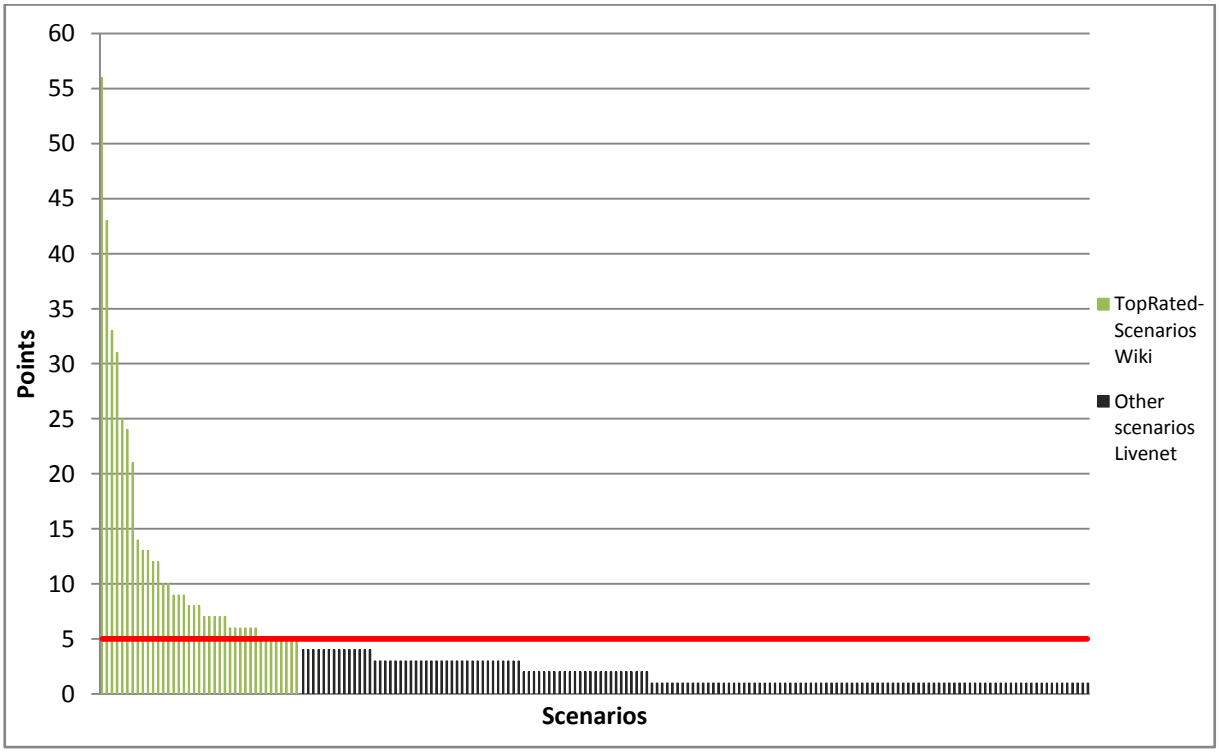


Figure 8-2 TopRated-Scenarios Wiki

In Figure 8-1 and 8-2 you can see the scenarios found for every application – Livenet and Wiki – and the number of points every scenario got. As you can see in both diagrams 2/3 of the scenarios only

have been found one, two or three times, which is not very often and not useable for empirical investigation.

So we decided to use only scenarios that have been found more often. Based on the 80:20 rule we created two lists of TopRated scenarios – one for Livenet and one for Wiki. The 80:20 rule is a very popular rule of thumb. As in the code development area 80% of failures can be found in only 20% of code, we claim that 20% of the scenarios cover 80% of all changes that will occur in the future. You can sum up the general conclusion of the 80:20 rule as: „focus your efforts on the high pay-offs“ [20], so we decided to focus on the rounded first 20 % of scenarios in our lists.

For Livenet 174 different scenarios have been found in total. For the Wiki System 193 different scenarios have been found in total. For Livenet the best 31 scenarios have been used for the TopRated list, which are 17.82 % of all scenarios found for Livenet. For the Wiki System the best 39 scenarios have been used for the TopRated list, which are 20.21 % of all scenarios found for the Wiki System. As can be seen in the tables above all scenarios that have at least 5 points are included in the TopRated lists. We chose 5 points as limit, because it is as close at 20 % as possible. In the Appendix you can see the complete list of TopRated-Scenarios for Livenet and Wiki, including their score and description.

8.2 Individual and team experience

Every participant of our controlled experiment had to fill out an Experience Questionaire. On this sheet there were questions related to software engineering, software projects, architecture evaluation and so on. The goal was to have the ability to compare the experience level of every student with the number and quality of scenarios they have found.

By filling out the Experience Questionaire every participant had the ability to reach 0 to 4 points, 0 meaning the minimum and 4 the maximum.

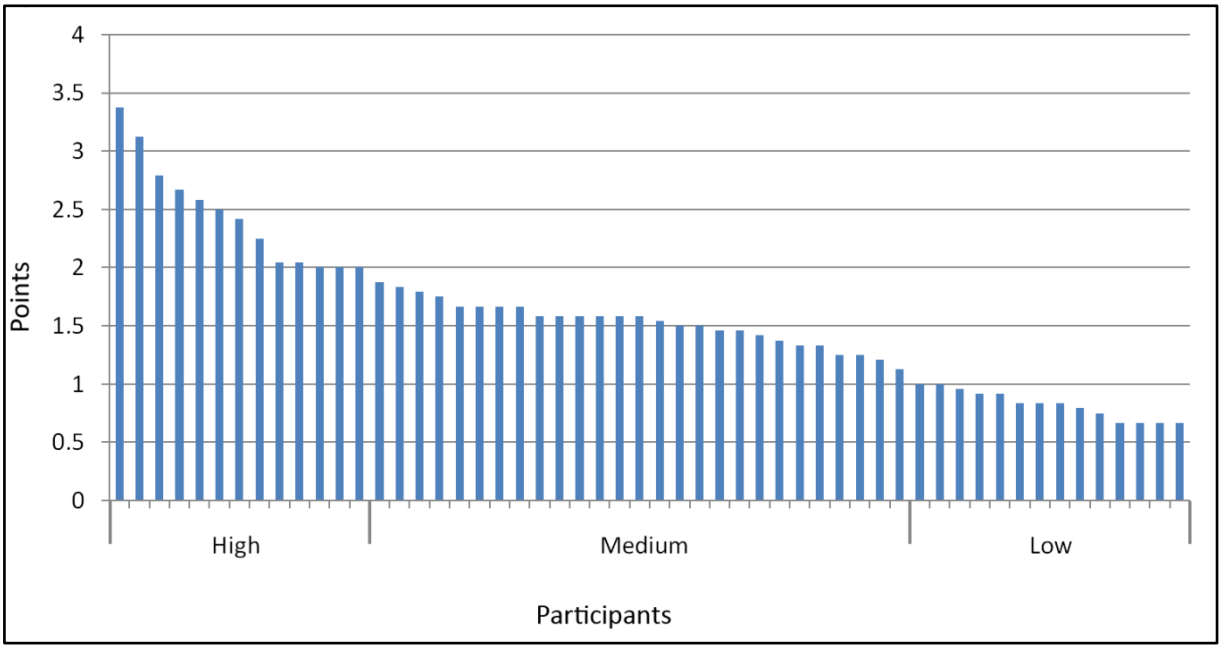


Figure 8-3 Experience of Participants

We defined three experience levels – low, medium and high – in which the participants should have been divided into. We tried to use the 80:20 rule again and had a look at Figure 8-3. There it can be seen that there is a break between participant 13 and 14. Participant 13 has 2 points where participant 14 has significantly less than 2 points. We thought that this would be a good boundary between high and medium and that's why we set it there. It's 24 % but not 20 % of all participants, but the break between these two participants is that clear that we decided to set the boundary there. On the other side we chose to set the boundary at 1 point. This means that 14 participants have low experience – that's 26 %. This leaves exactly 50% of the participants with the experience level medium.

Percent	Level	Points	Number of Participants
0-24%	Low	0 – 1	13
24%-74%	medium	1 – 2	27
74%-100%	High	2 – 4	14

Table 8-1 Experience levels

As can be seen in the table above the majority of the participants has medium level experience according to our Experience Questionaire.

In this thesis we focus on evaluating team-results by comparing results from teams communicating face to face and teams with tool-supported communication doing architecture evaluation. We have to check the experience levels not only of individual participants but of teams also. If there should be a huge difference between the team's experience, we have to evaluate our results in three different

experience levels individually. If the experience levels of all teams are nearly equal, we do not have to separate the evaluation and can have a look at the results for all teams at once. But we will have a look if experience of participants / teams has an influence on the number of good scenarios found, compared to the experience points of the teams later.

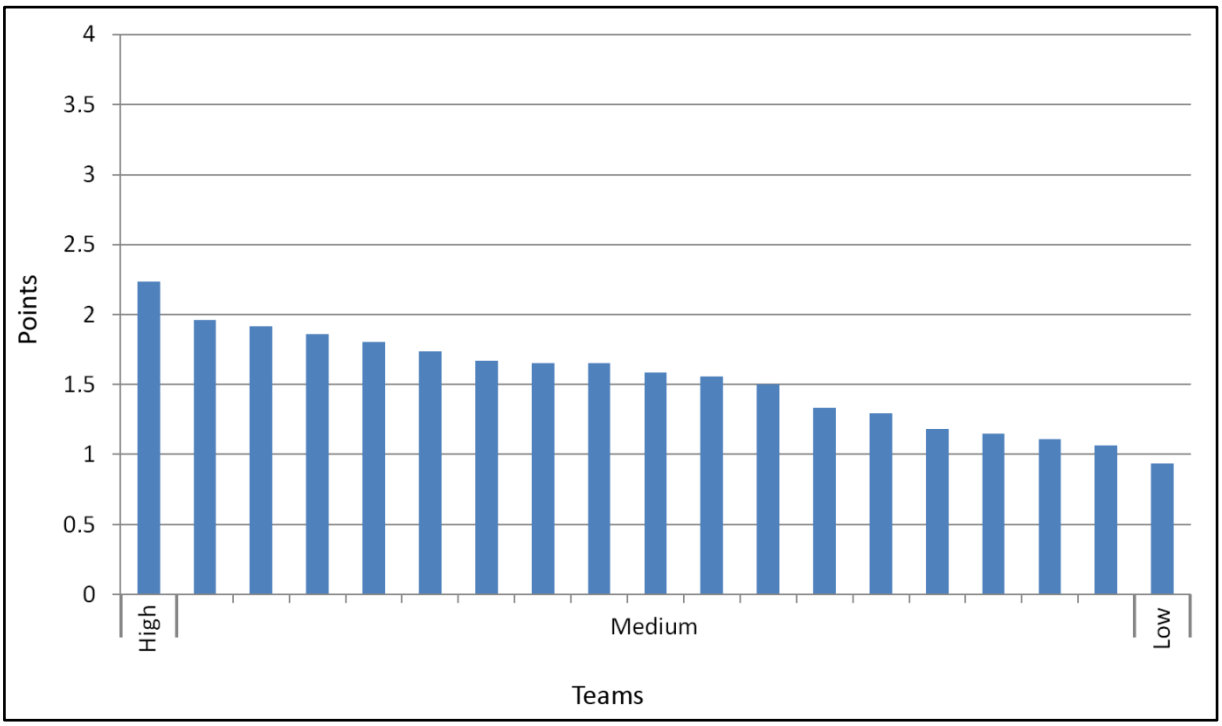


Figure 8-4 Experience of Teams

As can be seen in Figure 8-4 the bandwidth of the team-experience is not as broad as of the individual participants.

Team	Points	Level
sa_g21	2.24	high
sa_g06	1.96	medium
sa_g15	1.92	medium
sa_g10	1.86	medium
sa_g11	1.81	medium
sa_g17	1.74	medium
sa_g09	1.67	medium
sa_g16	1.65	medium
sa_g22	1.65	medium
sa_g34	1.58	medium
sa_g33	1.56	medium
sa_g05	1.50	medium
sa_g24	1.33	medium
sa_g08	1.29	medium
sa_g32	1.18	medium
sa_g31	1.15	medium
sa_g23	1.11	medium
sa_g12	1.06	medium
sa_g01	0.94	low

Table 8-2 Team Experience Levels

According to our definition of the 3 experience levels – low, medium and high – most of the teams do have medium experience, but one with low and one with high experience. The one team with low experience has 0.94 points, where the boundary to the medium level is 1. The one team with high experience has 2.24 points, where the boundary between medium and high is 2. So we will not handle these two teams separately, but evaluate all teams as a whole, because when looking at Figure 8-4 again we see that the experience level of all teams is very balanced.

Seeing that the teams all have nearly the same experience shows that we did the team-building process very well. The individual participants have been grouped together randomly, so that we do not have teams with very high experience or teams with very low experience. This also makes it easier to do evaluation. We can now include all teams in one evaluation process and do not have to split evaluations into three experience levels. We will have a look on the influence of experience on finding evaluation scenarios though, but we can compare all teams much better with nearly the same experience level.

8.3 Comparing different methods of team-meetings

The first question we want to evaluate is, if teams that communicate face to face find more important scenarios than teams communicating with some kind of tool-supported communication. In our controlled experiment all participants had to find scenarios as individuals first. In the second phase - the team phase – the participants had to discuss their scenarios in groups. In this team phase one half of the groups was sitting together physically in the same room (the so called face to face teams) and the other part was not sitting together physically – participants of a so called tool-support team were sitting in front of a computer, communicating with their team members over a chat-application.

8.3.1 TopRated-Scenarios and concrete findings by teams

First we will have a look at how often the TopRated-Scenarios have been found by teams with separated results for face to face teams (F2F) and toolsupport-communicating teams (TS).

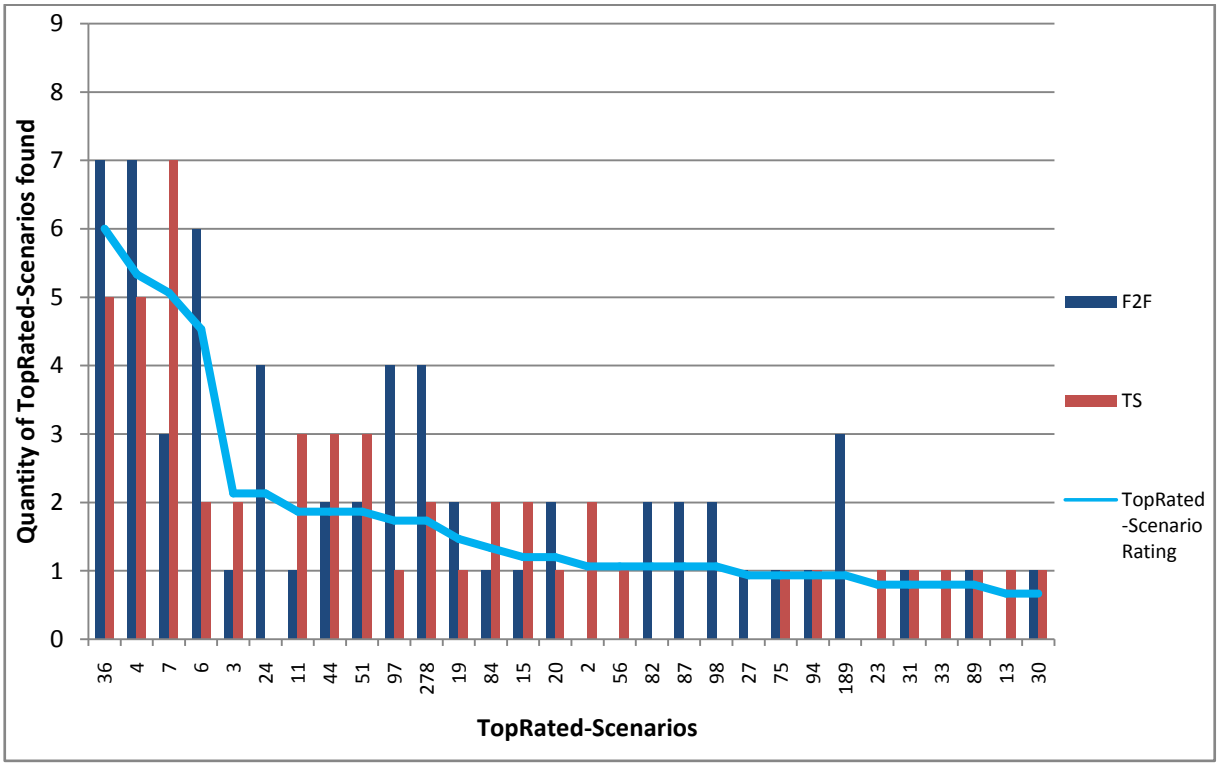


Figure 8-5 Quantity of TopRated-Scenarios found by teams - Livenet

In Figure 8-5 the TopRated-Scenarios for Livenet are ordered by their score. The scale of the TopRated-Scenario Rating has been adapted for this chart, so the relationship between the rating and the actual findings of the teams can be seen. It can be seen that the number of scenarios found by teams does not always follow the TopRated-Scenario rating. There are also different results between F2F and TS teams. Looking at the best rated scenarios in the TopRated list, the majority has been found more often by F2F than TS teams. Some TopRated-Scenarios on the other hand have been found more often by TS than F2F teams. Some scenarios have not been found very often at all

by teams, though they are high rated in the TopRated-Scenario profile. That means that these scenarios must have been found more often by individuals than teams, otherwise they would not have been rated so high in the TopRated-Scenario profile.

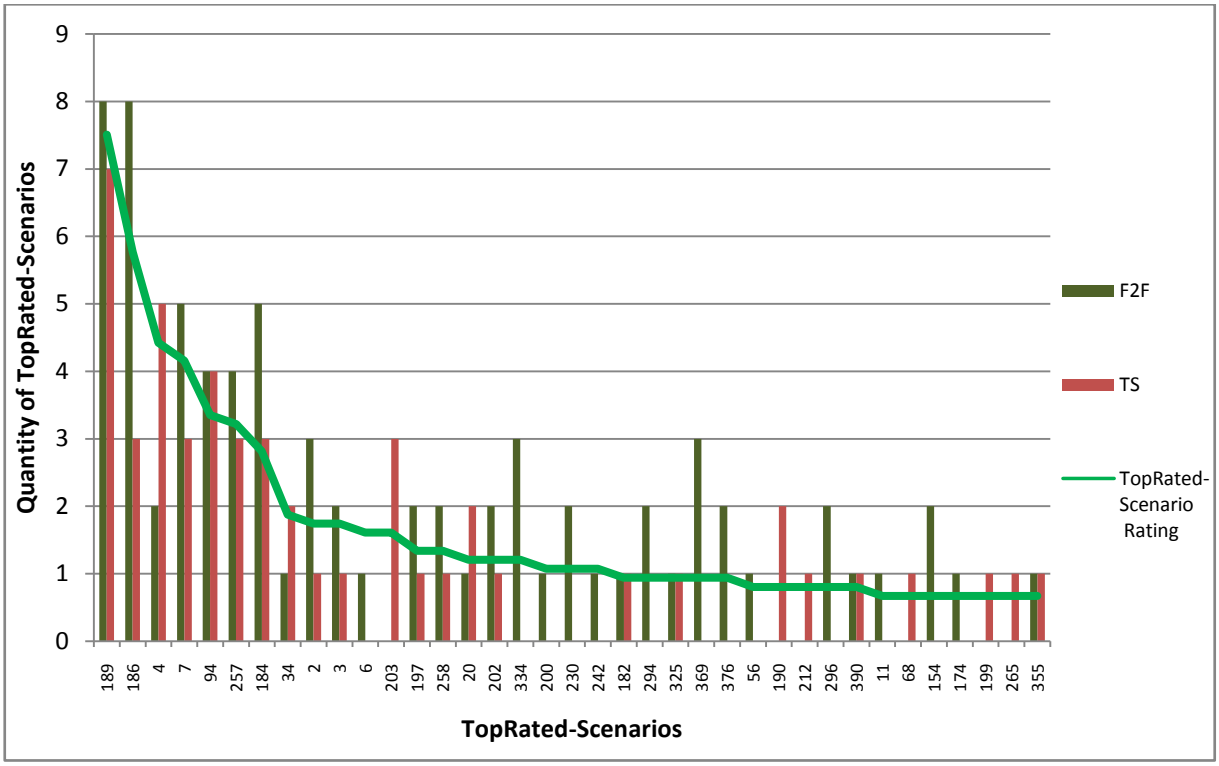


Figure 8-6 Quantity of TopRated-Scenarios found by teams - Wiki

In Figure 8-6 the TopRated-Scenarios for Wiki, which are ordered by score, can be seen and how often they have been found by F2F and TS teams. The scale of the TopRated-Scenario Rating has been adapted for this chart too, so the relationship between the rating and the actual findings of the teams can be seen. Like in Figure 8-5 for Livenet the order of the TopRated-Scenarios does not fit perfectly to the actual number of findings of the scenarios by the teams. What can be seen here when looking at the best rated TopRated-Scenarios is that F2F teams have found them more often than TS teams. Some TopRated-Scenarios have not been found by F2F or TS teams at all. These scenarios must have been found more often by individuals, otherwise they would not have been rated that high in the TopRated-Scenario profile. A more detailed look on the difference between individuals and teams will follow later. These two charts, Figure 8-5 and Figure 8-6 should give you a quick overview of the TopRated-Scenario profile and how often they have been found by F2F and TS teams for Livenet and the Wiki System.

8.3.2 TopRated-Scenarios found by teams for Livenet

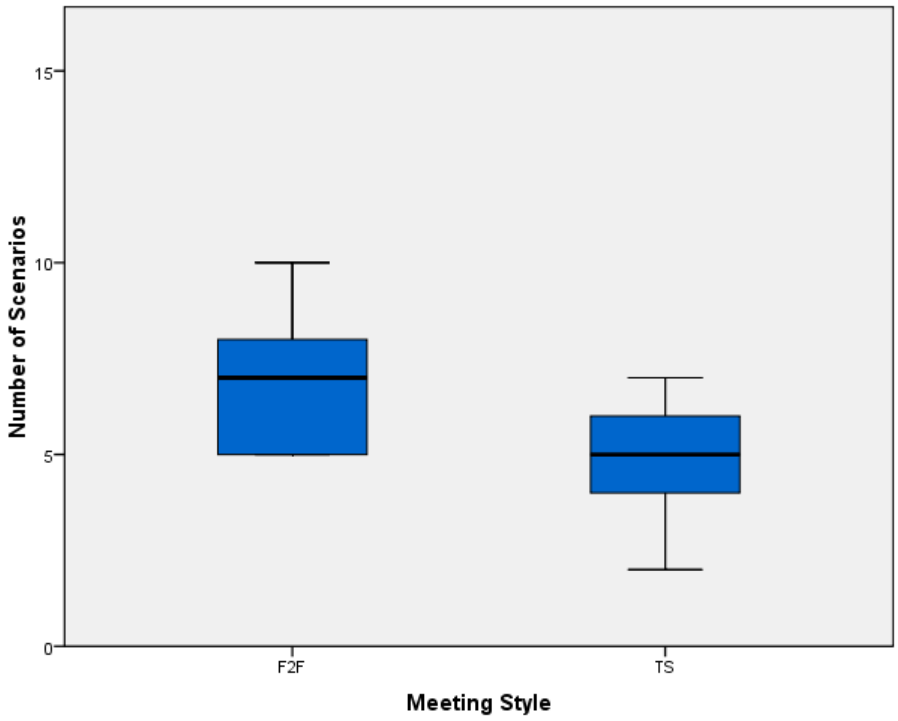


Figure 8-7 Boxplot of TopRated-Scenarios found by teams - Livenet

Figure 8-7 shows a boxplot of how many TopRated-Scenarios have been found by teams in Livenet. The results are divided into face to face (F2F) and toolsupport-communicating (TS) teams. As can be seen F2F teams found more scenarios then TS teams.

	Meeting Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. interval)
Number of Scenarios	F2F	9	6.89	1.97	0.035(s)
	TS	10	5.00	1.63	

Table 8-3 TopRated-Scenarios found by teams - Livenet

Table 8-3 shows, that there have been 9 F2F teams and 10 TS teams. The F2F teams did find 6.89 TopRated-Scenarios on average and the TS teams did find 5 scenarios on average for Livenet. When looking at the t-test with a 95% confidence interval we see it scored with 0.035(s) which is below 0.05. In this case the null hypothesis could be rejected and therefore F2F teams do find better scenarios then TS teams for Livenet.

8.3.3 TopRated-Scenarios found by teams for Wiki

But how do the results of the Wiki System compare to Livenet? When looking at Figure 8-8 we see a boxplot with the number of TopRated-Scenarios found by teams for Wiki.

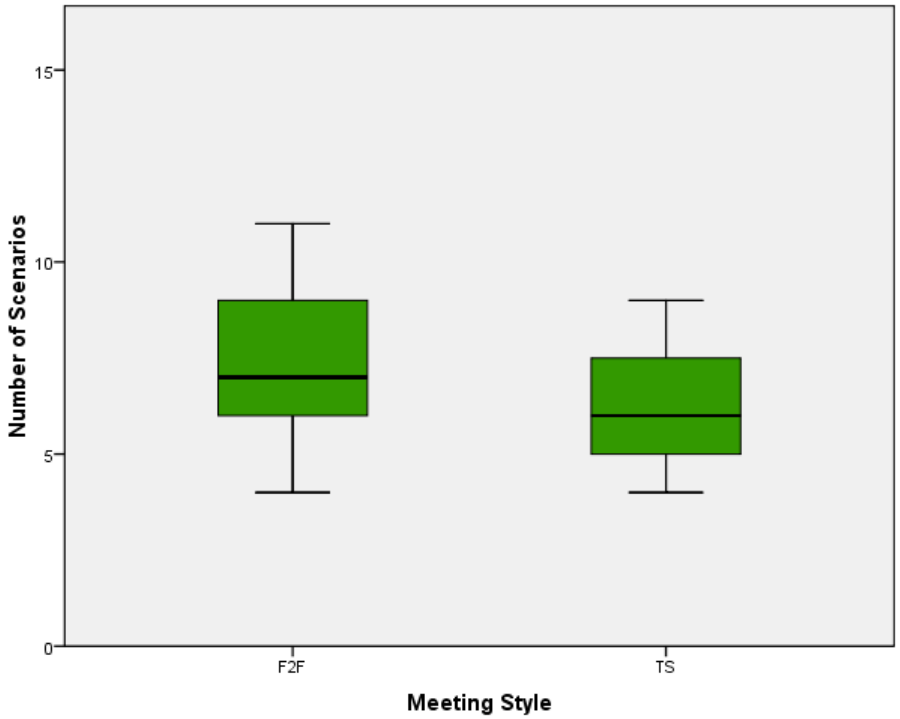


Figure 8-8 Boxplot of TopRated-Scenarios found by teams - Wiki

The results are also separated into F2F and TS teams. When compared to the Livenet results, the results of the Wiki System are not that clear. Table 8-4 shows that there have been 10 F2F teams and 8 TS teams. The F2F teams have found 7.50 TopRated-Scenarios on average and the TS teams have found 6.25 scenarios on average. This means that both team-meeting styles have found more scenarios on average for the Wiki System than for Livenet, which could be because the participants have gained experience in the session 1 – Livenet – and now have been better in session 2.

	Meeting Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. Interval)
Number of Scenarios	F2F	10	7.50	2.32	0.203(-)
	TS	8	6.25	1.67	

Table 8-4 TopRated-Scenarios found by teams - Wiki

What is for sure is that there is not that much difference between the results of F2F and TS teams at the Wiki System as has been at Livenet. At Livenet in session 1 there has been a difference of 1.89 but at the Wiki System in session 2 the difference is only 1.25.

The t-test with a 95% confidence interval in Table 8-4 shows that the results are not as clear as they have been at Livenet. The result is 0.203(-) which is more than 0.05 and therefore the null hypothesis cannot be rejected. In the case of the Wiki System it cannot clearly be said that F2F teams do find better scenarios than TS teams. They do find slightly more in number, but the t-test does not support this statement.

8.3.4 TopRated-Scenarios with change categories considered for Livenet

Grouping the participants into F2F and TS teams in the team-phase was not the only separation we did. In the beginning of the controlled experiment the participants were also separated whether they had to use change categories to find scenarios or not. Those change categories should help the participants to find scenarios. Such categories could be: User-Interface changes, Security-policy changes, Performance & scalability changes, etc. The idea was to find out which method would have better results for architecture evaluation – should participants be guided, so they can focus on what scenarios they have to find, or not, letting them find scenarios freely. The set of categories was carefully chosen of course so they covered the topics of the software evaluated. Babar and Biffl [22] did an evaluation about this topic and the result was that participants using change categories do have better results in software architecture evaluation than participants without them. So we will have a look at our results, divided into “categories used” and “categories not used” to see if there is a difference.

First we will have a look at the Livenet results. In the following boxplot (Figure 8-9) and tables (Tables 8-5 and 8-6) we will have a look at the results with the F2F and TS teams separated into teams that used change categories and did not use change categories.

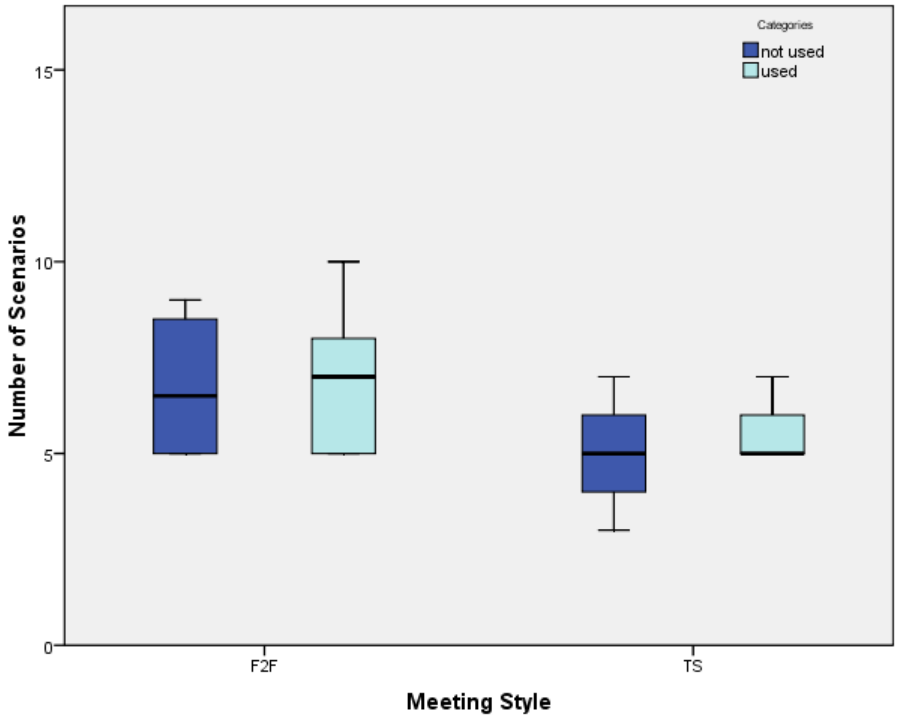


Figure 8-9 Boxplot of TopRated-Scenarios found by teams with categories separation - Livenet

	Meeting Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. Interval)
Number of Scenarios	F2F	4	6.75	2.06	0.191(-)
	TS	5	5.00	1.58	

a. Categories = not used

Table 8-5 TopRated-Scenarios found by teams with categories "not used" - Livenet

	Meeting Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. Interval)
Number of Scenarios	F2F	5	7.00	2.12	0.153(-)
	TS	5	5.00	1.87	

a. Categories = used

Table 8-6 TopRated-Scenarios found by teams with categories "used" - Livenet

When we look at the results in Table 8-5 and Table 8-6 we see that teams using change categories did a little better altogether. The mean of F2F teams is 7 and the mean of TS teams is 5 using categories. F2F teams not using categories have a mean of 6.75 where TS teams also have 5. So this seems like teams using categories are a little better than teams without, but what can be said is that F2F teams have better results overall, whether they use categories or not. Now we will have a look at the t-tests.

The t-tests in Table 8-5 (results for categories “not used”) and Table 8-6 (results for categories “used”) show that it cannot be said clearly that F2F teams are better than TS teams. The result of the t-test with a 95% confidence interval are 0.191(-) for categories “not used” and 0.153(-) for categories “used”. Though the results for categories “used” are better, they are still way over 0.05 which means that the null hypothesis could not be rejected with this results. When looking at the combined results, meaning not separating between categories “used” and “not used” as seen in chapter 8.3.2, a clear statement can be made. That may be the case because there are not enough teams to do such detailed evaluation. When separating in F2F and TS teams and then separating these groups in terms of change categories again, the particular statistic groups evaluated do not have enough elements to do meaningful statements. As can be seen in Table 8-5 and Table 8-6 there are only 4 or 5 teams in every group evaluated, which is not enough to do clear statements.

8.3.5 TopRated-Scenarios with change categories considered for Wiki

Let‘s have a look at the results of the Wiki System when we will separate the F2F and TS teams into categories „used“ and „not used“ as we did above for Livenet.

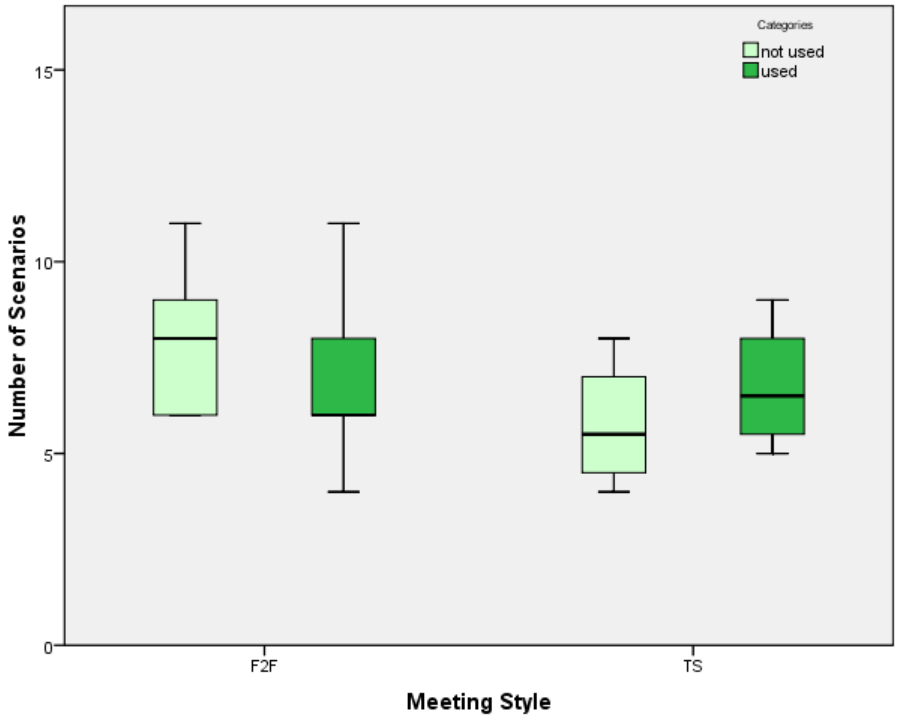


Figure 8-10 Boxplot of TopRated-Scenarios found by teams with categories separation - Wiki

	Meeting Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. Interval)
Number of Scenarios	F2F	5	8.00	2.12	0.121(-)
	TS	4	5.75	1.71	

a. Categories = not used

Table 8-7 TopRated-Scenarios found by teams with categories "not used" - Wiki

	Meeting Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. Interval)
Number of Scenarios	F2F	5	7.00	2.65	0.869(-)
	TS	4	6.75	1.71	

a. Categories = used

Table 8-8 TopRated-Scenarios found by teams with categories "used" - Wiki

Compared to the results from Livenet (chapter 8.3.4) the results from the Wiki System are not that clear. F2F teams are still better than TS teams, very clearly when categories have not been used. When categories have been used there is nearly no difference between F2F and TS teams. What is very interesting is that this time results of teams in terms of change categories are not as clear as they have been at Livenet. TS teams that used categories are better than teams without, but F2F teams using change categories are worse than teams without. Now we will have a look at the t-tests for the Wiki System.

When looking at the t-tests in Table 8-7 and Table 8-8 the results are very mixed. Table 8-7 shows the t-test (with 95% confidence interval) of teams that did not use change categories. The result is 0.121(-) which is more than 0.05 and therefore not meaningful. But when we look at Table 8-8 which shows the results for teams using change categories, the result is 0.869(-) which is way too high. So we cannot make meaningful statements for the Wiki System either when separating into categories “used” and “not used” teams. Like said before, that may be the case because there are not enough teams to do such detailed evaluation.

8.4 Comparing individuals and teams

The second question we ask is if teams find more important scenarios than individuals. In our controlled experiment everyone had to find scenarios individually first. In the team phase afterwards every scenario should have been discussed to write it down or not. That means that the results in the team-phase should have better results than those in the individual phase, because not useful scenarios should have been left out. So we will have a look at F2F and TS teams to evaluate which form of communication had better results compared to the individual phase. If the scenarios have been discussed well, the amount of more important scenarios in the specific scenario profile has to be higher as if the scenarios have not been discussed well.

8.4.1 TopRated-Scenarios found by individuals and teams for Livenet

At first we will have a look at the results from session 1 where change scenarios for the Livenet application had to be found.

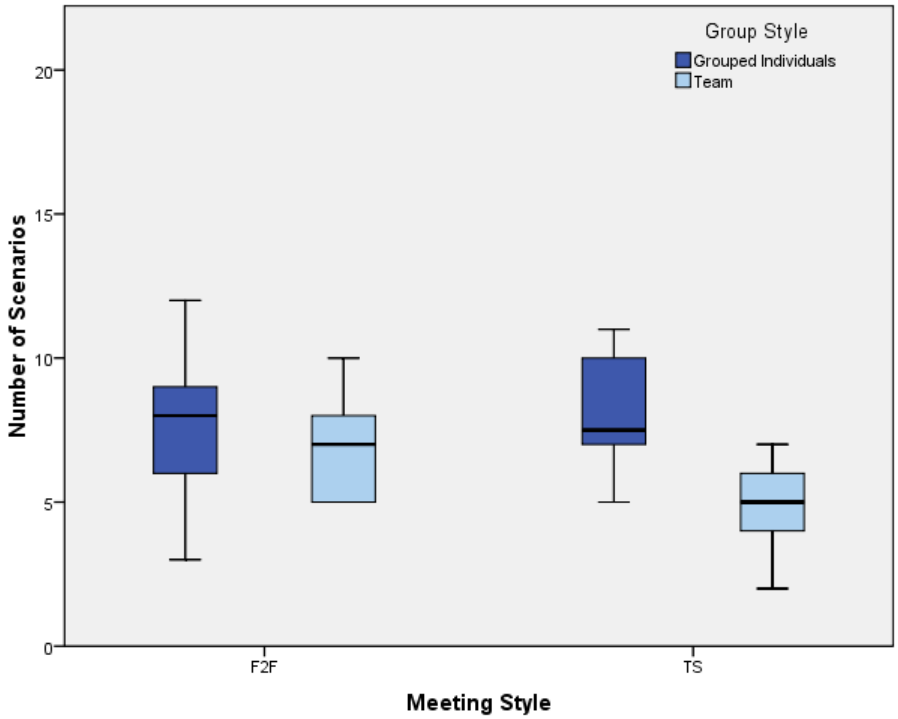


Figure 8-11 Boxplot of TopRated-Scenarios with different group styles for Livenet

In Figure 8-11 we can see the number of TopRated-Scenarios found by F2F and TS teams and the group's participants individually found scenarios, called 'grouped individuals'. 'Grouped individuals' means in other words, the scenarios the team-members of a team had found in the individual phase. This gives us a good overview of how many scenarios have been found in the individual phase of the team members and how many have made it onto the team scenario list.

Looking at the boxplot we see that F2F teams do have little less TopRated-Scenarios found compared to the team-participants individual findings. When looking at the TS teams we see that they have much less scenarios found in the team phase than they had found in the individual phase.

	Meeting Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. Interval)
Number of Scenarios	F2F	9	7.11	2.93	0.451(-)
	TS	10	8	2.06	

a. Group Style = Grouped Individuals

Table 8-9 TopRated-Scenarios found by grouped individuals for Livenet

	Meeting Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. Interval)
Number of Scenarios	F2F	9	6.89	1.97	0.035(s)
	TS	10	5	1.63	

a. Group Style = Team

Table 8-10 TopRated-Scenarios found by teams for Livenet

When looking at the tables in Table 8-9 and Table 8-10 we see that F2F and TS teams both have found more TopRated-Scenarios in their individual phase and less in their team phase. What we see too is that the difference between individual and team phase is not that big with F2F teams, but bigger with TS teams. This is probably because F2F teams have discussed their good scenarios very well, so not many TopRated-Scenarios got lost. Below we will have a comparison of TopRated and all scenarios found by individuals and teams to have a more detailed look at this.

Table 8-9 and Table 8-10 also show the t-tests with a 95% confidence interval. When we look at Table 8-10 we see that the result of 0.035(s) is below 0.05, but when we look at Table 8-9 we see that the result of 0.451(-) is way over 0.05 which means that there is no connection between meeting style (F2F or TS) and number of found scenarios which is clear, because in the individual phase the team-factor was not present.

	Group Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. interval)
Number of Scenarios	Grouped Individuals	19	7.58	2.479	0.027(s)
	Team	19	5.89	2	

Table 8-11 TopRated-Scenarios with group styles for Livenet

Table 8-11 shows results for the 2 different group styles – grouped individuals and teams. As can be seen the participants have found more TopRated-Scenarios individually than in teams, which maybe mean that they did not have enough time in the team phase to discuss all scenarios. The t-test for the 2 different group styles shows that the result of 0.027(s) is below 0.05 which means that there is a connection between the number of scenarios found and the group style, so the number of found scenarios by grouped individuals and teams is not coincidence.

8.4.2 TopRated- and all scenarios found by teams and individuals for Livenet

To see if F2F teams did better scenario discussion in the team phase than TS teams, we need to have a look at how many scenarios the participants and teams have found in general and how many TopRated-Scenarios have been their scenario profiles. By evaluating the percentage of TopRated-Scenarios in the scenario-profile of individuals and teams, we can have a look at how this percentage has changed from the individual to the team phase. If the percentage of TopRated-Scenarios in the scenario profile will be higher in the team phase than in the individual phase, this could mean that scenarios have been discussed well and not useful scenarios have been left out and only the useful have been kept. If the percentage will be the same in the individual phase and team phase this could mean that the scenarios in the participants scenario profile may have been gone through from one scenario to the next without discussion and so no bad scenario has been left out.

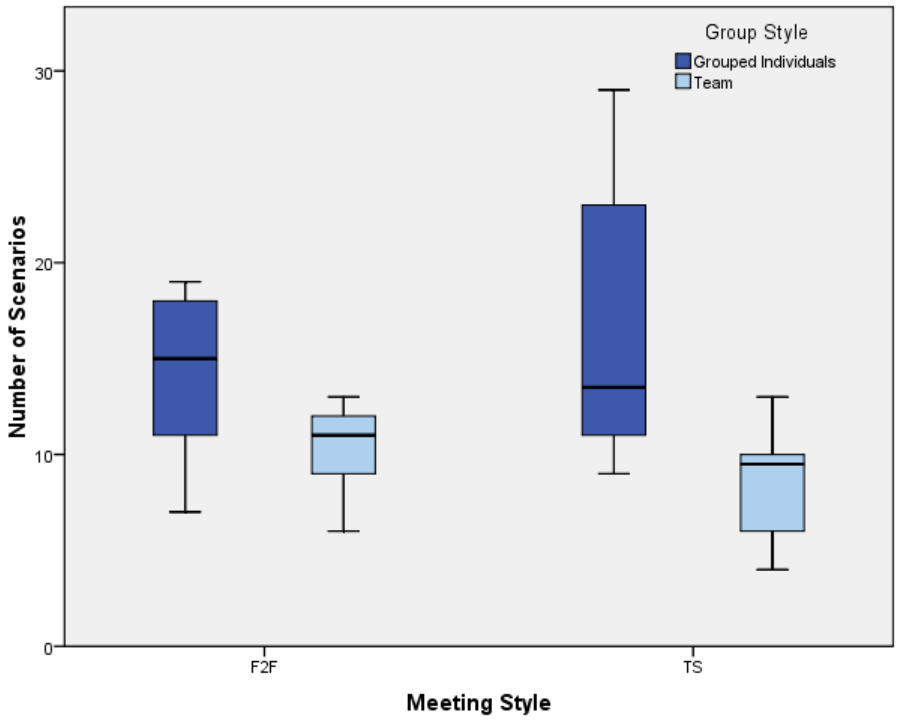


Figure 8-12 Number of all scenarios found by different group styles for Livenet

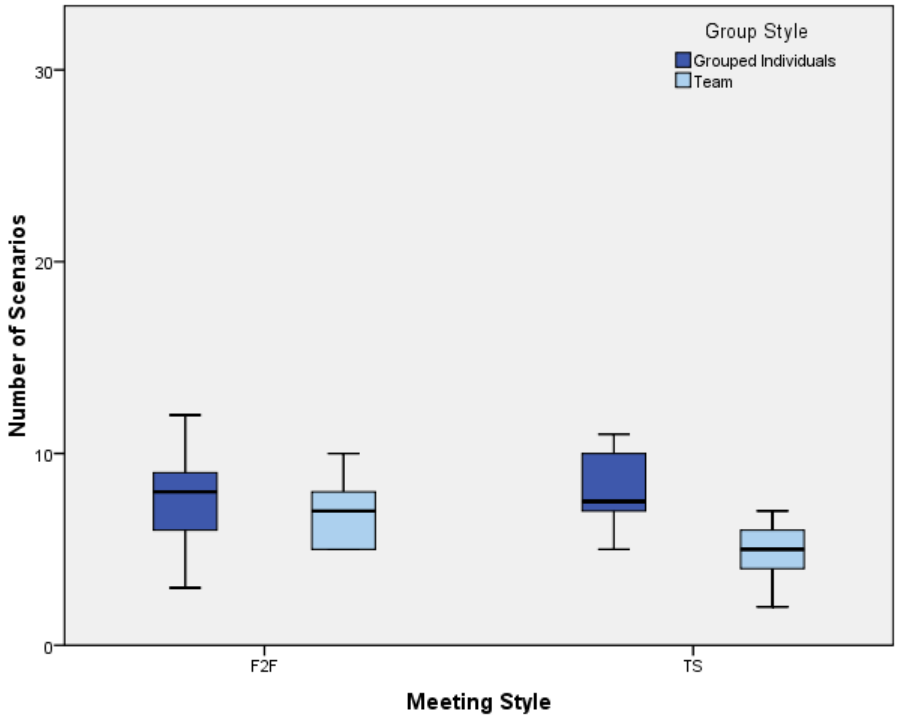


Figure 8-13 Number of TopRated-Sceanrios found by different group styles for Livenet

As can be seen in Figure 8-12 and Figure 8-13 much more "normal" scenarios have been found by individuals and teams than TopRated-Scenarios.

	Meeting Style	N	Mean	Std. Deviation
Number of Scenarios	F2F	9	14.22	4.38
	TS	10	16.50	7.49

a. Group Style = Grouped Individuals

Table 8-12 All scenarios found by grouped individuals for Livenet

	Meeting Style	N	Mean	Std. Deviation
Number of Scenarios	F2F	9	10.56	2.19
	TS	10	8.50	2.88

a. Group Style = Team

Table 8-13 All scenarios found by teams for Livenet

Table 8-12 and Table 8-13 show what we have learned before: More scenarios have been found in the individual phase than in the team phase. What is very interesting tough is that TS teams did find only half of the scenarios in the team phase they had found individually. F2F teams did find 2/3 of the scenarios they had found individually in the team phase, which is a better score. Let's have a look at the results of how many TopRated-Scenarios they have found.

	Meeting Style	N	Mean	Std. Deviation
Number of Scenarios	F2F	9	7.11	2.93
	TS	10	8.00	2.06

a. Group Style = Grouped Individuals

Table 8-14 TopRated-Scenarios found by grouped individuals for Livenet

	Meeting Style	N	Mean	Std. Deviation
Number of Scenarios	F2F	9	6.89	1.97
	TS	10	5.00	1.63

a. Group Style = Team

Table 8-15 TopRated-Scenarios found by teams for Livenet

As seen in Table 8-14 and Table 8-15 the participants also did find more TopRated-Scenarios individually than they did in the team phase, but the difference is not that big as with all scenarios found. Like mentioned before already, the difference with F2F teams is not as big as with TS teams when comparing individual and team phase.

Now that we have all necessary data collected, we will have a look at the difference of all scenarios and TopRated-Scenarios found individually and in the team phase.

Livenet		Grouped Individuals		Team		**Increase**
		All	TR	All	TR	
F2F	Amount (Mean)	14.22	7.11	10.56	6.89	
	% TopRated		50.00		65.25	**15.25**
TS	Amount (Mean)	16.50	8.00	8.50	5.00	
	% TopRated		48.48		58.82	**10.34**

Table 8-16 Comparison of all scenarios found with TopRated-Scenarios found for Livenet

As can be seen in Table 8-16 both meeting styles – F2F and TS – have found more TopRated-Scenarios in the team phase compared to the individual phase relatively. Individually F2F teams found 14.22 scenarios altogether, where 7.11 scenarios have been TopRated-Scenarios, which is 50%. Individually TS teams have found 16.50 scenarios altogether, where 8 scenarios have been TopRated-Scenarios. In the team phase F2F teams found 6.89 TopRated-Scenarios out of 10.56 scenarios altogether, which is 65.25%. In the team phase TS teams found 5 TopRated-Scenarios out of 8.5 scenarios altogether, which is 58.82%. This means that the percentage of TopRated-Scenarios in the scenario profiles of the teams has increased for both, F2F and TS teams. But the increase has been 15.25% with F2F teams and 10.34% with TS teams, which means that F2F teams have a 1.5 times higher increase of found TopRated-Scenarios than TS teams.

8.4.3 TopRated-Scenarios found by individuals and teams for Wiki

Now we will have a look at the results from session 2 where change scenarios for the Wiki System had to be found.

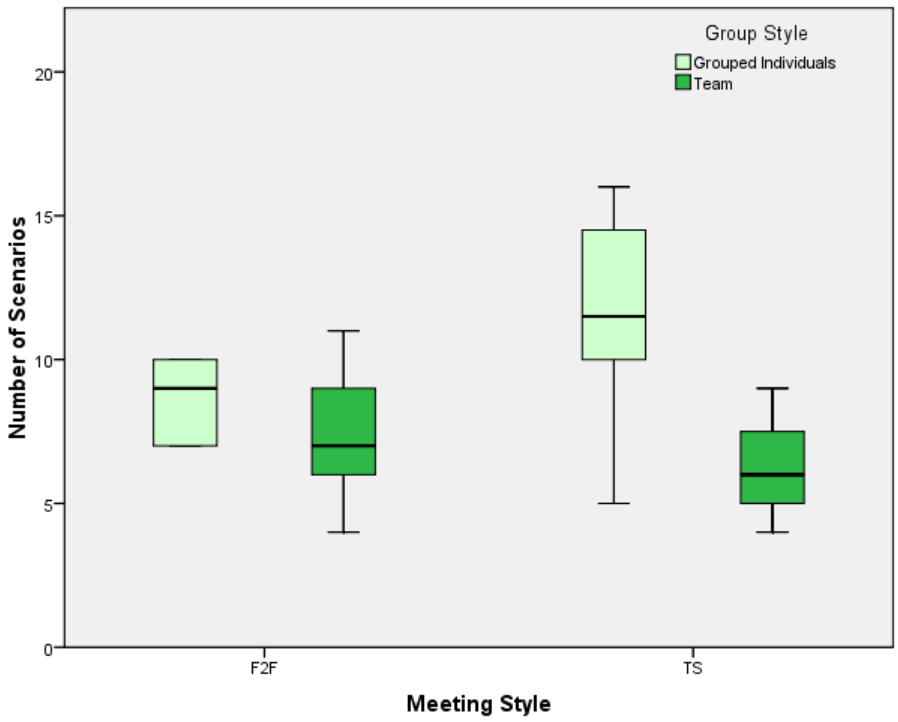


Figure 8-14 Boxplot of TopRated-Scenarios with different group styles for Wiki

In Figure 8-14 we can see the number of TopRated-Scenarios found by F2F and TS teams and their grouped individuals. This gives us a good overview of how many scenarios have been found in the individual phase of the team members and how many have made it onto the team scenario list. Looking at the boxplot we see that F2F teams do have found much less TopRated-Scenarios individually than the participants of the TS teams. When looking at the TS teams we see that they

have much less TopRated-Scenarios found in the team phase than they had found in the individual phase.

	Meeting Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. interval)
Number of Scenarios	F2F	10	9.9	3.7	0.327(-)
	TS	8	11.63	3.50	

a. Group Style = Grouped Individuals

Table 8-17 TopRated-Scenarios found by grouped individuals for Wiki

	Meeting Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. interval)
Number of Scenarios	F2F	10	7.5	2.32	0.203(-)
	TS	8	6.25	1.67	

a. Group Style = Team

Table 8-18 TopRated-Scenarios found by teams for Wiki

As we can see in Table 8-17 and Table 8-18 the participants have found more TopRated-Scenarios in their individual phase than in the team phase. What is very interesting is the fact that TS individuals have found much more TopRated-Scenarios than F2F individuals, but TS teams also have lost a lot of their scenarios in the team phase – much more then the F2F teams. Nearly ½ of the TopRated-Scenarios found by TS individuals got lost in the team phase.

When we look at the t-tests with a 95% confidence interval we see that the result is not below 0.05 in any case, which means that a clear answer is not possible.

	Group Style	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. interval)
Number of Scenarios	Grouped Individuals	18	10.67	3.61	0.001(s)
	Team	18	6.94	2.1	

Table 8-19 TopRated-Scenarios with group styles for Wiki

Table 8-19 shows results for the 2 different group styles – grouped individuals and teams. As can be seen, participants have found more TopRated-Scenarios individually than in teams, which maybe mean that they did not have enough time in the team phase to discuss all scenarios. The t-test for the 2 different group styles shows that the result is 0.001(s), which is below 0.05 and means that there is a connection between the number of scenarios found and the group style, so the number of found scenarios by grouped individuals and teams is not coincidence as has been for Livenet.

8.4.4 TopRated- and all scenarios found by teams and individuals for Wiki

To see if F2F teams did better scenario discussion in the team phase than TS teams, we need to have a look at how many scenarios the participants and teams have found in general and how many TopRated-Scenarios have been in their scenario profiles. By evaluating the percentage of TopRated-Scenarios in the scenario-profile of individuals and teams, we can have a look at how this percentage has changed from the individual to the team phase – like we did for Livenet above.

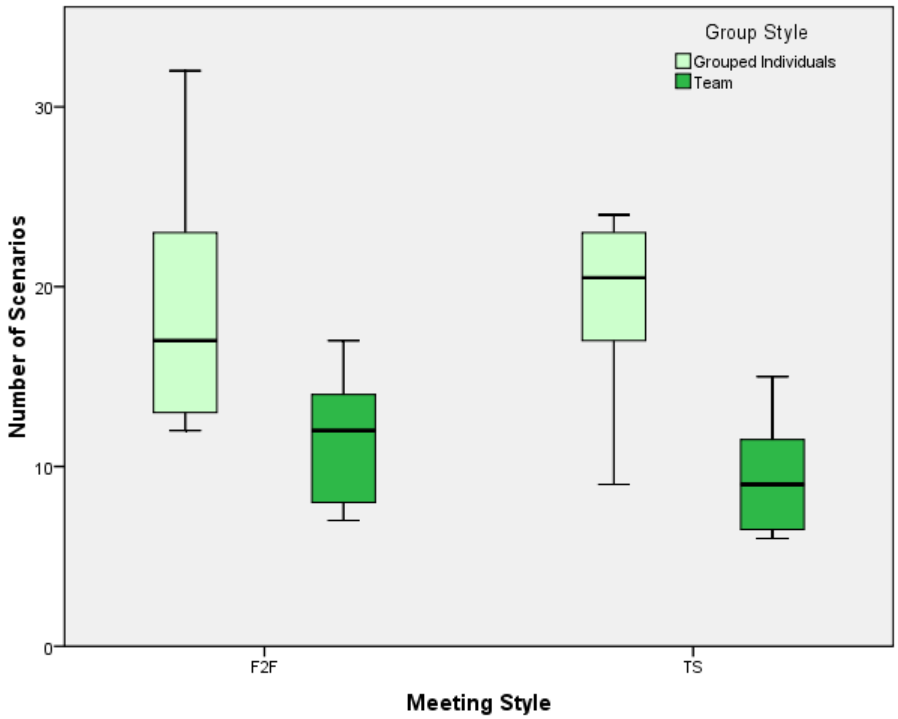


Figure 8-15 Number of all scenarios found by different group styles for Wiki

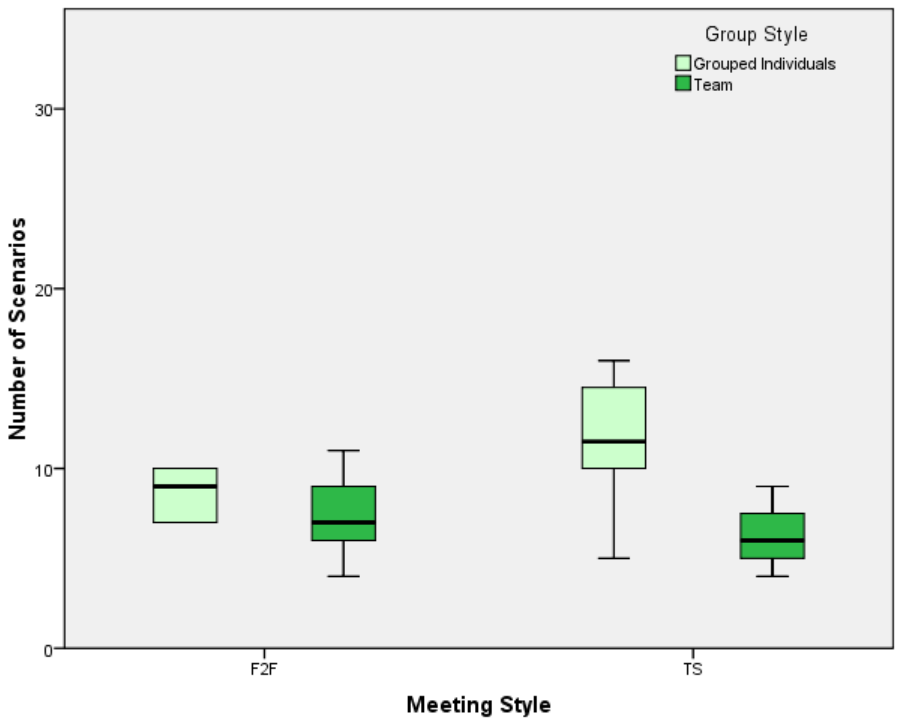


Figure 8-16 Number of TopRated-Sceanrios found by different group styles for Wiki

When looking at Figure 8-15 we see that individuals have found more scenarios than teams - which could mean again, that there was not enough time for the teams in the team phase to discuss their scenarios. When looking at the teams we see that F2F teams have found more scenarios than TS teams. Figure 8-16 shows how many TopRated-Scenarios have been found. Again individuals have found more than teams, but this time – in opposed to Livenet – TS teams have found more TopRated-Scenarios than F2F teams.

	Meeting Style	N	Mean	Std. Deviation
Number of Scenarios	F2F	10	18.70	6.5
	TS	8	19.25	5.09

a. Group Style = Grouped Individuals

Table 8-20 All scenarios found by grouped individuals for Wiki

	Meeting Style	N	Mean	Std. Deviation
Number of Scenarios	F2F	10	11.30	3.59
	TS	8	9.38	3.16

a. Group Style = Team

Table 8-21 All scenarios found by teams for Wiki

Table 8-20 shows that individuals of both meeting styles – F2F and TS – have found nearly the same amount of scenarios. Table 8-21 shows that F2F teams also have found more scenarios in the teams phase than TS teams.

	Meeting Style	N	Mean	Std. Deviation
Number of Scenarios	F2F	10	9.90	3.7
	TS	8	11.63	3.50

a. Group Style = Grouped Individuals

Table 8-22 TopRated-Scenarios found by grouped individuals for Wiki

	Meeting Style	N	Mean	Std. Deviation
Number of Scenarios	F2F	10	7.50	2.32
	TS	8	6.25	1.67

a. Group Style = Team

Table 8-23 TopRated-Scenarios found by teams for Wiki

When looking at the results of found TopRated-Scenarios in Table 8-22 we see that TS participants found much more individually than F2F participants, but when looking at Table 8-23 we see that F2F teams must have been much better in the team phase, because they have more TopRated-Scenarios in their profiles than TS teams, who lost nearly half of their TopRated-Sceanrios.

Wiki		Individual		Team		**Increase**
		All	TR	All	TR	
F2F	Amount (Mean)	18.70	9.90	11.30	7.50	
	% TopRated		52.94		66.37	**13.43**
TS	Amount (Mean)	19.25	11.63	9.38	6.25	
	% TopRated		60.42		66.63	**6.22**

Table 8-24 Comparison of all scenarios found with TopRated-Scenarios found for Wiki

Now we have all necessary data and numbers collected, we can have a look at Table 8-24. It shows that F2F and TS individuals found nearly the same amount of scenarios altogether, but TS individuals had 60% TopRated-Scenarios in their profiles, where F2F individuals only hat 53%. When looking at the team phase we see that both meeting styles have 66% of TopRated-Scenarios in their profiles, which is a little increase for TS teams, but a huge increase for the F2F teams. TS teams have an increase of 6.22% compared to the individual phase where F2F teams have an increase of 13.43% compared to the individual phase, which is twice as good.

8.5 Importance of experience in team-meetings

The third and final question we ask is: Do teams with more experience have better results, meaning that they find more important scenarios, than teams with less experience? By evaluating the experience level of every participant with an Experience Questionaire at the beginning, the experience level of every team can be calculated. We can now compare the experience level of every team with their results to see what influence the experience has. We will compare face to face and tool-support teams in that matter to see if there are different results. While communicating face to face every team-member might be able to contribute his experience better than team-members doing tool-supported communication, which could lead to better results for F2F teams. Figure 8-17 shows the experience of all teams. As mentioned before in chapter 8.2, every team could get a maximum of 4 points. When looking at the chart we see that the teams in our controlled experiment nearly have the same level of experience.

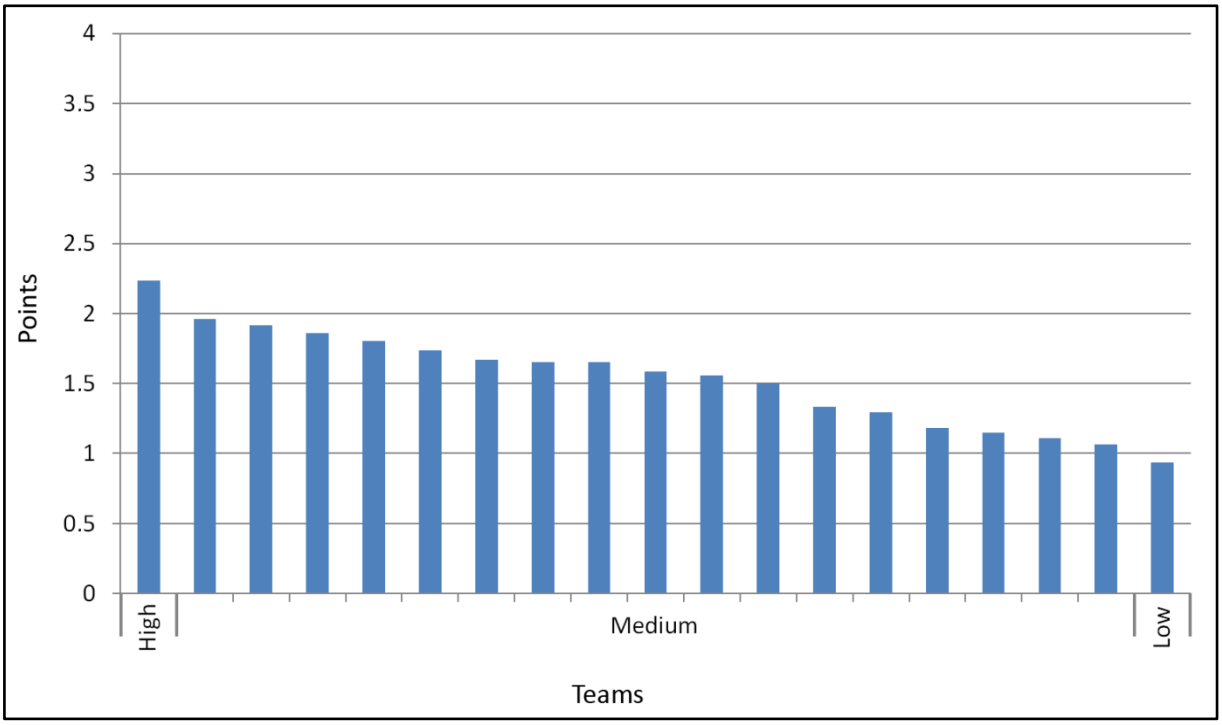


Figure 8-17 Experience of teams

8.5.1 Experience and found scenarios for Livenet

Figure 8-18 shows the experience points for F2F and TS teams for session 1 – Livenet. It can be seen that the average experience points of the 2 meeting styles are nearly the same.

Also Table 8-25 shows that F2F and TS teams nearly have the same average experience. F2F teams have 1.52 and TS teams 1.55 experience points.

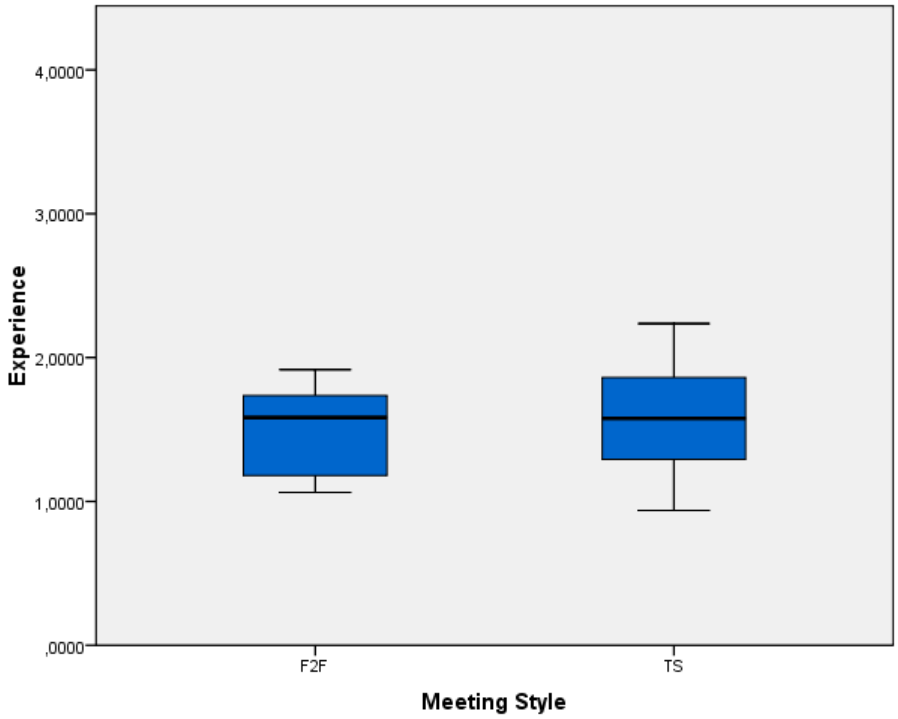


Figure 8-18 Boxplot of experience of teams for Livenet

	Meeting Style	N	Mean	Std. Deviation
Experience	F2F	9	1.52	0.31
	TS	10	1.55	0.4

Table 8-25 Experience of teams for Livenet

Figure 8-19 and Figure 8-20 show the experience of the teams in a descending order – the most experienced team on the left and less experienced team on the right. They also show the number of all scenarios found by the teams. If more experience would mean more found scenarios then the number of found scenarios should also descend from left to right, what is does not do. So it seems that there is no connection between the experience we measured and the number of scenarios found by the teams.

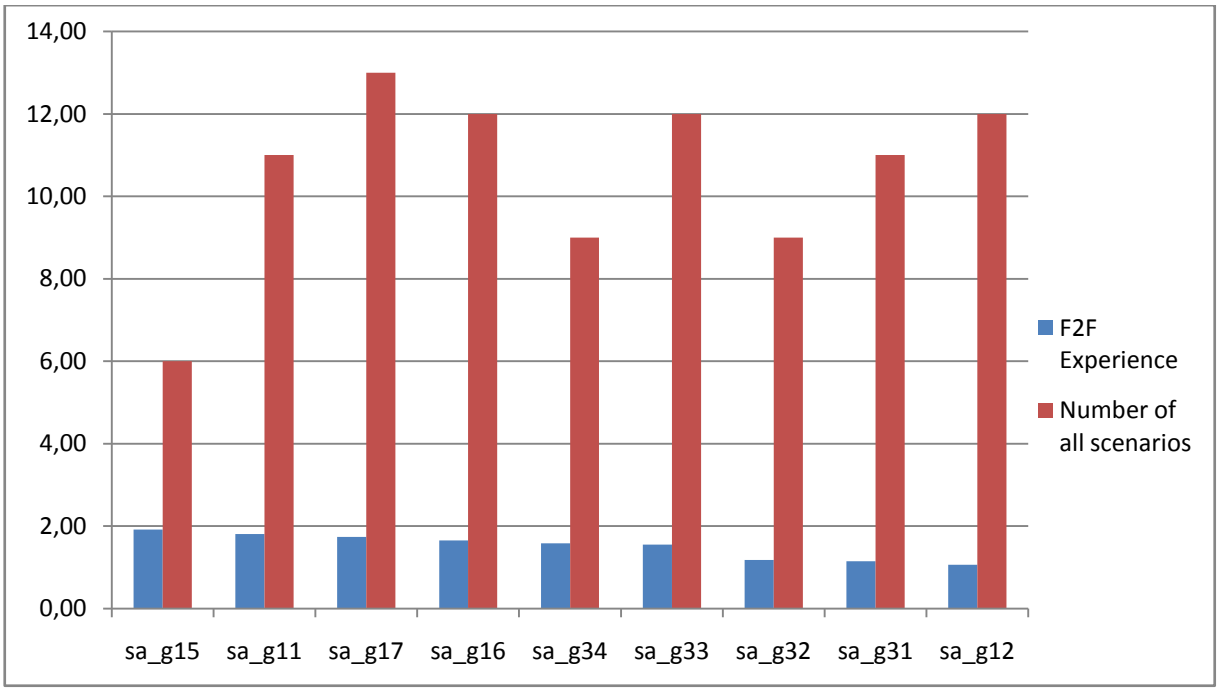


Figure 8-19 Experience and all found scenarios of F2F teams for Livenet

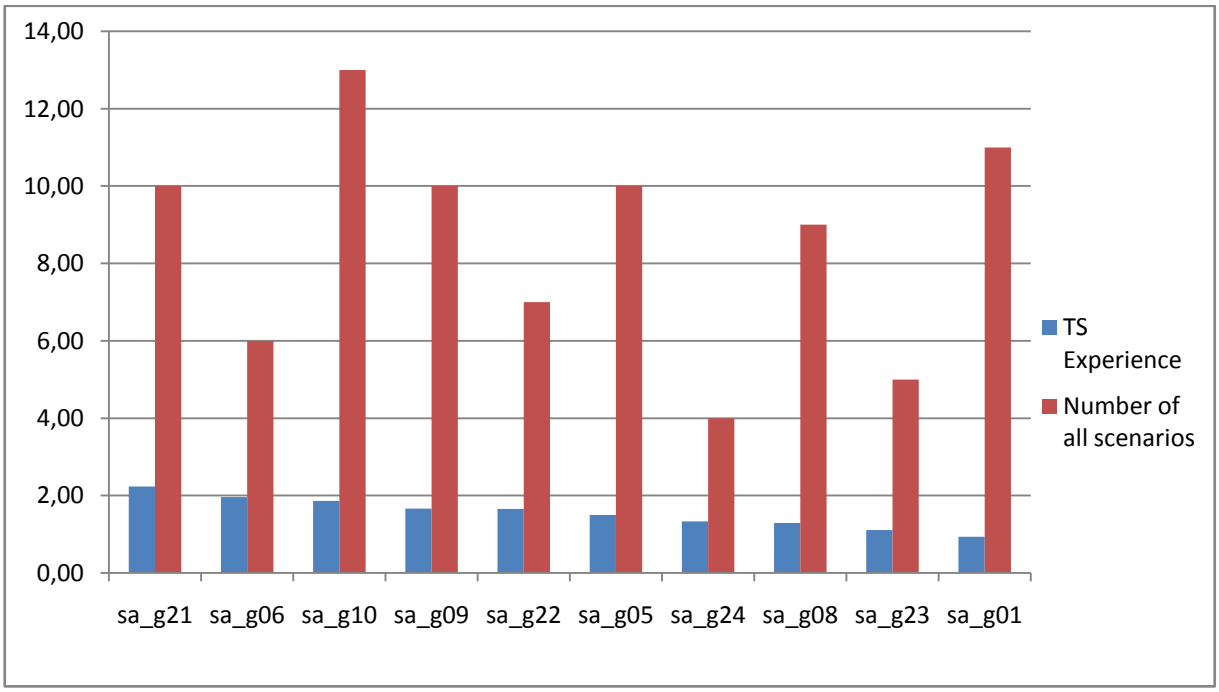


Figure 8-20 Experience and all found scenarios of TS teams for Livenet

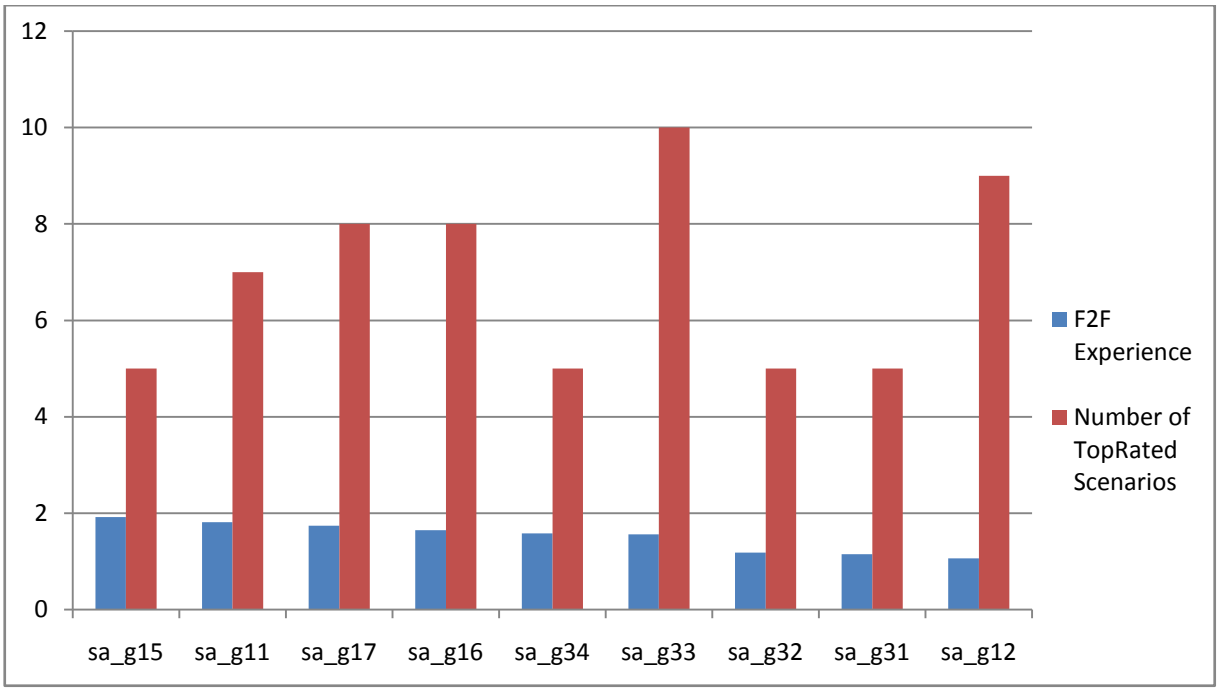


Figure 8-21 Experience and TopRated-Scenarios found of F2F teams for Livenet

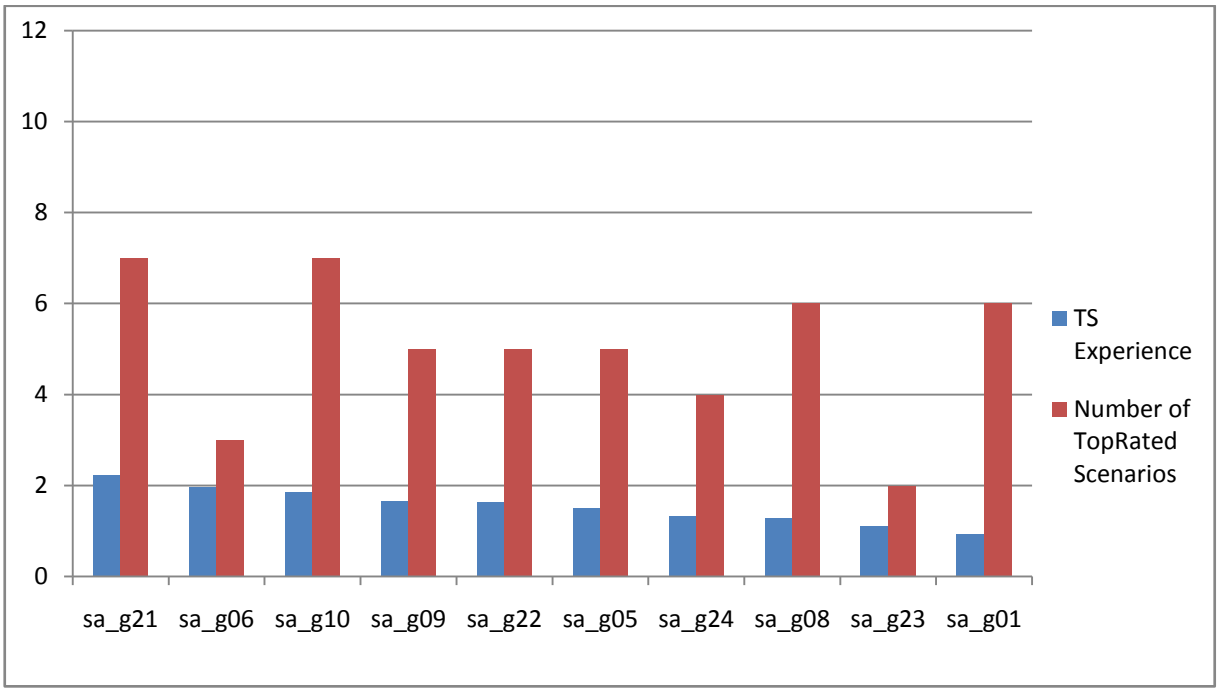


Figure 8-22 Experience and TopRated-Scenarios found of TS teams for Livenet

Figure 8-21 and Figure 8-22 also show the experience of the teams, descending from left to right. They also show the number of TopRated-Scenarios of the teams. Again, it seems that there is no connection between the experience of the teams we measured and the TopRated-Scenarios they have found.

	Experience	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. interval)
Number of Scenarios	>= 1.60	9	6.11	1.69	0.662(-)
	< 1.60	10	5.7	2.31	

Table 8-26 Experience and TopRated-Scenarios found for Livenet

To evaluate if there is a connection between the number of found TopRated-Scenarios and the experience of the teams, we did a t-test with a 95% confidence interval. We decided to split the teams into two groups – more experienced and less experienced teams – and set the boundary at 1.6 experience points. This boundary defines 9 teams more experienced and 10 teams less experienced. The result in Table 8-26 show that there is no connection – the result of the t-test is 0.662(-) which is much over 0.05.

8.5.2 Experience and found scenarios for Wiki

Figure 8-23 shows the experience of the teams for Wiki. Of course it is nearly the same as it has been for Livenet, because the team members did not change, the teams only switched meeting style, but in session 2 (Wiki System) one team less did participate than in session 1 (Livenet).

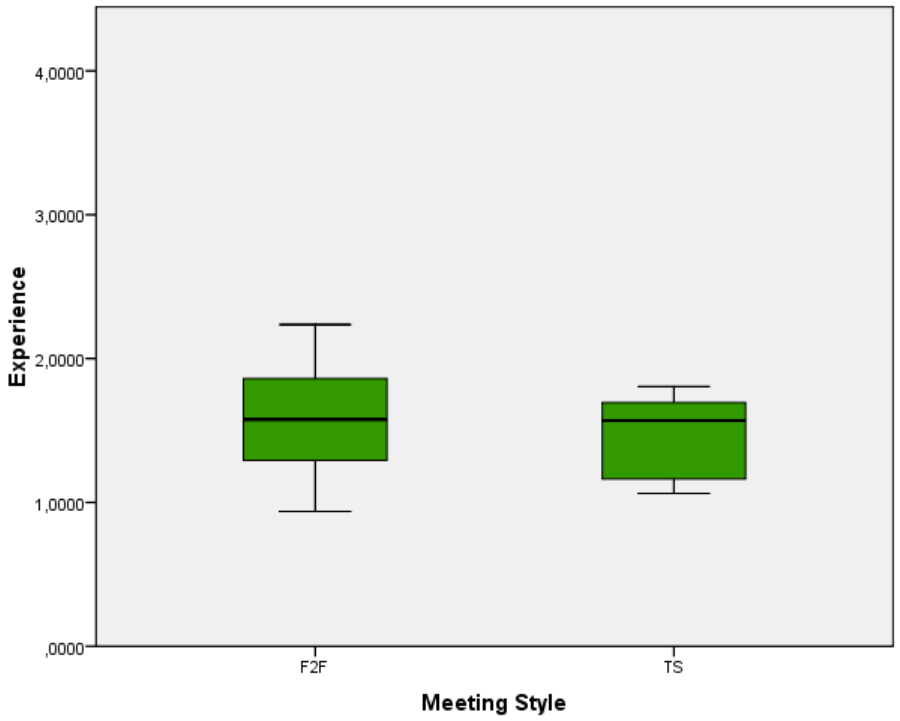


Figure 8-23 Boxplot of experience of teams for Wiki

	Meeting Style	N	Mean	Std. Deviation
Experience	F2F	10	1.55	0.4
	TS	8	1.47	0.29

Table 8-27 Experience of teams for Wiki

The experience of the teams is nearly the same. F2F teams have 1.55 experience points and TS teams have 1.47 experience points in average as can be seen in Table 8-27.

Figure 8-24 and Figure 8-25 show the experience of the teams descending from left to right. This means the team with the most experience is on the left and the team with the least experienced team on the right. The 2 diagrams also show the number of all scenarios found by the teams. If more experience would mean more found scenarios then the number of found scenarios should also decrease from left to right, what it does not do again.

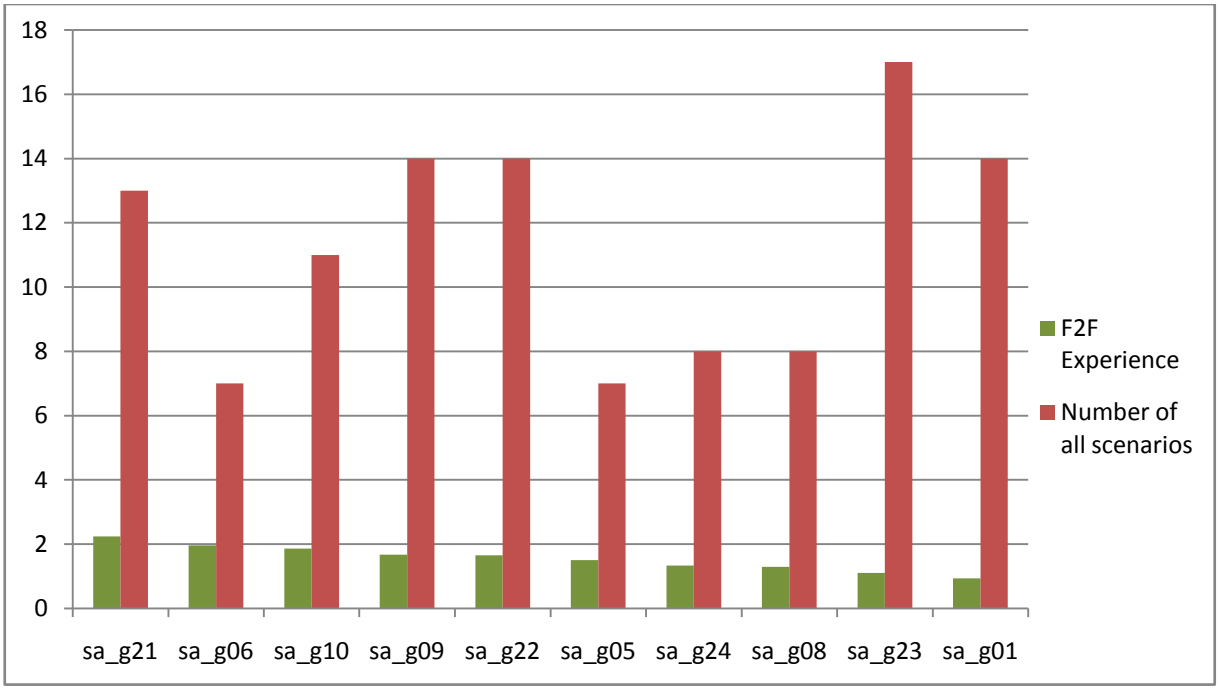


Figure 8-24 Experience and all found scenarios of F2F teams for Wiki

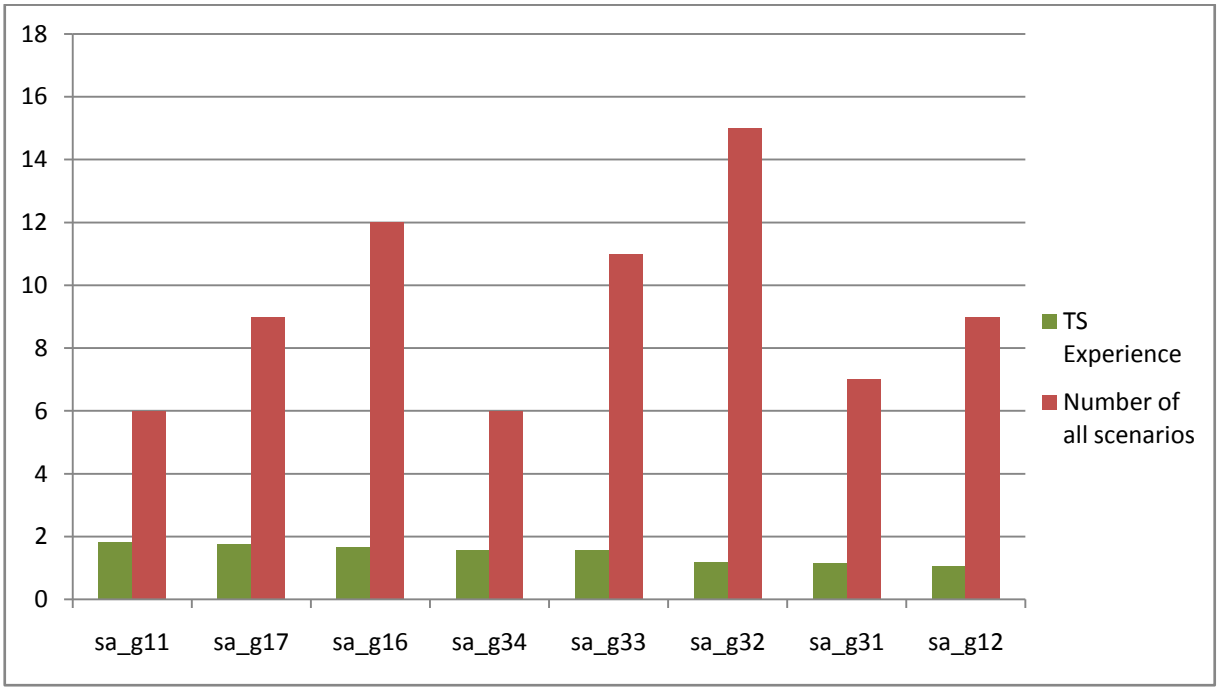


Figure 8-25 Experience and all found scenarios of TS teams for Wiki

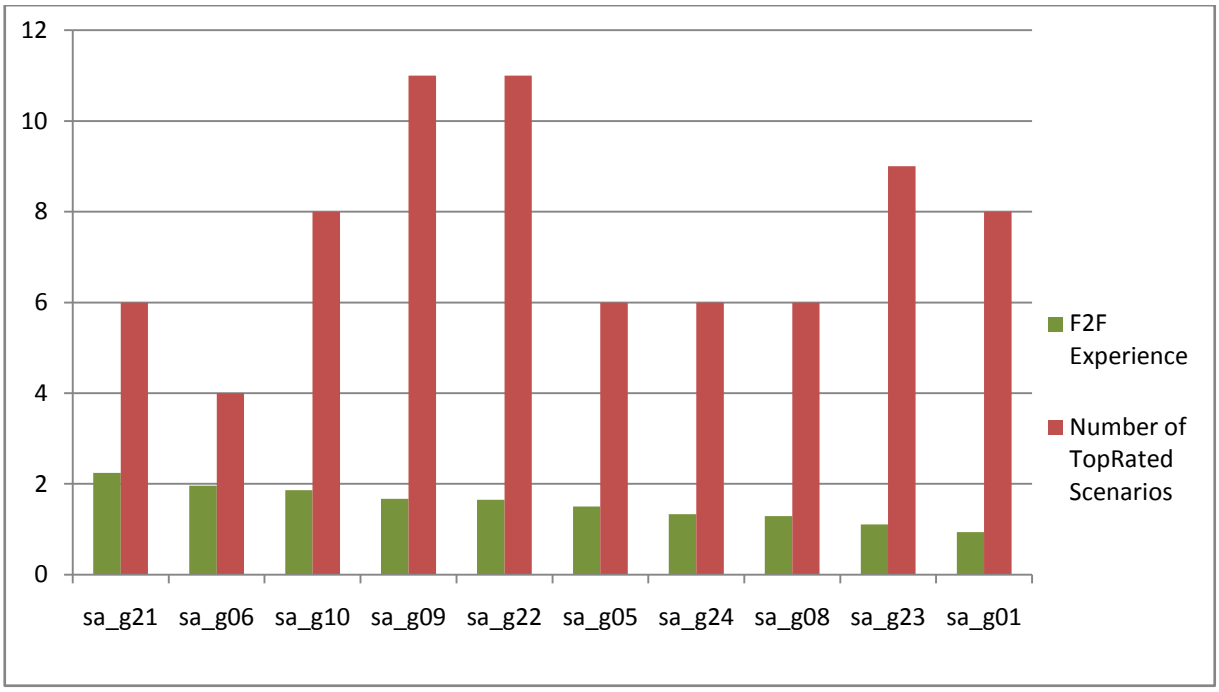


Figure 8-26 Experience and TopRated-Scenarios found of F2F teams for Wiki

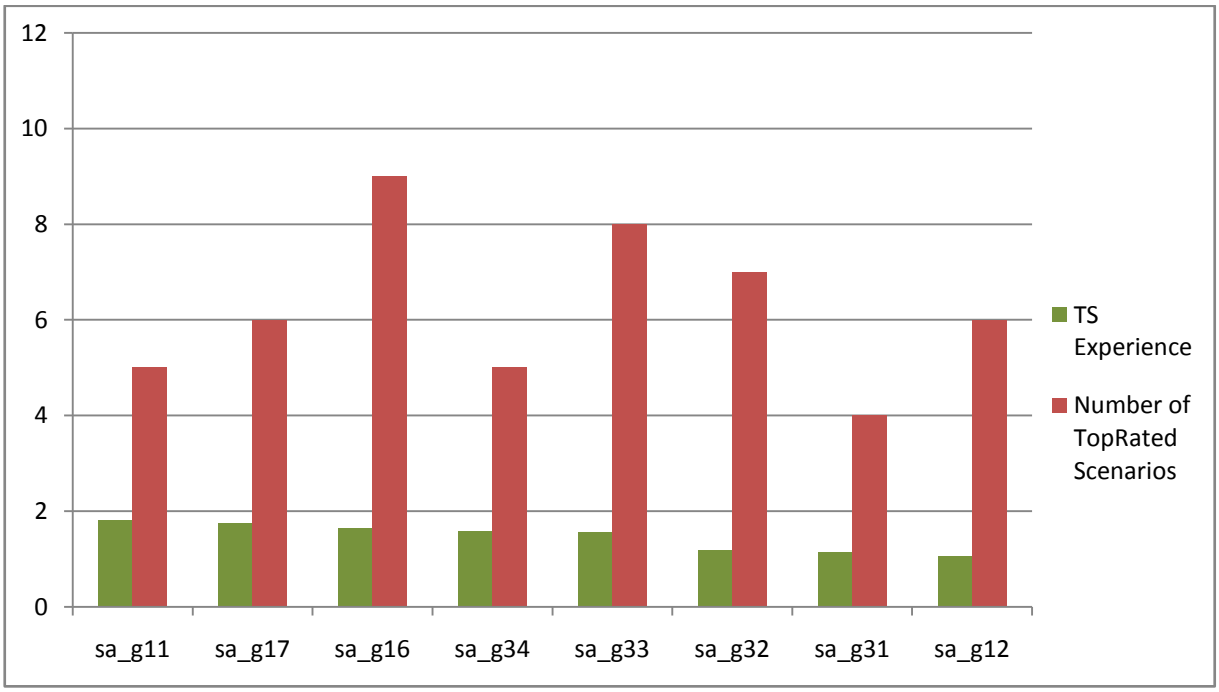


Figure 8-27 Experience and TopRated-Scenarios found of TS teams for Wiki

Figure 8-26 and Figure 8-27 again show the experience of the teams descending from left to right. They also show the number of found TopRated-Scenarios found by the teams. Again, the number of found TopRated-Scenarios does not descend from left to right, so it seems that there is no connection between experience and found scenarios.

	Experience	N	Mean	Std. Deviation	p-value (2-tailed, 95% conf. interval)
Number of Scenarios	>= 1,60	8	7.5	2.67	0.33(-)
	< 1,60	10	6.5	1.51	

Table 8-28 Experience and TopRated-Scenarios found for Wiki

We did a t-test with a 95% confidence interval to evaluate if there is a connection between experience and number of found scenarios. We again decided to split the teams into two groups – more experienced and less experienced teams – and set the boundary at 1.6 experience points. This boundary defines 8 teams more experienced and 10 teams less experienced. The result of the t-test is 0.33(-) which is way over 0.05 which means that there is no connection between the experience of the teams we measured and the number of the scenarios they found, as can be seen in Table 8-28.

9 Discussion

In this chapter we will discuss the meaning of the results from the last chapter. The questions that have been asked have been evaluated there. We will now take a look at these results and discuss what they mean and what we can learn from them.

9.1 Comparing different methods of team-meetings

The first question we want to evaluate is, if teams that communicate face to face find more important scenarios than teams communicating with some kind of tool-supported communication. In our controlled experiment all participants had to find scenarios as individuals first. In the second phase - the team phase – the participants had to discuss their scenarios in groups. In this team phase one half of the groups was sitting together physically in the same room (the so called face to face teams) and the other part was not sitting together physically – participants of a so called tool-support team were sitting in front of a computer, communicating with their team members over a chat-application.

More formally, the first null hypothesis is:

H0: The face to face discussion of software architecture scenarios for architecture evaluation in teams will not have better results than discussing them over tool-supported communication.

The first alternative hypothesis is:

H1: The face to face discussion of software architecture scenarios for architecture evaluation in teams will have better results than discussing them over tool-supported communication.

If the face to face communication has better results than the tool-supported communication in our experiment, we will reject H0.

As we have seen, F2F teams have found more TopRated-Scenarios both for Livenet and the Wiki System. F2F teams have found 6.89 TopRated-Scenarios for Livenet while TS teams have found 5. For Wiki F2F teams have found 7.50 TopRated-Scenarios while TS teams have found 6.25. As we can see, the difference between the two meeting styles is smaller in session 2 (Wiki System) than it has been in session 1 (Livenet). This could mean that all participants earned some experience in software architecture evaluation in session 1 and could use this experience in session 2, which results in a smaller difference in the number of scenarios found.

When looking at the t-test with a 95% confidence interval we did for Livenet, we see that the result is 0.03(s) which is below 0.05. This means that there is a connection between the number of TopRated-Scenarios found and the meeting style – so H0 can be rejected. When looking at the same t-test for Wiki we see that the result is over 0.05, which does not let us reject H0 in this case, because it cannot be said clearly if there is a connection between the meeting style and the number of Toprated-Scenarios for the Wiki System.

We also separated the results into teams that used change categories to find scenarios and teams that did not. Such categories could be: User-Interface changes, Security-policy changes, Performance

& scalability changes, etc. The idea was to evaluate which teams would find better results: Teams that are guide with change categories provided or teams that had to find scenarios freely. When looking at these results for teams with “categories used” and “categories not used”, we see again that F2F teams did find more TopRated-Scenarios than TS teams, both for Livenet and Wiki. The t-tests we did for this evaluation did not give as a clear result though. The results for teams with and without change categories where over 0.05 which means that we cannot see a relationship between the number of TopRated-Scenarios found and the meeting style in this case. The problem here might be that we did not have enough teams in our controlled experiment and therefore such a detailed statistical evaluation cannot be done. Every statistic element (F2F and categories, TS and categories, F2F no categories, TS no categories) only consists of 4 or 5 teams, which makes a well evaluated statistical statement very difficult. When not separating into “categories used” and “categories not used”, the number of teams are enough to do evaluations, which can be seen in the t-test mentioned, that is below 0.05 for Livenet.

So, when looking at the results, where we differed between F2F and TS teams only, we can reject the null hypothesis for Livenet and can therefore say that F2F teams do find better scenarios then TS teams for Livenet. For the Wiki System we cannot do this, because the t-test does not give us a satisfying result. This can have two reasons: 1) The participants earned some experience about the architecture evaluation process as a whole in session 1 and used this experience in session 2, which is why there is not as big difference between various teams as in session 1 and 2) We also changed the group of participants who used change categories in session 2. The participants who did use them in session 1 did not in session 2 and those who did not use them in session 1 did in session 2. This decision makes session 1 different from session 2. The group of participants who did use change categories in session 2 did not know them before and so could not use them in session 1. But the participants who used them in session 1 and not in session 2 could remember the change categories from session 1 (though they have been partly different) and so could use this knowledge in session 2. So we have the situation in session 2 that every participant somehow knows about the change categories where in session 1 really only half of the participants knew about them. A solution for future controlled experiments in this area might be, not to change participants who use categories. When they are assigned to the “categories used” group they should use them throughout the whole experiment.

9.2 Comparing individuals and teams

The second question we ask is, if teams find more important scenarios than individuals. In our controlled experiment everyone had to find scenarios individually first. In the team phase afterwards every scenario should have been discussed to write it down or not. That means that the results in the team-phase should have better results than those in the individual phase, because not useful scenarios should have been left out. So we will have a look at F2F and TS teams to evaluate which form of communication had better results compared to the individual phase. If the scenarios have been discussed well the amount of more important scenarios in the specific scenario profile has to be higher as if the scenarios have not been discussed well.

The second null hypothesis is:

H0: Teams communicating face to face cannot do better communication and so cannot discuss scenarios for software architecture evaluation more in detail than tool-supported communicating teams, which does not result in a higher amount of more important scenarios in their scenario profiles compared to the individual phase.

The second alternate hypothesis is:

H1: Teams communicating face to face can do better communication and so can discuss scenarios for software architecture evaluation more in detail than tool-supported communicating teams, which results in a higher amount of more important scenarios in their scenario profiles compared to the individual phase, because not useful scenarios are left out.

If the face to face teams have better results compared to the individual phase than tool-supported teams, H0 will be rejected.

When we look at the results we see that the participants always found more scenarios in general individually than in the team phase. They also found more TopRated-Scenarios individually than in teams. This may be a sign that the team phase has been too short for the teams to discuss all scenarios. If the team phase would have been longer than 60 minutes the teams may have found less scenarios in general, because useless scenarios would have been left out, but they may have found nearly the same number of TopRated-Scenarios in the team phase as they have found individually.

We see that F2F and TS individuals did find nearly the same amount of scenarios, which is clear, because there has been no team effect in the individual phase of course. But when looking at the team results, F2F teams have found more scenarios in general and TopRated-Sceanrios than TS teams.

To answer the question if F2F teams can do better discussion, because they are sitting together face to face, we had to evaluate how many scenarios did the participants find individually and how many percent of these found scenarios have been TopRated-Scenarios. We then had a look at the team results to see how many scenarios they did find and how many percent of those have been TopRated-Scenarios. When the scenarios have been discussed well, useless scenarios should have been left out in the team-phase which should result in a higher percentage of TopRated-Scenarios in the team's scenario-profiles. This was the case.

When we looked at the individual results we saw that around 50% (for Livenet) respectively around 56% (for Wiki) of the scenarios found are TopRated-Scenarios. When we looked at the team results we saw that 59% to 65% (for Livenet) respectively over 66% (for Wiki) of the scenarios found are TopRated-Scenarios. As can be seen the results were slightly different for Livenet and Wiki, but what we can say is that F2F teams always had a higher percentage of TopRated-Scenarios in their profiles than TS teams. F2F teams also had a higher percentage increase of TopRated-Scenarios in their profiles compared to the individual phase.

This means that F2F teams did better and more effective scenario discussion than TS teams, because they had a higher increase of TopRated-Scenarios in their profiles than TS teams.

When looking at the t-test we did we see that there is no connection between the number of found scenarios and the meeting style for individual scenario profiles which is clear, because there is no team effect in the individual phase. We also did a t-test to see if there is a connection between the number of scenarios found and the group style (grouped individual or team) and the result was below 0.05 which means that there is a connection.

When looking at the results we have for hypothesis 2 we can reject H0, which means that F2F teams do better discussion of scenarios found and therefore do leave out useless scenarios which results in a higher percentage of TopRated-Scenarios in their scenario-profiles.

9.3 Importance of experience in team-meetings

The third and final question we ask is: Do teams with more experience have better results, meaning that they find more important scenarios, than teams with less experience? By evaluating the experience level of every participant with an Experience Questionaire at the beginning, the experience level of every team can be calculated. We can now compare the experience level of every team with their results to see what influence the experience has. We will compare face to face and tool-support teams in that matter to see if there are different results. While communicating face to face every team-member might be able to contribute his experience better than team-members doing tool-supported communication, which could lead to better results for F2F teams. Figure 8-17 shows the experience of all teams. As mentioned before in chapter 8.2 every team could get a maximum of 4 points. When looking at the chart we see that the teams in our controlled experiment nearly have the same level of experience.

The third null hypothesis is:

H0: More experienced teams communicating face to face have not found more important scenarios for software architecture evaluation than teams doing tool-supported communication, because participants cannot contribute their experience better when communicating face to face.

H1: More experienced teams communicating face to face have found more important scenarios for software architecture evaluation than teams doing tool-supported communication, because participants can contribute their experience better when communicating face to face.

If face to face teams have better results than tool-support teams with the same experience level we will reject H0.

First we calculated the experience of every team and evaluated the average experience points of F2F and TS teams. The result was that F2F and TS teams nearly had the same experience level. This is a good indicator for us, because it says that we can compare our results very easily, because the teams are nearly equally experienced, so the results we have until now cannot be influenced by large differences in the team's experience. Then we compared the results with the number of all scenarios found – for F2F and TS teams. When looking at the charts we saw that high experienced teams did not necessarily find more scenarios than teams with less experience. Then we compared the experience points of the teams with the number of TopRated-Scenarios found and the picture was

the same: Teams with higher experience did not find more TopRated-Scenarios than lower experienced teams – whether if it had been F2F or TS teams.

To substantiate our results we decided to make a t-test to evaluate if there is a connection between the number of found scenarios and the level of experience of the teams. We decided to split the teams into two groups – more experienced and less experienced teams – and set the boundary at 1.6 experience points. This boundary defines 9 (for Livenet) respectively 8 (for Wiki) teams more experienced and 10 teams less experienced. The result shows that there is no connection – the result of the t-test is much over 0.05.

So, we cannot reject H0 in this case because there is no evidence that the experience has a connection to the number of scenarios found – at least the experience we measured. We focused on questions about software engineering, project management and software evaluation on our Experience Questionaire we handed out to every participant. Maybe another approach in measuring the experience of the participants would be more useful. It seems that knowledge in various software-engineering areas is not as useful as actual knowledge about architecture evaluation or software evaluation in general.

9.4 Feedback from the participants

Every participant had to fill out a Feedback Questionaire at the end of every team phase. In the questionnaire we asked them, if they had enough time to discuss scenarios and if not, how many additional minutes they would have needed. Figure 9-1 shows that 81% of the F2F team members thought that they had enough time. 19% thought that the time was not enough and said that they would need 18.67 minutes more in average. Figure 9-2 shows the feedback from the TS team members. Here 73% thought that they did not have enough time to discuss the scenarios and that they would need nearly half an hour more time for discussion. Only 27% of the TS team members said that they had enough time.

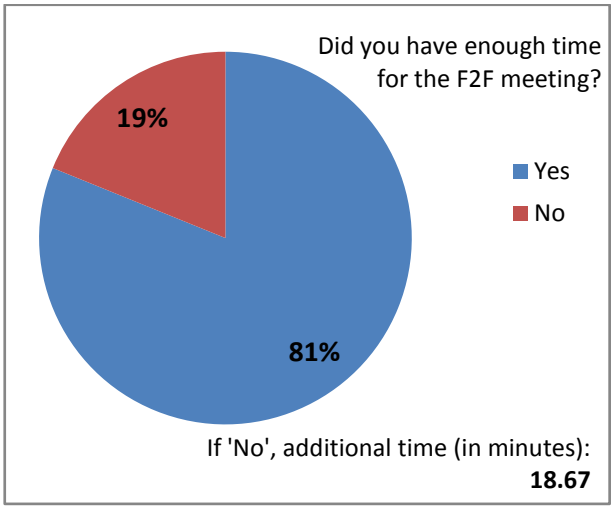


Figure 9-1 Time-Feedback of F2F teams

Did you have enough time for the F2F meeting?

Answer	Number of Participants
Yes	43
No	10

Table 9-1 Time-Feedback of F2F teams

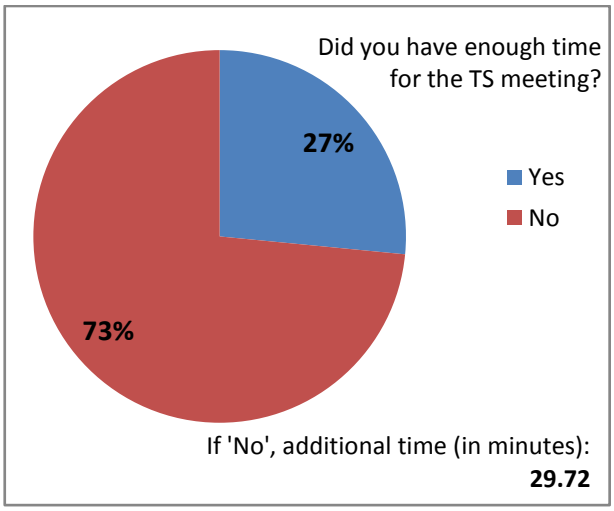


Figure 9-2 Time-Feedback of TS teams

Did you have enough time for the TS meeting?

Answer	Number of Participants
Yes	13
No	36

Table 9-2 Time-Feedback of TS teams

So, we see that TS teams needed much more time for scenario discussion than F2F teams. That's maybe, because scenario discussion is much more complicated doing discussion in a chat application than doing the discussion face to face. The participants also had to fill out a Final Questionaire at the end of session 2 where we asked them questions about the controlled experiment as a whole and about the difference meeting styles.

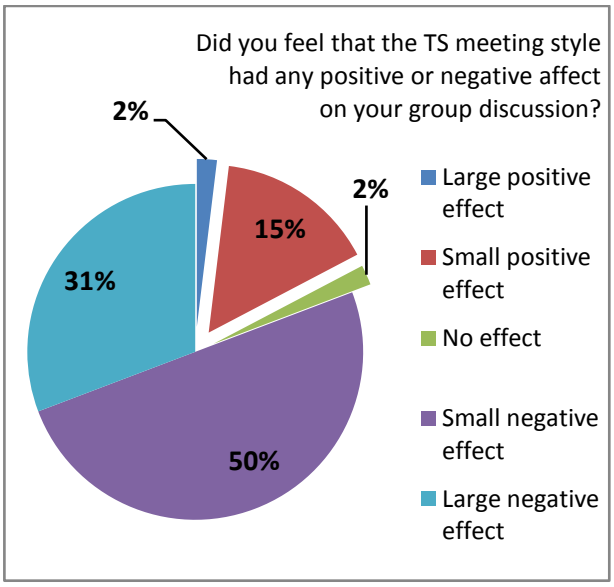


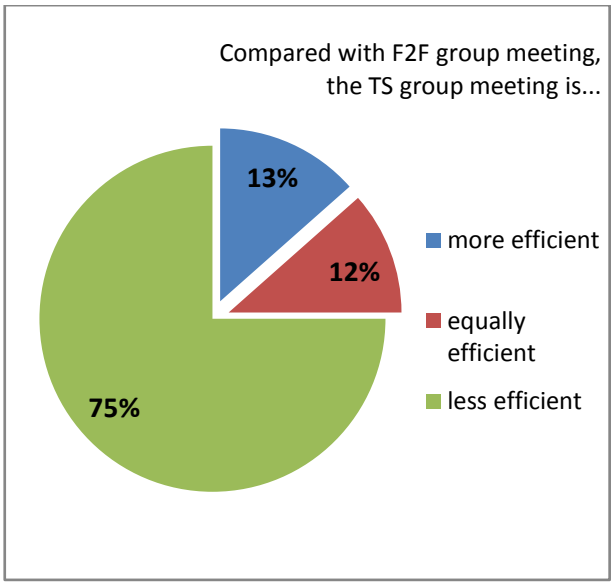
Figure 9-3 Effect of TS meeting style on group discussion

Did you feel that the TS meeting style had any positive or negative affect on your group discussion?

Answer	Number of Participants
Large positive effect	1
Small positive effect	8
No effect	1
Small negative effect	26
Large negative effect	16

Table 9-3 Effect of TS meeting style on group discussion

In Figure 9-3 we see that in total 81% of the participants thought that the TS meeting style had a negative effect on the group discussion and only 17% thought it had a positive effect.



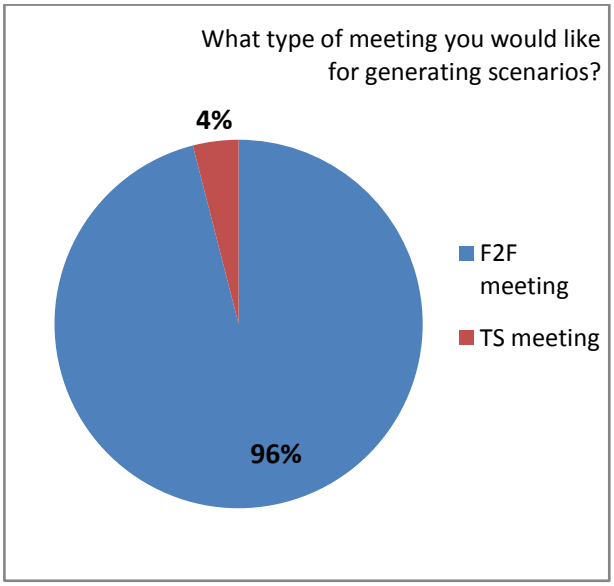
Compared with F2F group meeting, the TS group meeting is...

Answer	Number of Participants
more efficient	7
equally efficient	6
less efficient	39

Table 9-4 Comparison of F2F and TS meeting style

Figure 9-4 Comparison of F2F and TS meeting style

Figure 9-4 shows a comparison of the F2F and TS meeting style. 75% of the participants think that the TS meeting is less efficient than the F2F meeting, where only 13% thought that it is more efficient than the F2F meeting.



What type of meeting you would like for generating scenarios?

Answer	Number of Participants
F2F meeting	48
TS meeting	2

Table 9-5 Prefered meeting style

Figure 9-5 Prefered meeting style

Figure 9-5 shows that 96% of the participants prefer the F2F meeting style and only 4% prefer the TS meeting style.

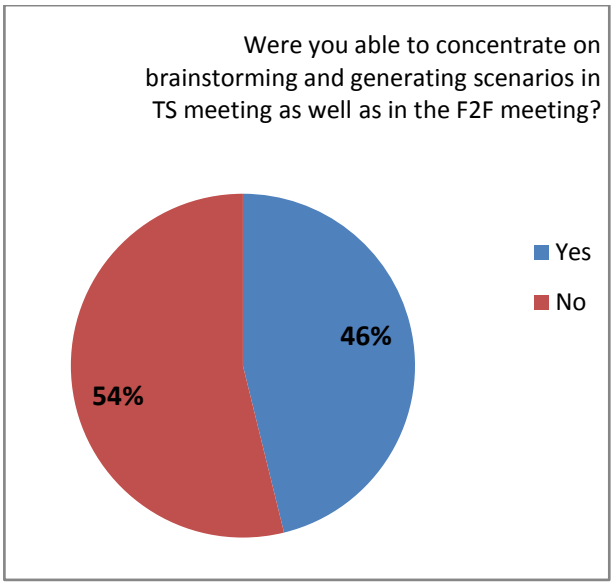


Figure 9-6 Concentration during TS meeting

Were you able to concentrate on brainstorming and generating scenarios in TS meeting as well as in the F2F meeting?

Answer	Number of Participants
Yes	24
No	28

Table 9-6 Concentration during TS meeting

Figure 9-6 shows that concentration on brainstorming and generating scenarios in general was no problem during the TS meating for 46% of the participants. 54% though had problems.

The feedback of the participants shows that they feel more comfortable with the F2F meeting style. They thought that it is more efficient and that chatting over a chat application has a negative effect on the meeting. They also would have needed much more time for their group meeting. This feedback substantiates our results from the evaluation – the F2F teams have better results than TS teams.

10 Conclusion

Software is all around us today. It is not only present in personal computers where we can see it and use it directly, but also in a lot of other things, we are not connecting with software. Software is running in cars, elevators, planes, electronic hand dryers, etc. and it is very important that this software is running well without errors. Some errors are not critical when they appear, for example when a hand dryer is not working. Some errors tough are critical when they appear – a crashing plane could be the result.

To ensure that software works the way it should it is important to follow a structured process of software engineering. By developing software step by step, using various software engineering processes, correctly working software is more likely than creating software without such processes. Sure, it also depends on the size of the software that has to be developed, but doing correct software engineering is no mistake – whether the software projects are big or small.

Also time and cost are very important factors in software engineering. Projects in general have to be in time and budget. Project Managers have to make sure that a software project is in time and the budget that is calculated is not exceeded. Therefore it is very important to ensure, that the software is working right and it is doing the things the customer wants the software to do. If it is not defined very clearly what the software should do, there could be big surprises in the end. When requirements of software products change during later phases of development, it gets difficult to implement those changed requirements. This results in higher costs and a longer project lifetime. The later such changes occur, the more expensive they are going to be. If the customer wants something to change in the software during the first phases of software development, where the planning of the software is done, it is not very critical to change requirements, because they can be built in easily. If the requirements are changing in later phases, where software actually is developed, lines of code are already written and modules are finished and integrated, it gets very difficult to implement the changed requirements, because already created pieces of software have to be changed and rebuilt.

So it is very important to define very clearly what the requirements are and what the customer expects the software to do, in the beginning of the project. If the customer wants the software to be changed later on and changes the requirements that have been accepted by both sides in the beginning of the project, then it is not that problematic for the project management. Then they have done their job right and if the customer wants the requirements to be changed he has to pay for that.

What is problematic though is, if the software is not fulfilling the wishes of the customer and does not meet the requirements defined in the beginning. Then it is the fault of the project team and the additional cost is not covered by the customer. So it is very important to ensure that the software product does exactly what it has to do and meets the requirements defined. But not only functional requirements that describe what software should do are important, but also non-functional requirements (also called quality requirements). They describe how software is working and how it is doing things. A non-functional requirement for example could be fast response times, whether there are only a few or hundreds of people using the software at the same time. These requirements are not to reach very easily. They cannot be implemented through lines of code directly. The whole software has to be developed and designed that way to fulfill non-functional requirements.

To make sure all functional and non-functional requirements of a software product are met correctly, it is important to design the architecture of a software product with care. The architecture describes the base of the software, like a blue print of a house. It describes how the software should work, what requirements it should meet, defines the modules and interfaces of the software product, etc. The architecture describes a software product in many different ways. That's why there are many different views people can have on software architecture. A programmer for example will have (and needs) a completely other view on the architecture than the project manager for example. The programmer is interested in modules, their connections, global variables, etc. The project manager on the other hand, wants to know when certain parts of the software product will be finished or which parts will be finished first.

So, it is very important that the architecture as a whole is designed as good as possible. To ensure that the architecture is the most optimal one possible, it has to be evaluated. Reviewing a software architecture takes a lot of time. One focus in software engineering is to shorten time for such evaluations without losing quality. In many cases different people involved in a software project come together and try to think about scenarios for the software architecture. Scenarios are well-known approaches to get a well-defined focus on what is needed and how the architecture of a software product should look like to meet the functional and non-functional requirements of the product. (A Scenario is a brief description of a single interaction of a stakeholder with a system. Scenarios are used to precisely define non-functional attributes, as mentioned in [39].)

To evaluate software architecture with scenarios as good as possible, the best and most helpful scenarios possible have to be found. There are many different methods of doing architecture evaluation like ATAM, SAAM, etc. Many studies have been done so far, evaluating what technique might be the best, but this area of software engineering is still under evaluation and the optimal method has not been found yet – the question is if there ever will be one best solution for architecture evaluation at all. There might be various methods suitable for different kinds of software projects.

But when focusing on methods that use scenarios for software architecture evaluation it is important to find a method that brings the best results. There are many possibilities to find scenarios during an architecture evaluation. Scenarios can be found by participants individually, which means that they think about the problem on their own and try to find scenarios alone. It is possible to let them find scenarios from their own view or from a completely opposite view. So a programmer could try to find scenarios from a programmers view or from a project manager's view on the other hand, which could be very useful, because a programmer might think of aspects, a project manager would never think of. It is also possible to do architecture evaluation in large groups of people where all are sitting together trying to find scenarios, which then will be discussed in the whole group. It is also possible the form smaller groups of people, 3 or 4 members for one team, and let many teams try to find scenarios and afterwards evaluate the results of all the teams.

The focus of this thesis is the team-meeting aspect of software architecture evaluation. These team meetings can be done in different ways. The team members can come together physically and try to find scenarios by discussing them face to face. This aspect costs money in the real world, because bringing together different people from different locations costs the companies money. It gets even more expensive when these group members are specialists and are coming from different continents. Such meetings are getting very expensive. The other possibility to do such team meetings

would be not meeting physically, but doing some kind of tool-supported communication. When doing tool-supported communication traveling costs can be minimized, because the team members do not have to meet in reality. The main question is: How do results from face to face communicating teams differ from teams doing tool-supported communication.

Empirical studies help to find those most convincing methods by rebuilding real circumstances of architecture evaluation in a controlled experiment. Our controlled experiment with 54 students from the Vienna University of Technology helped us to evaluate this and other questions. We let participants find scenarios for 2 different applications individually and in teams, where the teams have been separated into face to face (F2F) and tool-supported communicating teams (TS).

Based on the data we got from this controlled experiment we did find out that in every case F2F teams did find more and better scenarios than TS teams, based on our reference scenario profile, we have built upon the results we got from all participants. The results though were not that strong that it could easily said that F2F meetings are always better. When the traveling costs of realizing a F2F meeting are getting very high it might be better to do a TS meeting and accept weaker results, but to save a lot of money, because the members do not have to travel.

We also saw that communication in F2F teams is better than in TS teams, because we compared the individual phase with the team phase and learned that F2F teams had a higher increase of good scenarios in their scenario profiles than TS teams compared to the individual phase. But again, these positive communicating effects were not that strong to say that a physical meeting is always better then a virtual meeting.

We also tried to find a relationship between the number of scenarios found by teams and their experience level. Therefore we let every participant fill out an Experience Questionaire at the beginning of the controlled experiment where questions about software engineering, project management, software architecture evaluation, etc. have been asked. Based on this questionnaire we were able to calculate the experience of every team and tried to find a connection between the number of scenarios found and the experience of the teams. Unfortunately we did not find a connection between these 2 aspects.

What we learned with this experiment and its evaluation is that the participants need more time as we gave them for architecture evaluation – as individuals and as teams. 50% more time in every phase would be a good choice. Teams often had the problem that they could not discuss every scenario they had found individually in the team, so maybe the results and differences between F2F and TS teams would be different, if both teams would have more time for discussion.

We also learned that experience in software engineering topics does not guarantee good results in architecture evaluation. In general, it made no difference if the participant was master or a bachelor student, or if the participant had a lot of experience in software projects. The results were independent from those factors. When doing architecture evaluation, experience in this specific area can help. So it would be useful to teach the participants more in architecture evaluation topics before, than we did.

The third thing we learned was that the technique we used to evaluate the experience of the participants and teams did not lead to our desired result. The experience level we collected from every participant did not relate to the results of found scenarios. Like mentioned before, experience

in software engineering in general does not guarantee good results in architecture evaluation, but it should be considered to do a different approach in evaluating the experience of the participants in other similar experiments.

Another important aspect of future empirical studies in this area would be a higher number of participants. Especially when you try to evaluate team-meeting aspects, a high number of participants is necessary. In our controlled experiment we had the data of 19 teams. With this number you can do evaluations on the surface, like looking at F2F and TS team aspects. As soon as you try to go deeper by splitting those teams into “categories used” and “categories not used” ones for example, the number of teams is not enough to do exact statistical studies and statements.

As can be seen in our results and the feedback of our participants, the TS-meetings were not as good as the F2F-meetings. The results in TS-meetings have been weaker and also the participants did not like the TS-meeting style at all and found that it was too complicated and not as productive as the F2F-meeting style. This might be because of the chat-application we used. Doing discussions over a text-based chat application is difficult. An approach on further studies in this area would be using modern chat-applications that support audio and / or video conversation. By doing this the communication between the participants would be much better, which might result in better results and a higher acceptance from the participants.

When looking at our results, we see that we often have clear results for session 1 (Livenet) but no clear results for session 2 (Wiki System). One explanation might be the fact that we switched the group of participants who used change categories and the group who did not between session 1 and 2. The idea was, that for session 2 every participant should change meeting style and categories usage, so that everyone is doing evaluation the one way in session 1 and the other way in session 2. This approach should have given us a broad view (and results) of different evaluation techniques. But here comes the problem: The participants who did use categories in session 1 did not in session 2 and those who did not use them in session 1 did in session 2. This decision makes session 1 different from session 2. The group of participants who did use change categories in session 2 did not know them before and so could not use them in session 1, which is clear. But the participants who used them in session 1 and not in session 2 could remember the change categories from session 1 (though they have been partly different) and so could use their knowledge in session 2. So we had the situation in session 2 that every participant somehow knows about the change categories where in session 1 really only half of the participants knew about them. This makes results from session 2 different from results from session 1. The suggestion for further studies is, not to change participants that use change categories if the controlled experiment consists of more sessions and those sessions should deliver the same kind of data. The participants who get change categories as a guideline should use them throughout the whole experiment, so the experience and knowledge factor stays the same. Of course, if different sessions should help to evaluate the results compared to growing experience in architecture evaluation, changing categories groups could be helpful.

11 References

[1] Sommerville, I. *Software Engineering - 8th Edition*, Pearson Education, 2007

[2] Schach, S. R. *Classical and Object-Oriented Software Engineering – 3rd Edition,* Vanderbilt University, 1996, ISBN: 0256182981

[3] Royce, W. W. Managing the development of large software systems. *Proceedings, IEEE WESCON*, 1970, pp. 328-338

[4] Boehm, B. W. A spiral model of software development and enhancement. *Computer, Volume 21, Issue 5*, 1988, pp. 61-72

[5] Kruchten, P. *The Rational Unified Process – An Introduction – 3rd Edition,* Addison Wesley, 2003

[6] IEEE-STD 610.12-1990 - Standard Glossary of Software Engineering Terminology, 1990

[7] Bahsoon, R., Emmerich, W. Evaluating software architectures: development, stability, and evolution. *Computer Systems and Applications, 2003. Book of Abstracts. ACS/IEEE International Conference on*, 14-18 July 2003, pp. 47-

[8] Parnas, D.L., Weiss, D. Active Design Reviews: Principles and Practices. *Proceedings of the 8th International Conference on Software Engineering*, 1985, pp. 132-136

[9] Clements, P., Kazman, R., Klein, M. *Evaluating Software Architectures: Methods and Case Studies*, Addison Wesley, 2002

[10] Kazman, R., Klein, M., Barbacci, M., Lipson, H., Longstaff, T., Carriére, S.J. The Architecture Tradeoff Analysis Method. *Proceedings of the 4th ICECCS*, August 1998, pp. 68-78

[11] Kazman, R., Abowd, G., Bass, L., Webb, M. SAAM: A Method for Analyzing the Properties of Software Architectures. *Proceedings of the 16th International Conference on Software Engineering,* May 1994, pp. 81-90

[12] Clements, P. *Active Reviews for Intermediate Designs (CMU/SEI-2000-TN-009)*, Software Engineering Institute, Carnegie Mellon University, August 2000

[13] Klein, M., Kazman, R. Attribute-Based Architectural Styles. *Software Engineering Institute Technical Report CMU/SEI-99-TR-022*, Software Engineering Institute, Carnegie Mellon University, October 1999

[14] Kazman, R., Asundi, J., Klein, M. Quantifying the Costs and Benefits of Architectural Decisions. Proceedings of the 23rd International Conference on Software Engineering, May 2001, pp. 297-306

[15] Bengtsson, P., Lassing, N., Bosch, J., Van Vliet, H. Architecture-level modifiability analysis (ALMA). *The Journal of Systems and Software 69*, 2004, pp. 129-147

[16] Williams, L. G., Smith, C. U. PASA℠: A Method for the Performance Assessment of Software Architectures. *Proceedings of the Third International Workshop on Software and Performance*,

July 2002

[17] Abowd, G., Bass, L., Clements, P., Kazman, R., Northrop, L., Zaremski, A. Recommended Best Industrial Practice for Software Architecture Evaluation. *Software Engineering Institute Technical Report CMU/SEI-96-TR-025*, Software Engineering Institute, Carnegie Mellon University, January 1997

[18] Lassing, N., Bengtsson, P., Van Vliet, H., Bosch, J. Experiences with ALMA: Architecture-Level Modifiability Analysis. *The Journal of Systems and Software 61*, 2002, pp. 47-57

[19] Kruchten, P. Architectural Blueprints – The "4+1" View Model of Software Architecture. *IEEE Software Volume 12, Number 6*, November 1995, pp. 42-50

[20] Bengtsson, P., Bosch, J. An Experiment on Creating Scenario Profiles for Software Change. *Annals of Software Engineering, 9*, 2000, pp. 59-78

[21] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, A. Zaremski, Recommended Best Industrial Practice for Software Architecture Evaluation. *Software Engineering Institute Technical Report CMU/SEI-96-TR-025*, Software Engineering Institute, Carnegie Mellon University, January 1997

[22] Babar, M. A., Biffl, S., Eliciting Better Quality Architecture Evaluation Scenarios: A Controlled Experiment On Top-Down vs. Bottom-Up. *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, 2006, pp. 307-315

[23] Lawrence, G., Votta, Jr., Does Every Inspection Need A Meeting? *Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering*, 1993, pp. 107-114

[24] Bradac, M. G., Perry D. E., Votta L. G., Prototyping a Process Experiment. *Fifteenth International Conference on Software Engineering*, Baltimore, May 1993

[25] Keane, J. F., Keane, M., Teagan, M., *Productivity Management in the Development of Computer Applications*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1984

[26] Fagan, M. E., Design and Code Inspections to Reduce Errors in Program Development. *IBM Syst. J.*, Vol. 15, no. 3, 1976, pp. 182-211

[27] Kruchten, P., Architectural Blueprints—The "4+1" View Model of Software Architecture. IEEE Software 12 (6), 1995, pp. 42-50

[28] Garlan, D., Shaw, M., An Introduction to Software Architecture. *Advances in Software Engineering and Knowledge Engineering*, Vol. 1, World Scientific Publishing Co., 1993.

[29] Parnas, D. L., On the Design and Development of Program Families. *IEEE Transactions on Software Engineering*, Vol. 1, 1976, pp. 1-9

[30] Smith, C., *Performance Engineering of Software Systems*. Addison-Wesley, Reading, Ma., 1990

[31] Smith, C., Woodside, M., System Performance Evaluation: Methodologies and Applications. CRC Press, 1999

[32] Williams, L. G., Smith, C. U., Performance Evaluation of Software Architectures. *Proceedings of the Workshop on Software and Performance (WOSP98)*, Santa De, NM, 1998

[33] Bahsoon, R., Evaluating Software Architectures for Stability: A Real Options Approach. Research Abstract. *Proceedings of the 25th International Conference on Software Engineering*, Portland, USA, 2003

[34] Bahsoon, R., Emmerich, W., ArchOptions: A Real Options-Based Model for Predicting the Stability of Software Architecture. *Proceedings of the Fifth Workshop on Economics-Driven Software Engineering Research, EDSER 5*, 2003

[35] Höhn, R., Höppner, S., *Das V-Modell XT - Grundlagen, Methodik und Anwendungen: Grundlagen, Methodik Und Anwendungen*. Springer, 2008

[36] Kazman, R., Abowd, G., Bass, L., Clements, P., Scenario-Based Analysis of Software Architecture. *IEEE Software*, Nov. 1996, pp. 47-55

[37] Barbacci, M., Carriere, J., Kazman, R., Klein, M., Lipson, H., Longstaff, T., Weinstock, C., Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis. *CMU/SEI-97-TR-29, Software Engineering Institute*, Carnegie Mellon University, 1997

[38] Shaw, M., Garlan, D., *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996

[39] Bass, L., Clements, P., Kazman, R., *Software Architecture in Practice*, Addison Wesley, 1998

[40] Perry, D., Wolf, A., Foundations for the study of software architecture. *SIGSOFT Software Engineering Notes 17 (4)*, October 1992, pp. 40-52

[41] Kazman, R., Abowd, G., Bass, L., Webb, M., Analyzing the Properties of User Interface Software Architectures. *School of Computer Science Technical Report CMU-CS-93-201*, Carnegie Mellon University, 1993

[42] *Military Standard, Defense System Software Development (DOD-STD-2167A)*. United States Department of Defense, Washington D.C., 1988

[43] Pressman, R., *Software Engineering: A Practitioner's Approach*, 3rd edition, McGraw-Hill, New York, 1992

[44] Smith, C. U., Williams, L. G., *Performance Solutions: A Practical Guide to Creating Responsive, Scaleable Software*. Addison-Wesley, Reading, MA, 2002

[45] Smith, C. U., Williams, L. G., Performance Engineering Evaluation of COBRA-based Distributed Systems with SPEED. *Computer Performance Evaluation, Lecture Notes in Computer Science*, Vol. 1469, Springer, 1998, pp. 321-335

[46] Booch, G., Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide*. Reading, MA, Addison-Wesley, 1999

[47] Buschmann, F., Meunier, H., Rohnert, P., Sommerlad, P., Stal, M., *Pattern-Oriented Software*

Architecture: A System of Patterns. Chichester, England, John Wiley and Sons, 1996

[48] Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, Reading, MA, Addison-Wesley, 1995

[49] Schmidt, D., Stal, M., Rohnert, H., Buschmann, F., *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects.* Chichester, England, John Wiley and Sons, 2000

[50] Medvidovic, N., Taylor, R. N., A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering* 26 (1), 2000, pp. 70-93

[51] Hoffmann, H. P., UML 2.0-Based Systems Engineering Using a Model-Driven Development. *CrossTalk – The Journal of Defense Software Engineering*, Vol. 18, No. 11, November 2005, pp. 17-22

[52] Kazman, R., Barbacci, M., Klein, M., Carriere, S. J., Woods, S. G., *Experience with Performing Architecture Tradeoff Analysis*, Software Engineering Institute, Carnegie Mellon University, 1999

[53] Garlan, D., Shaw, M., *An Introduction to Software Architecture*, Carnegie Mellon University, 1993

[54] Tortora, G., Ambriola, V., *Advances in Software Engineering and Knowledge Engineering*, London, World Scientific Pub Co, 1993

[55] Mussbacher, G., Amyot, D., Mansurov, N., Understanding existing software with use case map scenarios. *Telecommunications and beyond: The Broader Applicability of SDL and MSC*, 2007, pp. 124-140

[56] Mussbacher, G., Amyot, D., *Introduction to use case maps*, 2001

[57] Amyot, D., *UCM quick tutorial version 1.0*, University of Ottawa, September 1999

[58] Barbacci, M., Klein, M., Longstaff, T., Weinstock, C., *Quality Attributes*, *CMU/SEI-95-TR-021, Software Engineering Institute*, Carnegie Mellon University, December 1995

[59] Basili, V., The role of controlled experiments in software engineering research. *Empirical Software Engineering Issues. Critical Assessment and Future Directions*, Berlin, Germany, Springer, 2007, pp. 33-37

[60] Bosch, J., Bengtsson, P., An experiment on creating scenario profiles for software change, *Annals of Software Engineering, Vol. 9, Issue 1-4*, 2000, pp. 59-78

[61] Winkler, D., Biffl, S., Babar, M. A., Impact of experience and team size on the quality of scenarios for architecture evaluation. *Proceedings of the 12th international conference on evaluation and assessment in software engineering*, 2008, pp. 1-10

[62] Koreimann, D., *Grundlagen der Software Entwicklung*, Munich, Oldenburg Verlag, 2000

[63] Van Vliet, H., De Bruin, H., *Scenario-Based Generation and Evaluation of Software Architectures*, Berlin, Springer, 2008, pp. 128-139

[64] Thaller, G., *Software Qualität*, Berlin, VDE Verlag, 2000

[65] Mayr, H., Projekt Engineering, Leipzig, Carl Hanser Verlag, 2001

[66] Soni, D., Hofmeister, C., Nord, R., *Applied Software Architecture: A Practical Guide for Software Designers*, Amsterdam, Addison-Wesley, 1999

[67] Obbink, H., Ionita, M., Hammer, D*., Scenario-Based Software Architecture Evaluation Methods: An overview*, The Netherlands

[68] Lundberg, L., Bosch, J., Häggander, D., Bengtsson, P., Quality attributes in software architecture design. *Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications*, 1999, pp. 353-362

[69] Biffl, S., *Qualitätssicherung - Script*, Vienna University of Technology, 2004

[70] *Rational Unified Process: Best Practices for Software Development Teams*, Rational Software White Paper, TP026B, Rev 11/01

[71] Jacobson, I., Booch, G., Rumbaugh, J., *The Unified Software Development Process*, Pearson Education, 2003

12 Appendix

12.1 TopRated-Scenario Profiles

12.1.1 Livenet

Scenario Number	Score	Scenario Description
36	45	Integration of external communication systems e.g. chat, video conference, VOIP, …
4	40	Management of access rights (e.g. changing role by user himself, different views on documents, resetting passwords, locking accounts)
7	38	Versioning (Undo-Function) – viewing old document version - viewing of changes (Wiki)
6	34	Scalability/Portability (e.g. distributed systems)
24	16	Livenet portable for mobile devices (small screens)
3	16	Converting to other file formats (pdf, …);
51	14	Save chat history for later reviews
11	14	Multiple access of many users to documents (working on it at the same time)
44	14	Automated notification about new/changed artifacts in group
278	13	Database enlargement (for big documents, forum, users, files,…)
97	13	New roles defined and implemented
19	11	Developing Offline-Mode – Saving locally – Synchronization
84	10	Support of various (new) evaluation techniques
15	9	Support of different browsers (should look the same everywhere)
20	9	AJAX or similar technology for better performance / efficiency
98	8	Authentication with ID-card instead of username/password
56	8	General information and some data of the users are commonly accessible - skills viewed by team leader for collecting team
87	8	Add calendar (including balancing of team-member appointments)
82	8	Document access/security - not every format should be uploadable - restrict

		access to certain users
2	8	Multilingual
94	7	Versioning (Undo-Function) – viewing old document version - viewing of changes (Wiki)
75	7	Comment function on artifacts (user wants to discuss artifact/annotation)
27	7	Automatic notification on mobile devices on certain events
189	7	Supporting different kinds of media files, e.g. pictures, videos, sounds (within text)
31	6	Review mode before general publishing
33	6	Permission management (for groups of users)
89	6	Adding of architectural views (participants can select their view or view can be assigned to its role)
23	6	Replace Database with faster and better performing one
13	5	Scalability of Email-Server / Should handle more traffic and higher number of mails / mailing list
30	5	Modify download-management (single files, multiple files, folders, …)
39	5	Adopting system to up to date standards of Browser (IEEE)

12.1.2 Wiki

Scenario Number	Score	Scenario Description
189	56	Supporting different kinds of media files, e.g. pictures, videos, sounds (within text)
186	43	Online editor for graphics, forms, texts, … (tinyMCE (WYSIWYG-editor), possibility to 'pimp' content)
4	33	Management of access rights (e.g. changing role by user himself, different views on documents, resetting passwords, locking accounts)
7	31	Versioning (Undo-Function) – viewing old document version - viewing of changes (Wiki)

94	25	Versioning (Undo-Function) – viewing old document version - viewing of changes (Wiki)
257	24	Subscribing to change history through RSS feed (other media possible: sms, Skype,…)
184	21	Possibility to rate articles and average rating for one author should be visible
34	14	Implement search function based on meta-data (xml)
3	13	Converting to other file formats (pdf, …);
2	13	Multilingual
203	12	Ban user
6	12	Scalability/Portability (e.g. distributed systems)
258	10	Using HTML and CSS tags for writing articles, also LaTeX
197	10	Detailed settings for notification (promptly, periodically, single or collected, subscribe to certain categories only, …)
20	9	AJAX or similar technology for better performance / efficiency
334	9	Make pages non editable / lock them for specific users and or IP addresses
202	9	Support for adding attachments to pages
242	8	RSS-Feed Integration
200	8	Make pages non editable / static
230	8	Content translation into other languages (automatically)
325	7	Intelligent search functions (like Amazon) logging of individual search key words + data mining
182	7	Implementation of chat room and / or forum
376	7	If an user edits an article, it is marked as locked
294	7	Display change history in 2 columns to compare changes better
369	7	List of all recent changes
190	6	Possibility to change settings in change history
296	6	Preview function on editing articles/pages in Wiki
155	6	Customizable user interface for every user

212	6	Bad word filter
56	6	General information and some data of the users are commonly accessible - skills viewed by team leader for collecting team
390	6	Overview of link structure of different articles
154	5	Implement auto-saving function
174	5	Check external links if they are dead links
355	5	Include spam protection
68	5	Faster server, database and data lines
265	5	Improve user profiles to help building communities (E-Mail, pictures [e.g. portrait], descriptions, ...)
31	5	Review mode before general publishing
11	5	Multiple access of many users to documents (working on it at the same time)
199	5	Spell checking functionality (also automatically)

12.2 Questionaires

Experience Questionaire		
Student-ID		
Name		
LVA	☐ QS-VU	☐ Testen
Session	☐ Tuesday, June 3rd	☐ Friday, June 6th

Please answer the questions in this questionnaire.

Note, that your answers will be treated anonymously. Your ID and name will be used to link your answers to your scenario reports. We will remove all personal data gathered for evaluation purposes afterwards.

1. Demographic Data

Age		Gender	

2. How do you rate your ability to understand English documents?

no experience						excellent
	0	1	2	3	4	

3. How many programming courses have you attended?

4. Estimate your general software engineering experience.

no experience						excellent
	0	1	2	3	4	

5. Estimate your project management experience.

no experience						excellent
	0	1	2	3	4	

6. Estimate your experience in software development (implementation).

no experience						excellent
	0	1	2	3	4	

7. Estimate your experience in developing software in a professional environment (e.g., working in a company).

no experience						Excellent
	0	1	2	3	4	

8. Estimate the time, you are participating in professional software engineering projects (in months / years).

9. Estimate the size of the largest project, you participated in (in person months/years)

10. Estimate your experience in quality assurance.

no experience						excellent
	0	1	2	3	4	

11. Estimate your experience in quality assurance in a professional environment (e.g., working in a company).

no experience						excellent
	0	1	2	3	4	

12. Estimate your experience with requirements and design documents.

no experience						excellent
	0	1	2	3	4	

13. Estimate your experience with Use Cases.

no experience						excellent
	0	1	2	3	4	

14. Estimate your experience in Software Design review / inspection?

no experience						excellent
	0	1	2	3	4	

15. Estimate your experience with architecture reviews.

no experience						excellent
	0	1	2	3	4	

16. Estimate your experience in developing of scenarios.

no experience						excellent
	0	1	2	3	4	

17. Estimate your experience in using collaborative tools.

no experience						excellent
	0	1	2	3	4	

18. Estimate your experience in chatting (skype, ICQ, etc.)

no experience						excellent
	0	1	2	3	4	

Please use the backside of this page for additional comments.

Thank you for your contribution!

Team Feedback Questionnaire (F2F)		
Student-ID		
Name		
Team-ID		
LVA	☐ QS-VU	☐ Testen
Session	☐ Tuesday, June 3rd	☐ Friday, June 6th
Application	☐ LiveNet	☐ Wiki
Scenario Categories	☐ Used	☐ Not used

Please answer the questions in this questionnaire.

Note, that your answers will be treated anonymously. Your ID and name will be used to link your answers to your scenario reports. We will remove all personal data gathered for evaluation purposes afterwards.

1. How many team-members did you know prior to this meeting?

0	1	2

2. Did you have enough time for the team meeting?

Yes ☐ (0) No ☐ (1)

3. If not, how much additional time would you need (in minutes)?

4. Did you follow the instructions (guidelines) of the applied method during the execution?

Never						All the time
	0	1	2	3	4	

5. Have the instructions been helpful to you? If not, please provide suggestions for improvement on the backside of this sheet.

No						Yes
	0	1	2	3	4	

6. Do you think that a team meeting is helpful for scenario elicitation?

No						Yes
	0	1	2	3	4	

7. How satisfied are you with the team meeting in general?

Unsatisfied						Satisfied
	0	1	2	3	4	

8. How satisfied are you with the discussion in today's team meeting?

Unsatisfied						Satisfied
	0	1	2	3	4	

9. How satisfied are you with the discussion outcome of today's team meeting?

Unsatisfied						Satisfied
	0	1	2	3	4	

10. How effective was the communication during the meeting?

Not effective						Very effective
	0	1	2	3	4	

11. Please rate the communication process you used for generating scenarios in your team:

	0	1	2	3	4	
Impersonal						Personal
Bad						Good
Difficult						Easy
Informal						Formal
Simple						Complex

12. Please describe the **conversation** you had with your team members for discussing scenarios:

0 .. Strongly disagree 4 .. Strongly Agree	0	1	2	3	4
Pleasant					
Agreeable					
Interesting					
Argumentative					
Cooperative					

Please use this page for additional comments on the Face-To-Face team meeting

Thank you for your contribution!

Team Feedback Questionnaire (Tool)		
Student-ID		
Name		
Team-ID		
LVA	☐ QS-VU	☐ Testen
Session	☐ Tuesday, June 3rd	☐ Friday, June 6th
Application	☐ LiveNet	☐ Wiki
Scenario Categories	☐ Used	☐ Not used

Please answer the questions in this questionnaire.

Note, that your answers will be treated anonymously. Your ID and name will be used to link your answers to your scenario reports. We will remove all personal data gathered for evaluation purposes afterwards.

1. How many team-members did you know prior to this meeting?

0	1	2

2. Did you have enough time for the team meeting?

Yes ☐ (0) No ☐ (1)

3. If not, how much additional time would you need (in minutes)?

4. Did you follow the instructions (guidelines) of the applied method during the execution?

Never						All the time
	0	1	2	3	4	

5. Have the instructions been helpful to you? If not, please provide suggestions for improvement on the backside of this sheet.

No						Yes
	0	1	2	3	4	

6. Do you think that a team meeting is helpful for scenario elicitation?

No						Yes
	0	1	2	3	4	

7. How satisfied are you with the team meeting in general?

Unsatisfied						Satisfied
	0	1	2	3	4	

8. How satisfied are you with the discussion in today's team meeting?

Unsatisfied						Satisfied
	0	1	2	3	4	

9. How satisfied are you with the discussion outcome of today's team meeting?

Unsatisfied						Satisfied
	0	1	2	3	4	

10. How helpful do you believe the collaboration tool was to discuss scenarios in the meeting?

Not helpful						Helpful
	0	1	2	3	4	

11. How well could you interact with the remote participants in the meeting?

Not well						Very well
	0	1	2	3	4	

12. How effective was the communication during the meeting?

Not effective						Very effective
	0	1	2	3	4	

13. Please rate the communication process you used for generating scenarios in your team:

	0	1	2	3	4	
Impersonal						Personal
Bad						Good
Difficult						Easy
Informal						Formal
Simple						Complex

14. Please describe the **conversation** you had with your team members for discussing scenarios:

0 .. Strongly disagree 4 .. Strongly Agree	0	1	2	3	4
Pleasant					
Agreeable					
Interesting					
Argumentative					
Cooperative					

15. Did you find LiveNet useful for the team-meeting?

No						Yes
	0	1	2	3	4	

16. Would you apply a tool like LiveNet in the future?

No						Yes
	0	1	2	3	4	

Please use this page for additional comments on the Tool-Supported team meeting and suggestions for improvement.

Thank you for your contribution!

Final Questionnaire		
Student-ID		
Name		
LVA	☐ QS-VU	☐ Testen
Session	☐ Tuesday, June 3rd	☐ Friday, June 6th
Scenario Categories	☐ Used	☐ Not used

Please answer the questions in this questionnaire.

Note, that your answers will be treated anonymously. Your ID and name will be used to link your answers to your scenario reports. We will remove all personal data gathered for evaluation purposes afterwards.

1. Overall, did you feel **you** performed well in developing scenarios for non-functional requirements in:

	a distributed arrangement using the collaborative tool.
	both arrangements.
	a face-to-face arrangement.

2. Overall, did you feel your **group** performed well in developing scenarios for non-functional requirements in:

	a distributed arrangement using the collaborative tool.
	both arrangements.
	a face-to-face arrangement.

3. Did you feel that using the collaborative tool had any positive or negative affect on your **group discussion**? e.g., you may have been able to discuss issues more quickly (a positive effect) or you may have found it more difficult to discuss issues (a negative effect).

	Large positive effect
	Small positive effect
	No effect
	Small negative effect
	Large negative effect

4. Compared with face-to-face group meeting, do you feel that a collaborative tool based group meeting is

	more efficient?
	equally efficient?
	less efficient?

5. Do you like to contribute your opinion on sensitive issues during a meeting **anonymously** if its possible?

	Yes		No

6. Overall, what **type of meeting** you would like for generating scenarios, a face-to-face or distributed arrangement using a collaborative tool?

	Face-to-face arrangement
	Distributed arrangement using collaborative tool

7. While using collaborative tool to generate scenarios, were you able to **concentrate** on brainstorming and generating scenarios as well as in the face-to-face meeting?

	Yes		No

8. Please provide some comments of your experience regarding the following topics (see also backside of this questionnaire).

   a) Describe the effect that the collaborative tool had on your group meeting compared to a face-to-face meeting.

b) Were there any aspects of the **face-to-face** meeting that **facilitated/hindered** efficient scenario discussion?

c) Were there any aspects of the **distributed meeting** that **facilitated/hindered** efficient scenario discussion?

d) How can we **improve the efficiency and effectiveness** of the distributed meeting arrangement supported by a collaborative tool?

e) What are the **three most difficult issues** you faced while developing scenarios to specify non-functional requirements during this exercise?

Thank you for your contribution!