Institut für Softwaretechnologie

Abteilung Software Engineering

Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart

Bachelorarbeit

Evaluating and Improving a Scenario-based Analysis Method for Service-based Systems

Patrick Koss

Studiengang: B.Sc. Softwaretechnik

Prüfer: Prof. Dr. Stefan Wagner

Betreuer: Justus Bogner, M.Sc.

begonnen am: 19.12.2018

beendet am: 19.06.2019

Abstract

Modifiability and maintenance are important topics in software development. Many metrics, such as McCabe's cyclomatic complexity, therefore attempt to quantify the maintainability of software quality attributes, in particular the sub characteristic modifiability. Quantifying specific changes in systems is not an easy task. With scenario-based methods, certain changes can be described. However, this requires a high manual effort. A previously developed tool-supported method attempted to estimate the difficulty and complexity of changes by focusing on service-based systems and using a scenario-based assessment. Because the quantification of specific changes to a system cannot be fully automated, the tool-assisted approach has tried to be lightweight, so it does not require much manual effort.

This work focuses on improving and evaluating the existing tool-supported method. We collected information about the technical background, then analyzed the existing tool-supported method and made a list of improvements. Some of them have been implemented. In a next step, we analyzed the now improved tool-supported method by conducting hands-on-interviews with seven participants to give a qualitative assessment, and by conducting a survey with 40 participants to provide a quantitative assessment.

Our most important finding in the evaluation is that the survey participants found the lines of code estimation is neither very accurate nor applicable and therefore not suitable for our tool-supported method according to the participants of the survey. Story Points seem to be best suited as an effort estimation method for our five predefined estimation methods according to the participants of the survey. We found no correlation between the personal background of a participant, whether a participant has worked with scenario-based evaluation in real projects, or his familiarity with service-based systems and our effort estimation methods and our evaluation functions. Our findings from the hands-on interviews have been to improve the ease of use for creating scenarios, to better visualize the evaluation features and to determine which evaluation features were helpful and which not.

After collecting a derived improvement list and implementing some suggestions, we believe that our tool-supported method can be used to estimate the effort of changes in a real service-based system, as participants, as our review notes, find the evaluation view and evaluation features helpful.

Keywords: scenario-based evaluation method, lightweight, service-based systems, modifiability, maintainability, tool-supported method

Table of Contents

List of Figures

List of Tables

1 Introduction

In today's software development practice, maintenance is becoming an increasingly important topic. New technologies and market changes are constantly taking place, so the released software has to be constantly maintained.

This section shows the motivation of the thesis, its scope and research objectives and method of research.

1.1 Motivation

According to *Stephen R. Schach* in his classic book "Object-Oriented and Classic Software Engineering" (Schach, 2011), maintenance is the biggest cost of all software development costs, such as designing the system, coding or unit testing. Between 1976 and 1981 maintenance costs accounted for 67% of all software costs. Between 1992 and 1998, maintenance costs already accounted for 75% of all software costs. There may be a tendency for these costs to increase in the future.

Since maintenance costs account for a large part of all software development costs, it is desirable to be able to easily adapt the architecture to future changes. One also wants to estimate the cost of these changes as they are very interesting for different stakeholders. For managers, it is common to be interested in the time and money involved in the software change. For developers, it might be interesting how easy it is to implement a change.

To measure the quality of software, especially maintainability, metrics can be applied. The metrics can be used to automatically and objectively evaluate a quality attribute of a software. *Ostberg and Wagner* (Ostberg & Wagner, 2014) listed metrics that well-represented in the literature and often implemented in various tools. One metric, for example, is McCabe's cyclomatic complexity, which analyzes the code for its complexity. An attempt is made to quantify the maintainability of the quality feature, since less complexity means easier modifiability. The result of cyclomatic complexity is a number that reflects whether a human being can still understand the code or is too complex to understand it. The less complex the code, the easier it is to understand and therefore easier to change.

Creating and evaluating specific changes requires considerable manual effort. Scenario-based evaluation methods, such as SAAM, describe what steps are required to identify scenarios and to evaluate these scenarios in relation to the current software architecture.

Applying an automatic method for specific changes saves time, reduces manual effort, and simplifies evaluation. Therefore, such an automatic method is desirable. For most software architectures, it is not possible to reduce the manual effort required to evaluate specific changes in the architecture or function of the software. For service-oriented systems, this may be possible. In a previous thesis, a lightweight, tool-based method for evaluating service-based systems with scenario-based assessment was created. This method attempts to exploit the features of service-based systems to make scenario-based assessment easier and less time-consuming compared to manual evaluation.

This thesis focuses on improving and evaluating the lightweight and tool-supported method.

- Can we make even better use of the features of service-based systems to create and evaluate a system and scenarios in our tool?
- Can we give more information about the rating in our tool to find flaws in the architecture of the service-based system?
- Which methods of estimating the effort of change are helpful to the users of our tool? So far, one can estimate with hours. But what about additional estimation methods?

1.2 Scope and Research Objectives

This thesis has two main goals. First, improve on the existing, lightweight and tool-supported method. We try to improve the usability of the tool. The evaluation of the method becomes more meaningful by attempting to add additional attributes related to service-based systems and effort estimation techniques. Second, evaluate the improved, lightweight, and tool-supported method in terms of usability, how useful it is to evaluate service-based systems, and how well the effort estimate can be applied in our context.

In the previous work, a meta-model for the change of service-based systems was developed to quantify the qualitative assessment of the scenario-based evaluation. Based on this, it can be improved. This work should therefore answer the questions:

- What features of service-based systems can be used to improve the metamodel?
- What parts of eliciting scenarios can be improved and still be lightweight?
- What information can be given to the user when evaluating service-based systems?

- What cost estimations methods are useful for the user?

After the improvement of the tool and the implemented meta-model the tool will be evaluated. This is done to achieve the second main objective of this work.

1.3 Research Method

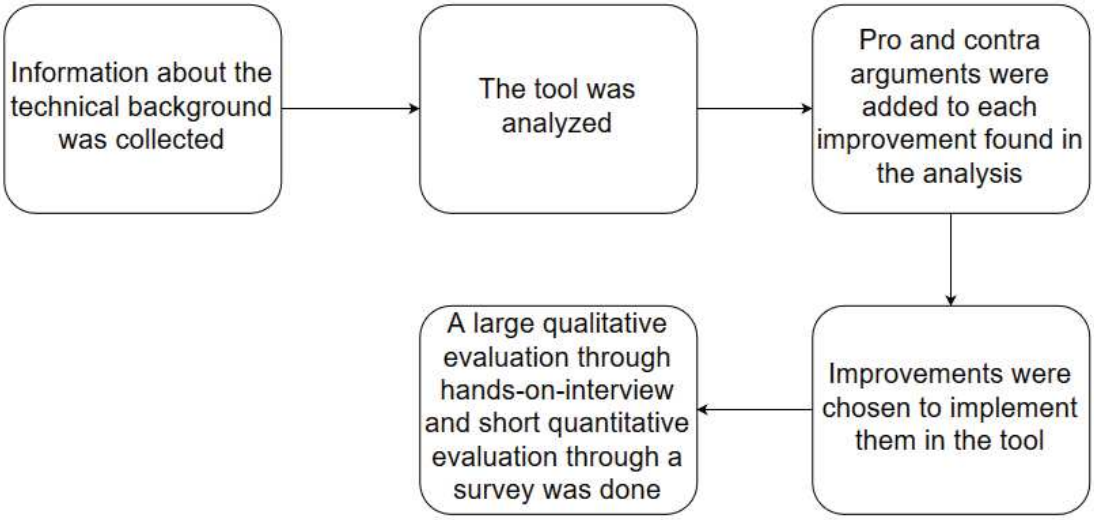


Figure 1 Proceed model

To improve the tool, the following steps were performed. First, information about service-based systems was collected to understand the characteristics that make the process of creating and evaluating a system created in the tool easier and more meaningful. In the same step, knowledge about scenario-based evaluation was acquired to know what is important when creating scenarios in the existing tool.

After acquiring information about the technical background, the tool was analyzed in the second step. Special attention was paid to ease of use to make the use of the tool more intuitive. Another big part of the analysis was the evaluation features of the tool to get more valuable information about how the system responds to the created scenarios. For future improvements to the tool, it was helpful to analyze the existing code and see if it could be rewritten to make it easy to modify. The last big point for improvement was the cost estimation methods. The result of this step was a list of improvements.

In the third step, a description and pros and cons arguments were added to each improvement to decide which improvement to implement and which not to do in the next step.

In the last step of the phase, some improvements were selected and implemented in the tool. They were selected according to the practicability in the time of the bachelor thesis and the added value that the improvement brings to the user. The benefits of improvement must outweigh its counter-arguments.

To evaluate the improved method and its tool support, a survey and hands-on interviews were conducted. The main objective of the survey was to evaluate the cost estimation methods and to find out the usefulness and importance of the evaluation features in the improved method and its tool support. The survey is a short quantitative evaluation with many participants. The aim of the hands-on interviews was to assess the usability of the tool, its ease, its support in creating scenarios, and the importance of rating attributes. The hands-on interviews are a comprehensive qualitative assessment with a small number of participants.

2 Technical Background

This chapter discusses the technical background needed to understand the context of the method and tool. In detail, service-based systems are described in the first part of the chapter, and scenario-based evaluation procedures are explained in the second part.

2.1 Service-Based Systems

Service-based systems are systems that place heavy emphasis on services as the primary architecture component used to implement and perform business and nonbusiness functionality. Representative of service-based systems are service-oriented architecture (SOA) and microservice-based systems. The two representatives are very different architecture styles, but they share many characteristics. (Richards, 2016)

All service-based systems share some features in common (Richards, 2016):

- They are distributed systems. This means that service components are accessed remotely via a remote access protocol. Examples of such a remote access protocol are Representational State Transfer (REST), SOAP (Simple Object Access Protocol) or Java Message Service (JMS).
- They are modular. In the context of service-based architecture, modularity means encapsulating parts of the large application into stand-alone services that can be individually designed, developed, tested and deployed without depending on other components or services. This feature can improve the maintenance of the architecture by allowing individual services to be maintained rather than replacing the whole application in a big-bang approach.
- They use service contract models. For service-based systems, two basic types of service contract models can be used. The models are service-based contracts in which the service is the sole owner of the contract and is generally free to develop and modify the contract without considering the needs of service users, and Consumer-driven contracts, which have close cooperation between the service owner and the service consumer, so that the needs of the service consumer are taken into account in the contracts that bind them.
- They use BASE (basic availability, soft-state, eventual consistency) transactions. Basic availability in the context of service-based systems means that the service works most of the time. Soft-State states that storage does not have to be write-

consistent, and that different replicas do not need to be consistently consistent. Eventual consistency indicates that stores have consistency at a later time.

- They have two basic types of service classifications. Service Type and Business Area. The classification refers to the role that the service plays in architecture. Service type classification refers to the type of role the service plays in the overall architecture. Business area classification refers to a role that an enterprise service plays in relation to a particular business functional area.
- Each service of a service-based system has a certain granularity, which depends on the specific representative of a service-based system. For example, microservice-based systems have small, fine-grained services, which means that a service generally serves a single purpose and does one thing well. In comparison, service-oriented architecture includes service components ranging in size from small application services to very large enterprise services. According to *Richards* (Richards, 2016), "it is common to have a service component within an SOA represented by a large product or even a subsystem." This means that services in SOA are often rather coarse-grained.

2.2 Scenario-Based Evaluation Methods

One goal of scenario-based evaluation is to evaluate a quality attribute of the software architecture like reliability, security or maintainability, by means of scenarios. Another goal of the scenario-based evaluation is, for example, the analysis of the software architecture. There are many different approaches to the evaluation of quality attributes, e.g. scenarios, simulation, mathematical modeling, and experience-based reasoning. For each quality attribute, another approach is helpful. Scenarios are well-suited for the analysis of development-related software qualities, as they combine individual interpretations of a software quality into a common view. Software qualities such as maintainability, which is the focus of this work, can be well expressed in a change scenario. (Bengtsson & Bosch, 2000)

A question arises here. What are scenarios? *Tekinerdogan* (Tekinerdoğan) answers it with “A scenario is considered to be a brief description of some anticipated or desired use of the system. Scenarios that can be directly supported by the architecture(s) are called direct

scenarios. Scenarios that require the redesign of the architecture are called indirect scenarios."

Scenarios can be used not only to evaluate a quality attribute, but also to check the software architecture for changes in the system. Scenario-based evaluation methods are indeed an established approach to architecture design evaluation, as they perform a structured evaluation to verify that certain objectives are met and to analyze the decisions required to do so. (Kazman, Bass, Abowd, & Webb, 2002)

It should be noted, however, that scenario-based evaluation is not widespread in the industry. *Woods et al* (Woods, 2012) sees this for a number of reasons. One of them is that the techniques are complicated and expensive to use. Another lacks confidence in the benefits of such reviews. The last reason he draws from his experience is that most methods focus on evaluating an abstract architecture, rather than examining system implementation as part of the process.

There are many methods in the literature which use a scenario-based rating to evaluate the software architecture. The methods are: Software Architecture Analysis Method (SAAM), Architecture Level Modifiability Analysis (ALMA), Software Architecture Performance Evaluation (PASA), and Architecture Trade-Off Analysis (ATAM). Each method focuses on a different goal, which is described in the next paragraphs.

The main goals of SAAM are the evaluation of the software architecture based on the required quality attribute, but also the comparison of different software architectures with regard to given properties. SAAM was originally developed for modifiability but is now also used for various quality features. SAAM comprises six activities: scenario development, description of the software architecture, classification and prioritization of scenarios, evaluation of individual scenarios, scenario interaction and overall evaluation. (Babar & Gorton, 2004; Kazman, Bass, Abowd, & Webb, 2002)

ALMA is a goal-oriented assessment, i.e. a goal is first determined and then analyzes the architecture towards that goal. ALMA can be applied from top to bottom and from bottom to top. Top-down launches form a predefined scenario classification. Bottom-up on the opposite side starts with concrete scenarios and the setting up of categories of scenarios. The method uses impact analysis to evaluate the software architecture against change scenarios. Impact analysis is performed by identifying the components affected by the scenarios, working out the necessary modifications, and determining the effects of

ripples. The results are interpreted according to the evaluation objective. (Bengtsson P. , 2002; Babar & Gorton, 2004)

The main goal of ATAM is a structured argumentation towards analyzing a software architecture capability with focus on multiple quality attributes. It also helps to make trade-offs between competing attributes. Unlike ALMA, which is most effective during the design phase or during software development, ATAM is most effective when applied at the final stage of a software architecture. The outputs of ATAM are lists of scenarios, sensitivity points, trade-off points and risks. (Rick Kazman, Mark Klein, ATAM: Method for Architecture Evaluation, 2000; Babar & Gorton, 2004)

The main objective of PASA is to assess the ability of candidate software architectures to address the performance goals of a system. PASA leads the software architecture analysis on the basis of performance-related scenarios. In addition, PASA also considers quality attributes during the analysis as well as trade-offs that need to be made. It is also used to compare different software architectures. Unlike ALMA and ATAM, which are best applied at a particular stage of development, PASA can be applied early in the development cycle, after deployment, or during an upgrade of a legacy system. (Babar & Gorton, 2004)

The last paragraph of this section introduces SAAM and its detailed steps to evaluate a software architecture to understand how a scenario-based evaluation method works. According to *Takinerodogan* (Tekinerdoğan) SAAM works like this:

"SAAM takes as input a problem description, requirements statement and architecture descriptions. The steps of SAAM are as follows:

1. Describe candidate architecture: The candidate architecture is described which includes the system's computation and data components, as well as all component relationships, sometimes called connectors.
2. Develop scenarios: Development of scenarios for various stakeholders; the scenarios illustrate the kinds of activities the system must support and the anticipated changes that will be made to the system over time.
3. Perform scenario evaluations: Scenarios are categorized into direct and indirect scenarios. For each indirect task scenario, the required changes to the architecture are listed and the cost of performing these changes is estimated. A modification to the architecture means that either a new component or connection is introduced, or an existing component or connection requires a change in its specification.

4. Reveal scenario interaction: Different indirect scenarios that require changes to the same components or connections are said to interact at the corresponding component. Determining scenario interaction is a process of identifying scenarios that affect a common set of components. Scenario interaction measures the extent to which the architecture supports an appropriate separation of concerns. Semantically close scenarios should interact at the same component. Semantically distinct scenarios that interact indicate an improper decomposition.

5. Overall evaluation: Finally, each scenario and the scenario interactions are weighted in terms of their relative importance and this weighting used to determine an overall ranking. The weighting chosen will reflect the relative importance of the quality factors that the scenarios manifest."

3 Related Work

Since one of the goals of this work is to improve the existing tool, the focus is on the literature that has attempted to solve similar issues, use the metrics for cost estimation, and evaluate the software architecture.

A method that is also lightweight and focuses on evaluating software architecture with scenarios is MORPHOSIS (Koziolek, Domis, Goldschmidt, Vorst, & Weiss, 2012). It analyzes the architecture in terms of a number of future development scenarios. In addition, it integrates the enforcement of the architecture in the build process of the system. Finally, a report framework for several novel code metrics at the architectural level is created from recent literature. It performs an evolution scenario analysis according to an extended version of the ALMA method. It combines a top-down and bottom-up scenario. To determine the amount of affected source code and possible ripple effects, they used dependency tools such as NDepend and CppDepend. Its scenario description template consists of 13 items. Some points describe business goals of a scenario, five-year occurrence probability and abstraction of the cost estimation on a scale of 1 to 10 for implementing the scenario. Using the template information, MORPHOSIS can create a diagram that provides an overview of evolutionary scenarios. The size of each bubble indicates the impact of the scenario on the architecture and thus correlates with estimated costs (Figure 2). Remarkably, they use an ordinal scale for cost estimation because they did not have enough data to make reasonable estimates through lines of code.

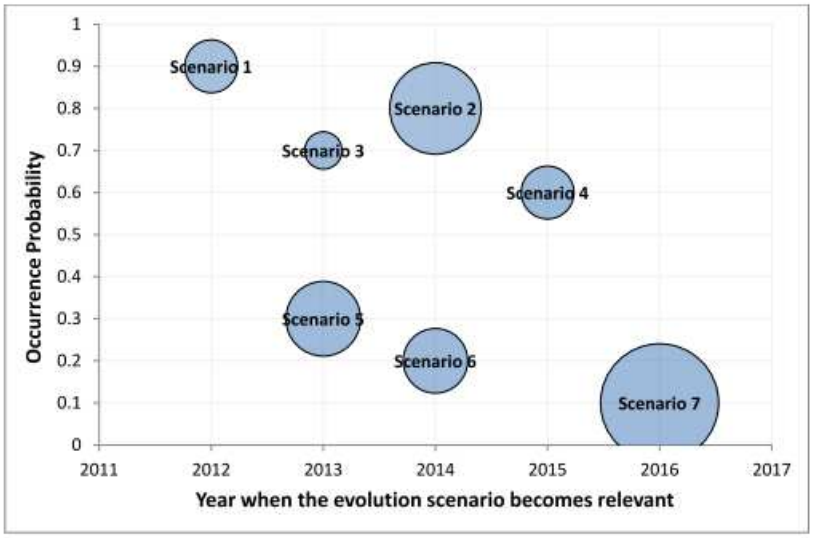


Figure 2 Evolution Scenario Ranking (Koziolek, Domis, Goldschmidt, Vorst, & Weiss, 2012)

One tool for analyzing the change propagation caused by a change request in the software architecture model is KAMP (Karlsruhe Architectural Maintainability Prediction) (Rostami, Stammel, Heinrich, & Reussner, 2015). With KAMP, software architects can model the initial architecture with annotated context information, which includes technical and organizational tasks called the base architecture, and the architecture after implementing the change request called the target architecture. Then, the tool calculates the differences between the base and target architecture models and generates task lists by applying derivation and interpretation rules to the architectural model. In addition to structurally passing architecture changes, KAMP addresses all key workspaces, including tasks such as test development and execution, build configuration, deployment, and related artifacts that must be addressed throughout the software lifecycle. KAMP creates higher quality and more homogenous task lists compared to manual analysis.

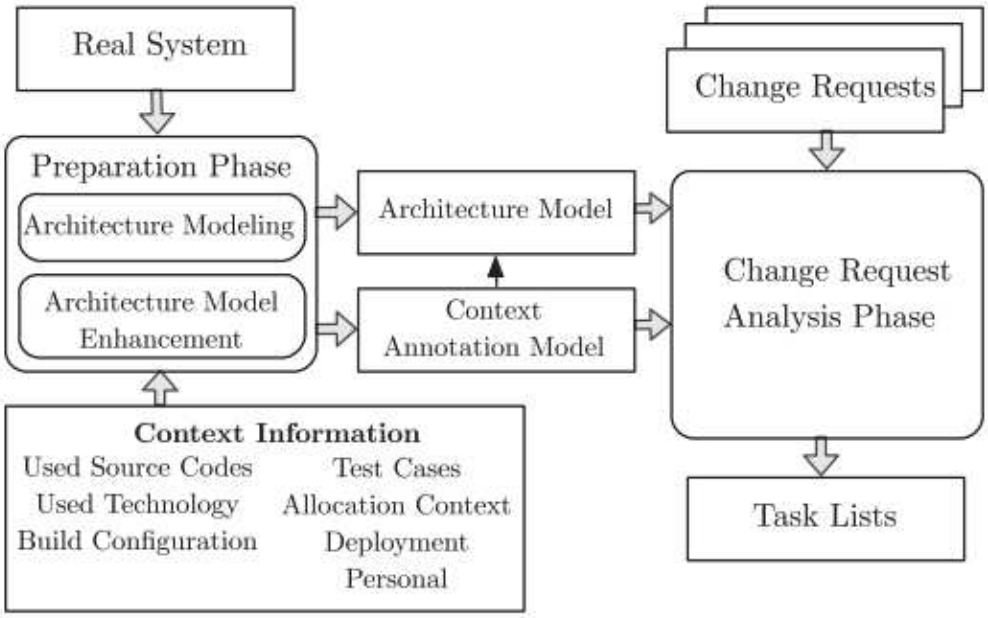


Figure 3 KAMP Overview (Rostami, Stammel, Heinrich, & Reussner, 2015)

The main contribution of their approach is the automatic evaluation of context information and identification of tasks required to implement a change request.

The next part of the related work deals with various approaches to estimating change costs.

ALMA (Bengtsson P. , 2002) uses a maintenance effort equation to calculate the total cost of changing the system in the identified scenarios. The equation is:

$$E_M = \frac{\sum_I((\sum_{CC_i} s_j) * P_{CC} + (\sum_{NP_i} s_j) * P_P + (\sum_{NC_i} s_j) * P_{nc})}{C(MP)} * CT$$

E_M is the maintenance effort.

C(MP) is the cardinality of the scenario set as well as the number of the scenarios in the set.

CCi is the set of components that must be changed to insert the change scenario, including a size estimate for the required change in the code lines. NPi and NCi contain similar information for new plug-ins and new components.

CT is the expected number of changes, such as the estimated change traffic to be predicted for the time period.

$P_{CC,}$ P_P and P_{NC} are weights that reflect the productivity difference between activities, write new code, change code, and write a plugin. *Bengtson et al.* (Bengtsson P. , 2002) indicates $P_{NC} > P_P >> P_{CC.}$ Bengtson cites Maxwell that the productivity of writing new code (> 250 LOC / person-month) is six times higher than the modification code (<40 LOC / person-month).

Another cost estimation model is the Constructive Cost Model (COCOMO) 2 (Boehm, et al., 1995). It is used for software project investment decisions, project budgeting and timing, discussion of costs and schedules, and risk decision making. This model consists of three sub models. First, the application composition model that estimates the first size estimate based on object points and a formula for size and productivity. Second, the early design model that applies when all system requirements are met. In this phase, the effort is estimated in function points. Third, the model for the architecture that is applicable when designing the system architecture. Many multipliers influence the estimate at this stage. Since this work assumes that all services are set, we will take a closer look at the post architectural model. In this phase a person month is calculated according to the following formula:

$PM = A*size^E*M$ with $M = cost\ drivers$

Boehm et al (Boehm, et al., 1995) recommends that A be a coefficient of 2.5. The size is measured in KSLOC (kilo source code lines), and E is an exponent that should represent the growing overhead of increasing project size. The value for E is between 1.01 and 1.26. There are 17 cost drivers in this phase, divided into 4 categories. Each cost driver is judged on an ordinal scale of -2 to 3 how well cost drivers can be hit and gets a number between 0.71 and 1.43.

This work is influenced by the above methods. The main difference between this work and its related work is that we focus on service-based systems, not the software architecture in general. In addition, we use our own scenario-based evaluation method, not existing ones such as ALMA or ATAM. However, we should say that our meta-model was influenced by the existing scenario-based evaluation methods. Another difference is that our method is particularly easy to build a system and create scenarios. So, we need minimal manual effort.

4 Analysis and Improvements of the Existing Tool

This section is divided into analysis of the method, the user interface and the code of the existing tool. In each section a brief description of the existing tool in this special part will be given. Then improvements are shown, which can be improved, and then the pros and cons arguments are weighted if the improvement is to be implemented.

4.1 Analysis of the Method

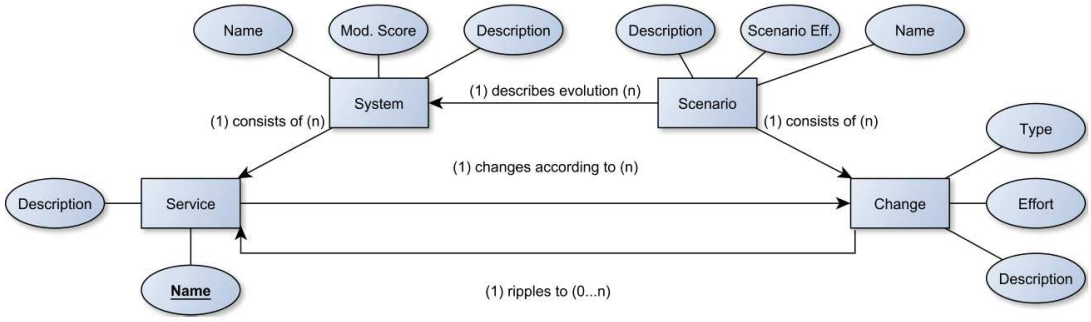


Figure 4 Existing meta model

The existing method can be used to create a system with a name and a description. A system consists of n services, where n> 0. A service also has a name and a description. For a system, scenarios with a name and description can be collected. A scenario consists of m changes with m> 0. Each change has a type, effort, description and an affected service. The effort for a change is measured in hours. The type could be an addition of a new service, a modification of an existing service and a deletion of an existing service. When a service is changed, this change may affect services that the service communicates with. Therefore, the changed services must be able to cause ripple effects. This is represented by the ripples 0 ... k. If the system is evaluated using the determined scenarios, the scenario cost is calculated for each scenario. After that the modifiability score is estimated for the system based on all scenario efforts.

Improvement 1: Add dependencies between the services

To better identifying ripple effects, it could be very helpful to know how services communicate with each other. Services have the property that they communicate through interfaces with each other. It can thus be found out which services are dependent on comparing the interfaces for each service with the interfaces of the other services. However, it is not lightweight to create a service with its interfaces. To make it lightweight, the

dependencies to other services can be established without creating the actual interfaces. In this way, the effort is not too high.

Pros: When the dependencies of a service are known, recommendations can be made about services that are likely to change as a service change. Recommendations can therefore be given to the ripple effects. Unless the user is forced to enforce dependencies, there is no additional overhead to creating a service.

Cons: Not all potential ripple effects may be shown. For this reason, the improvement has a major limitation that only dependent services are displayed as potential ripples. Another argument against the improvement is the fact that not all potential ripples are actually ripples. So, the user can get confused and add a ripple that is not one.

Improvement 2: Add "service that is modified most" as evaluation feature for the system

Bengtsson et al (Bengtsson P. , 2002) use one example of his ALMA method to show how many components are affected by a scenario. The fewer components are affected by a single scenario, the better the architectur. For our method, it would also be helpful to know which service is affected by most changes.

Pros: Vulnerabilities can be detected in the architecture. In the existing system, it is easy to find out what this service is, since all affected services are known and have ripple effects, so that the user does not need any extra effort.

Cons: The meaning is not so clear.

Improvement 3: Add "service that is modified most" as evaluation feature for a scenario

The description and argumentation are the same but in respect to scenarios.

Improvement 4: Categorize each scenario

Influenced by ALMA and SAAM, each of which first elicits scenarios and then classifies them to reduce redundant scenarios, it might be helpful for our method to categorize scenarios. The reason is that the user can be helped to identify the critical services that need to be changed for a particular category of scenarios.

Pros: It can be seen how easy it is to implement a specific category of scenarios. For future predictions, it can be seen how easy or difficult a system can be to change in this category of scenarios.

Cons: As far as we can tell, there is no literature that analyzes the common categories or classes for scenarios. Only *Bengtsson et al.* (Bengtsson P., 2002) names several classes in one of his experiments. However, these are classes that relate to the software architecture he analyzes, and they are not generic classes. How do you know which categories to recommend to the user? The user might not find the appropriate category for his scenario, and the entire context adds no value to the evaluation.

Improvement 5: Add an ordinal scale from 1 to 10 as effort estimation for a change

Koziolek et al. (Koziolek, Domis, Goldschmidt, Vorst & Weiss, 2012) have tried in their lightweight method MORPHOSIS to estimate the financial impact of each scenario using the lines of code of the affected subsystem. However, they did not have enough data to make reasonable estimations. Therefore, they opted for a more abstract cost estimate in the form of an ordinal scale from 1 to 10. For our method, it might also be interesting to use an ordinal scale for how difficult it is to implement a change.

Pros: It is very intuitive to use an ordinal scale to implement a change. Therefore, no additional knowledge is needed to use such a method.

Cons: No obvious negative effects.

Improvement 6: Add the ALMA modifiability score as effort estimation for the total system

Bengtsson et al (Bengtsson P. , 2002) created a maintenance effort equation that generates a result based on modified code, new plug-in code and new written code. The exact equation is described in the related work section. Bengtson created a comparable score with a measurable metric, here lines of code.

Pros: The score is comparable and takes into account the different effort required to change code and write new code. For software developers who know the exact code of the services, it is easy to estimate how many lines of code need to be written or changed.

Cons: The evaluation does not really imply the effort of implementing the scenarios. What does a score of 600 or 1000 say? Are these good or bad numbers? Because services are technology-independent, they can be written in different programming languages, resulting in different lines of code for changing a service. Therefore, one cannot compare the lines of code and the rating well. For people who have not written the service, it may be difficult to estimate the lines of code that need to be changed or rewritten.

Improvement 7: Implement the Constructive Cost Model (COCOMO) 2 for estimating the person months of a change

The COCOMO 2 takes many factors into account when changing the architecture. The exact factors are described in the related work section. We can probably make it easier for our method because filling in 17 factors to make a change is not very quick and easy. The idea is to specify a scale from about 4 to about 12 to represent the sum of the minimum of all factors up to the maximum of all factors. In this way, it is necessary to provide the user with a table with the exact value of a factor, since each cost driver has five different values as far as the fulfillment of the factor is concerned. In addition to the scale representing the 17 factors, a user must also determine the lines of code that need to be changed to implement the change.

Pros: The COCOMO 2 is well researched and takes many factors into account to make accurate estimates, especially if the design of the architecture is already known. With this model, it is possible to specify exactly how many person months a change need.

Cons: This model is very complex. A user needs a lot of time to understand what each factor means. Then the user has to determine the number of a factor, which also costs a lot of time. Overall, the COCOMO 2 is not lightweight.

Improvement 8: Add lines of code as effort estimation for a change

Influenced by ALMA and the COCOMO 2 we think it could be useful to estimate the effort of a change and also the total system in lines of code. It can also be helpful to estimate the lines of code in hours. *Bengtsson et al* (Bengtsson P. , 2002) cites *Maxwell* that about 1.4 lines of code per hour can be written if the existing code wants to be changed. According to this assessment, we can recommend the user the effort in hours when he create a change with the lines of code effort estimation method.

Pros: A person who knows the services well can appreciate the lines of code needed to change a service or write new code. With the hour estimation, we can also specify a more intuitive metric.

Cons: Because services are technology-independent, they can be written in different programming languages, resulting in different lines of code for changing a service. Therefore, the lines of code cannot be compared well. For people who have not written the service, it may be difficult to estimate the lines of code that need to be changed or rewritten.

Improvement 9: Add cosmic function points as effort estimation for a change

The COSMIC Generic Software Model [1] presents itself as second generation in functional size measurement. In order to understand what is different between the cosmic function points and the normal function points, the definition of the function points is helpful. *Matson et al* (Matson, Barrett, & Mellichamp, Software Development Cost Estimation Using Function Points, 1994) define “Function point analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user. The function delivered is unrelated to the language or tools used to develop a software project.”

The difference between cosmic function points and function points is that the generic COSMIC software model is based on basic principles of software engineering. This means that the functional user requirements of a software can be analyzed into unique functional processes consisting of sub-processes. A sub-process can be either a data movement or a data manipulation. A data move shifts a single data group of attributes describing a single "object of interest", the latter being a "thing" of interest to a functional user. The size of a piece of software is then defined as the total number of data movements (Entries, Exits, Reads and Writes) summed over all functional processes of the piece of software. Each data movement is counted as one ‘COSMIC Function Point’. An example to illustrate the method is the function process "Start cooking process". Here are a "start button", a "heater" and a "cooking light" the functional users. The first data movement is to receive the start signal. This movement type is an entry. The second data movement is "sending a power-on command to the heater" as the output data movement type. The last data movement is "sending a power on command to the cooker lamp", which is an exit. For each step, a cosmic function point is required, which yields a total of three cosmic function points.

Pros: Cosmic function points are independent of the language, tools or methods used for the implementation. The problem of the lines of code can thus be solved, that services can be written in different programming languages because they are technology-independent. Cosmic function points are based on the system user's external view of the system. Non-technical users of the software system have a better understanding of how to measure functionality with function points. So, a person who wants to estimate the cost

[1] https://cosmic-sizing.org/cosmic-fsm/

of a change does not need to know how the service works exactly at the coding level. It is enough to know what a service is doing and how it interacts with other services.
Cons: A user must become familiar with the method.

Improvement 10: Add story points as effort estimation for a change

According to *Coelho et al.* (Coelho & Basu, 2012) agile software developing is gaining popularity and replacing traditional methods of developing software. Agile software development can use story points to estimate the effort. Another estimation method in agile software development is, for example, planning poker. A story Point is a unit to measure the size of a user story or a feature. A Story Point is assigned based on the effort, the complexity and the inherent risk involved in developing a feature. (Usman, Mendes, Weidt, & Britto, 2014)
Pros: Since the development of agile software has gained popularity and replaced traditional methods of developing software, perhaps a decent number of software developers have heard story points as an estimation method.
Cons: Estimating the effort required to develop a user story requires the user to have some experience in the estimation, access historical data, and be able to use a trial-based estimation approach (Usman, Mendes, Weidt, & Britto, 2014). Another drawback is that the story point estimate can vary from team to team.

4.2 Analysis of the Usability

The existing tool is a web app written in Vue JS[2]. The developed method is applied in the web app. When a system is created with its services in the tool, the system is created first and then only one service can be added at a time. This leaves open the possibility that a system can have zero services that do not correspond to reality. Since only one service can be created at the same time, additional services must be added via a pop-up in the system overview. In general, the process of editing a system and a scenario will involve multiple pop-ups. When creating scenarios, the same process was used as with the systems. A scenario is created, and then only one change can be added after creating the scenario for the first time. If more changes are to be added, this can be achieved through the scenario overview by clicking on two pop ups. Again, the possibility remains open that a scenario cannot have any changes. In our method, a scenario should have at least

[2] https://vuejs.org/

one change. The navigation through the web app is done via a toolbar at the top of the web app.

Improvement 11: Replace the toolbar with a slide menu

The design of a one-page web app is very popular nowadays. A good example is the video platform YouTube[3]. The idea of this consideration is to replace the toolbar with a slide menu.

Pros: The design is becoming more popular as many large platforms use the design of a slide menu for in-app navigation.

Cons: Common Web sites use toolbar navigation. Therefore, most common web surfers are familiar with toolbar navigation. So, it is unclear if this really leads to more familiarity.

Improvement 12: Add a search bar to the data tables

In the existing tool, there is no way to search for specific elements of a data table. So far, the data of systems and scenarios are managed by tables. It is useful to search for specific elements and attributes, e.g. the description. The addition of a search bar is therefore the logical implication of these assumptions.

Pros: Faster way to search for a specific item in the data table.

Cons: No obvious negative effects.

Improvement 13: Add paging to the data tables

A data table can exceed a large number of items very quickly. As a result, the page must be scrolled to find the right element of the table. It is not very user friendly to have this big table at a glance. Therefore, it makes sense to add paging to the table. That way, all the information on one page of the web app will be displayed without scrolling.

Pros: One side of the app becomes clearer, as the clarity increases. The user is not confused about too much data.

Cons: No obvious negative effects as the user can choose how many elements he wants to see in the table.

[3] https://www.youtube.com/

Improvement 14: Make the process of creating and editing systems/scenarios more consistent

As mentioned in the description of the existing web app, the process of creating and editing a system or scenario is user-unfriendly and does not represent the reality. Since systems and services are coherent, it makes sense to have several steps in the build process, where in the first step information about the system can be collected and in the second step, all services can be added to the system. It is noteworthy that this method does not first create the system in the database and then add the services to the created system. The same stepper can be used for the editing process, but first the system data is entered from the database. The user therefore has the same user interface for editing and adding a system. Another weakness of the existing web app was that the system name is not updateable. It is possible with the stepper method. The same arguments apply to the scenarios and changes.

Pros: The web app will be more consistent. The confusing pop ups will be replaced by a consistent design.

Cons: No obvious negative effects.

Improvement 15: Add an overview page for systems and scenarios

Systems and scenarios contain much information. Displaying all systems and for each system all its services and for each service its dependencies would make the table very large and confusing for the user. Instead, the indication that a system has x services is more useful in the system overview. To continue to display the exact details of a system, a link to a detail view page in the summary table can be added. In the detail view, the above-mentioned information about the system can be displayed clearly. The same arguments apply to scenarios and changes.

Pros: A clear overview in the system overview and in the detail view is obtained. It is also a really consistent design because it is applicable to both scenarios and systems.

Cons: No obvious negative effects.

Improvement 16: Add an option to compare two system in respect to the evaluation

Bengtsson et al. (Bengtsson P. , 2002) compares two architectural designs based on how many components are changed when a scenario is implemented. In our tool, it might be interesting to compare two systems in relation to the same scenarios. Therefore, there should be an option on the evaluation page to place two systems on the site and compare

them manually. It is also helpful to clone systems and their scenarios to change the cloned system and compare the modified system to the original system based on the same scenarios.

Pros: Alternative architectures of a system can be compared in the same scenarios as the original systems. It may be considered to change the architecture of the original system to improve the general modifiability of the system in the identified scenarios.

Cons: It is questionable how valuable the overall information is. The calculated effort for the modified system is the same if the effort estimate for a change is not adjusted. Maybe the modification in the alternative design is easier, which is not so clear. The main difference is how many services are changed in relative proportion to all services. In some scenarios, there may be a need to change a lower percentage of services because the services may be split in the alternative design. The smaller the proportion of all changed services, the better the architecture. So, the point is that maybe fewer services need to be changed in relation to all services.

Improvement 17: Add a method to upload a system in our tool

For example, creating a system with 30 services and also creating dependencies between the services can be very time consuming. In our tool, we want a simple way with little effort. Therefore, it may be useful to automate the creation of a system by creating an interface that allows the user to upload their system. Then the system should be parsed into our format. Romano et al. (Romano & Pinzger, 2012) tried to do something similar. They use the Web Services Description Language (WSDL) and the Eclipse Modeling Framework (EMF). The method is particularly applicable to web services, so further investigation is needed to find something similar for service-based systems in general. Once the system is described in WSDL, it can be parsed into an EMF model using the org.eclipse.wst.wsdl API. Next, the EMF can be converted to XSD with XSD Transformer. Through an XSD element, the attributes can be easily read and inserted into our system.

Pros: A lot of manual work is saved because the system is not created manually. Another advantage is that the XSD model can be cloned and the user can change the cloned scheme. Thus, as with KAMP, a task list can be created and grouped into scenarios. This improvement would not only automate the process of creating the system, but also the process of creating scenarios.

Cons: No obvious negative effects as the user can freely use this method.

4.3 Analysis of the Code

Vue.js is a progressive framework for building user interfaces. It uses a single file components approach. This means that the HTML part, the JavaScript part, and the CSS part can be written to a single file component. It is tempting for every page in the web app to be written in a very large component. The problem is, if a component gets too big, say 400 lines of code, it is hard to understand and therefore hard to change. This section lists the improvements to the code being analyzed.

Improvement 18: Abstract the components

In the existing web app, each page is a component. A component is therefore very large and has several functions. For example, in the System Overview component, there is a table that displays all systems and in the same component are modals that can be used to manipulate a system. This example component can be improved by making the system overview a wrapper component, making the table a custom component, and making the modal a custom component. In the wrapper component, the table component and the modal component can be registered. Each graphical and functional element has its own component. This makes it easier to modify the components compared to the previous one big component. This principle of component abstraction can be applied to all existing components. Future modifications save a lot of working time due to the strict separation of elements and functions.

Pros: Modification will be easier and faster in the future. Also, the components are more structured.

Cons: The restructuring and rewriting of components initially requires a lot of effort.

Improvement 19: Vuetify as style library

For Vue a special style library has been developed. It is called Vuetifiy[4]. Vuetify uses a material io approach. This design is widely used in Google Apps, which may increase the app's feel.

Pros: Vuetify is well documented and makes it easy to use. Updates are constantly being provided that provide the framework with more and more UI components. It is well integrated with Vue.

Cons: The HTML part in every component needs to be replaced.

[4] https://vuetifyjs.com/en/

Improvement 20: Add the Vuex store

"Vuex[5] is a state management pattern + library for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion. It is a self-contained app with the state, the source of truth that drives the app, the view, a declarative mapping of the state and the actions, the possible ways the state could change in reaction to user inputs from the view."[5] A "store" is basically a container that holds the application state. States accesses the variables that are in the store in each component of the app. It can help for in app communication and some variables needs to be global.

Pros: It enables a variety of functions that are otherwise difficult to implement, e.g. dark and light themes and multi-language support. It improves in-app communication between components.

Cons: It is very complex, so the complexity of the code becomes more difficult to change.

4.4 Selecting the Improvements

From the method improvements, we believe that most methods of estimating the effort are worth adding and evaluating. Although we believe that the improvement six is not meaningful enough by adding the ALMA modifier modification estimate to the user, we will not add it to our tool and therefore do not rate it. We think that COCOMO 2 is too complex for our lightweight construction, so it will not be added. Improvements one to four, which are related to the improvement of the evaluation, will be included in our tool as there is a chance that the rating of a system based on the identified scenarios will be improved.

The improvements in usability are all done, except for improvement 17, because this improvement is too complex to implement. Therefore, improvements 11 through 16 are implemented in the web app, as we expect it to greatly enhance usability. The code enhancements are all made because they support usability improvements.

The new meta-model looks like this:

[5] https://vuex.vuejs.org/

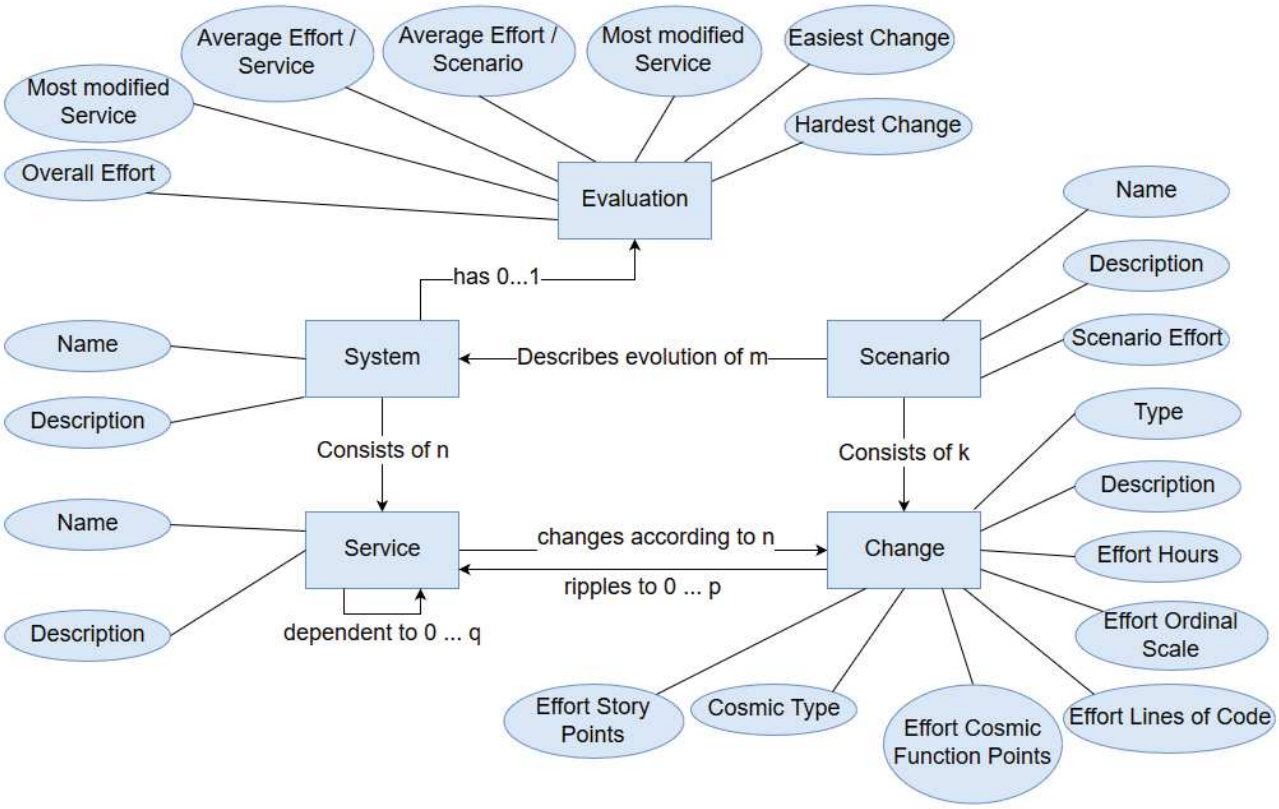


Figure 5 Improved meta model

5 Example Case Study of the improved Method

To illustrate how our improved method of evaluating service-oriented systems works with a scenario-based evaluation method, we apply our method to an example system. We have chosen a relative current example: Uber. Uber has recently redesigned its architecture, transforming it from a monolithic to a microservice architecture.

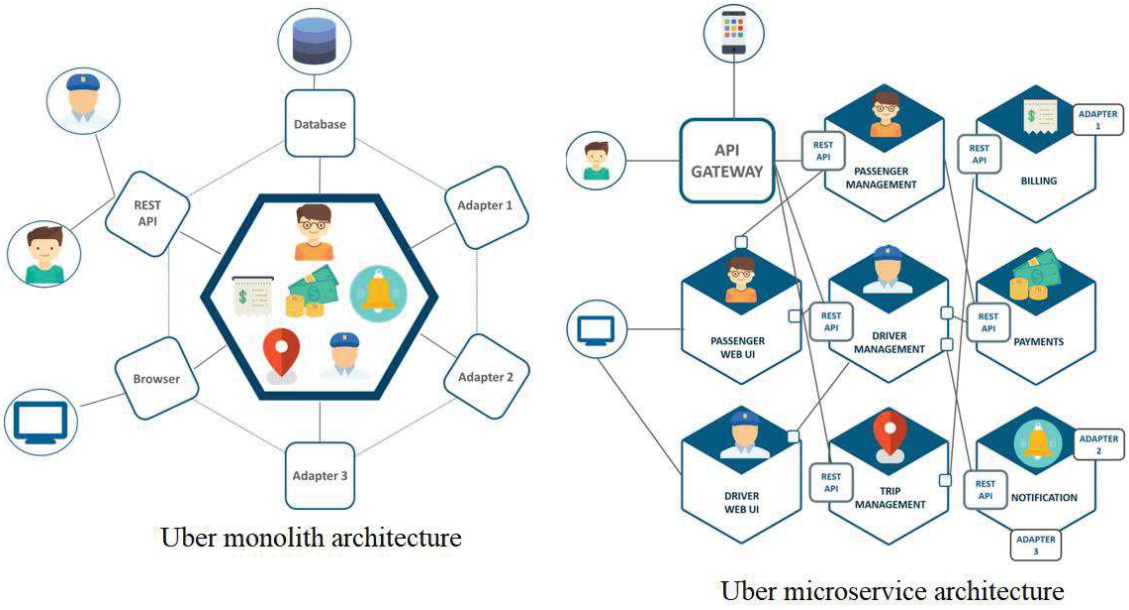


Figure 6 Uber comparison by Sahiti Kappagantula (https://dzone.com/articles/micro-service-architecture-learn-build-and-deploy-a)

As visible in figure 6, Uber has now 8 services:

Service name	Description	Dependency
Billing	It manages the billings. Every Trip has a billing.	-
Notification	It sends notification to drivers and passengers.	-
Payments	All payments are made through this service.	-
Driver management	It manages the drivers.	Payments, Notification

Driver Web UI	A driver can accept the journey of a passenger here.	Driver management.
Passenger management	It manages the passengers.	Payments
Passenger Web UI	A passenger can book his trip and receive notifications here.	Passenger management, Driver management
Trip management	It manages the route from the location of a passenger to his destination.	Billing

Table 1 Services of Uber

Since Uber is a large company and whose software architects have thought well about the architecture, it is difficult to create change scenarios. We tried it anyway:

Scenario name	Description	Category	Number of changes
Add a validation for user input	To verify user input, a new validation service is created that is protected against common attacks such as cross site scripting.	Security	1
Add new payment methods	Adds new payment methods to the payment service.	Market driven	3
Better trip calculation	Modify the algorithm and some usage APIs for faster trip calculation.	Algorithm change	2
Change database for persistent data	Because another database model	New database management system.	2

	improves performance, the database is changed.		
Design and implement more user-friendly UI	Significant redesign and implementation of user interfaces for driver and passenger.	User interface overhaul	3
More notifications	For more feedback, more notifications should be created.	Notification improvement	2

Table 2 Potential change scenarios

For these scenarios the following changes will be created:

Change description	Relates to scenario / scenario description	Type	Effort	Affected service	Ripples to services
It validates the user input.	Add a validation for user input / To verify user input, a new validation service is created that is protected against common attacks such as cross site scripting.	Addition	200 hours 8 280 LOC 40 CFP 40 SP	[new addition] Validation	Passenger Web UI, Driver Web UI

Add Bitcoin as payment method	Add new payment methods / Adds new payment methods to the payment service.	Modification	70 hours 4 98 LOC 20 CFP 20 SP	Payments	Passenger management, Driver management, Notification, Passenger Web UI, Driver Web UI
Add GiroPay as payment method	Add new payment methods / Adds new payment methods to the payment service.	Modification	50 hours 3 70 LOC 10 CFP 8 SP	Payments	Passenger management, Driver management, Notification, Passenger Web UI, Driver Web UI
Add Ethereum as payment method	Add new payment methods / Adds new payment methods to the payment service.	Modification	70 hours 4 98 LOC 20 CFP 10 SP	Payments	Passenger management, Driver management, Notification, Passenger Web UI, Driver Web UI

Use other APIs	Better trip calculation / Modify the algorithm and some usage APIs for faster trip calculation.	Modification	80 hours 4 112 LOC 18 CFP 13 SP	Trip management	
Change algorithm for improving performance	Better trip calculation / Modify the algorithm and some usage APIs for faster trip calculation.	Modification	200 hours 7 280 LOC 55 CFP 40 SP	Trip management	
Change the database in the passenger management	Change database for persistent data / Because another database model improves performance, the database is changed.	Modification	160 hours 5 224 LOC 23 CFP 20 SP	Passenger management	Passenger Web UI

Change the database in the driver management	Change database for persistent data / Because another database model improves performance, the database is changed.	Modification	160 hours 1 224 LOC 25 CFP 20 SP	Driver management	Driver Web UI
Reimplement the Web UI for passengers	Design and implement more user-friendly UI / Significant redesign and implementation of user interfaces for driver and passenger.	Modification	800 hours 6 1120 LOC 65 CFP 100 SP	Passenger Web UI	
Change Web framework	Design and implement more user-friendly UI / Significant redesign and implementation	Modification	2000 hours 9 2800 LOC 80 CFP 100 SP	Passenger Web UI	

	of user interfaces for driver and passenger.				
Reimplement the web UI for drivers	Design and implement more user-friendly UI / Significant redesign and implementation of user interfaces for driver and passenger.	Modification	800 hours 6 1120 LOC 65 CFP 100 SP	Driver Web UI	
Notification when driver has arrived	More notification / For more feedback, more notifications should be created.	Modification	160 hours 1 224 LOC 25 CFP 20 SP	Notification	Driver management, Driver Web UI, Passenger management, Passenger Web UI
Notification when driver is rated	More notification / For more feedback, more notifications should be created.	Modification	160 hours 1 224 LOC 25 CFP 20 SP	Notification	Driver management, Driver Web UI, Passenger management,

					Passenger Web UI

Table 3 Changes for the scenarios. Effort is estimated in hours, on an ordinal scale from 1 to 10, in lines of code (LOC), in cosmic function points (CFP) and story points (SP)

With this scenario, we can evaluate the system in terms of total cost and performance of the system in individual scenarios. We only consider the hourly method for the effort estimate. When evaluating with other cost estimation methods, the results differ slightly.

First, the "critical service that is most modified" is calculated. To do this, we put all affected services through a change and the ripples from a change in an array and count the element that is most in the array.

- ⇨ [PWU, DWU, PAY, DM, PM, PWU, DWU, PAY, DM, PM, PWU, DWU, PAY, DM, PM, PWU, DWU, TM, TM, PM, PWU, DM, DWU, PWU, PWU, DWU, N, DM, DWU, PM, PWU, N, DM, DWU, PM, PWU]
- ⇨ First, we set the temporary item to Passenger Web UI and count it nine times. Then set the maximum counter to nine.
- ⇨ We count nine times for the next item, so do not do anything.
- ⇨ If a counter exceeds 9 times, place the maximum counter on the counter of the item and the temporary item on the counted item. If the counter is just the maximum counter, do nothing.
- ⇨ Return the temporary item. Here it is Passenger Web UI.

Second, the "critical service with the highest total cost" is calculated. It works by putting all the services involved and their effort into an array. The next step is to go through an array of affected services and the previously created array. If a service matches in both arrays, the overhead is summed and stored in a result array after both array iterations, along with the service.

- ⇨ First array: [{Payments, 70}, {Payments, 50}, {Payments, 70}, {Trip Management, 80}, {Trip Management, 200}, {Passenger Management, 160}, {Driver Management, 160}, {Passenger Web UI, 800}, {Passenger Web UI, 2000}, {Driver Web UI, 800}, {Notification, 160}, {Notification, 160}]
- ⇨ The affected services array: [Payments, Trip Management, Passenger Management, Driver Management, Passenger Web UI, Driver Web UI, Notification]

- ⇨ Result array: [{Payments, 190}, {Trip Management, 280}, {Passenger Management, 160}, {Passenger Web UI, 2800}, {Driver Web UI, 800}, {Notification, 320}]
- ⇨ In the final step, the array is sorted in terms of effort and its 0-element returned, which is then the element with the highest overhead. Here it is the Passenger Web UI.

Third, we also charge the "service with the lowest overall effort". This is luckily the last element of the array in the previous calculation. So here it is Payments.

Fourth, the "combined effort for the overall system" is calculated. It is the sum of all scenario efforts that are calculated from the sum of the efforts of the associated changes. Here it is 4910 hours.

With the total effort, we can calculate the average cost per service of 613.75 hours and the average expense per scenario of 818.33 hours.

Note that hypothetically added services and hypothetically deleted services are not included in this assessment.

For each scenario, the hardest change, the simplest change, the most effective change, the number of services involved, and their overhead are calculated in much the same way as the system's formulas. That's why we only present the result.

Scenario	Hardest Change	Easiest change	Most impactful change	Scenario effort	# of affected services	Category
Add a validation for user input	Addition:It validates user input / 200 hours	Addition:It validates user input / 200 hours	Addition:It validates user input / affects 3 services	200 hours	3 / 9	Security
Add new payment methods	Modification:Add Ethereum / 70 hours	Modification:Add GiroPay / 50 hours	Modification:Add Ethereum /	190 hours	7 / 9	Market driven

			affects 6 services			
Better trip calculation	Modification:Change algorithm / 200 hours	Modification:Use other APIs / 80 hours	Modification:Change algorithm / affects 1 service	280 hours	1 / 9	Algorithm change
Change database for persistent data	Modification:Change the database in the driver management / 160 hours	Modification:Change the database in the passenger management / 160 hours	Modification:Change the database in the driver management / affects 2 services	320 hours	4 / 9	New database management system
Design and implement more user-friendly UI	Modification:Change Web Framework / 2000 hours	Modification:Reimplement the Web UI for drivers / 800 hours	Modification:Reimplement the Web UI for passengers / affects 1 service	3600 hours	2/ 9	User interface overhaul
More notifications	Modification:Notification when driver is rated / 160 hours	Modification:Notification when driver has arrived / 160 hours	Modification:Notification when driver has arrived / affects 5 services	320 hours	5 / 9	Notification improvement

Table 4 Evaluation table for the scenarios

6 Evaluation

In this section, we present the evaluation of our improved, tool-based method. We conducted a qualitative evaluation through hands-on-interviews. With these interviews, we attach particular importance to how intuitive the method is and what can be improved. For a qualitative assessment, we conducted a survey that paid particular attention to the evaluation of effort estimation methods. In both forms of evaluation, we examined how useful the evaluation of a system is based on the scenarios created.

After showing our results of the evaluation, we will draw conclusions from this in the last part of this section.

6.1 Survey

This section is divided into the design of the survey and the results we found in the survey.

6.1.1 Survey Design

In this section, we describe how we conducted the survey. The survey consists of five parts. In the first part, we described the survey and its purpose. In the second part, we gather information on the effort estimation methods, which were hours, an ordinal scale, lines of code, cosmic function points and story points, in terms of familiarity, precision and applicability. In the third and fourth part, we collect information about the evaluation functions in our tool. In the last part, we asked the participants to share their professional background. The **Survey: Scenario-based Modifiability Analysis of Service-based Systems** (which can be found in the appendix) takes approximately five to seven minutes and has a total of 17 questions. Six questions about the estimation of the scenario effort, six questions about the evaluation features and five questions about the personal experience.

We shared the survey on platforms like Thesius[6], Surveycircle[7], Xing[8] and Researchgate[9], on several email lists and with personal contacts.

[6] https://www.thesius.de/

[7] https://www.surveycircle.com/de/

[8] https://www.xing.com/

[9] https://www.researchgate.net/

6.1.2 Results

All in all, we received 41 replies. We used only 40 answers because one participant wrote "test" in each optional field and answered each question with "five," which required an opinion on a scale of one to five. That is why we excluded him.

First, we describe the general statistics of work experience:

- 32.5% of all participants work in academia active, 35% in industry and 32.5% in both academia and industry.
- The average work experience is 11 years.
- 50% of all participants previously used a scenario-based method in real projects.
- The average familiarity with scenario-based methods is 2.83 on a scale of 1 to 5, where 1 is unfamiliar and 5 is familiar.
- The average familiarity with service- or microservice-based systems in general is 4.1 on a scale of 1 to 5, where 1 is unfamiliar and 5 is familiar.

The experience of our participants is quite good for the evaluation of our effort estimation methods and assessment features. We focus on service-based systems, so an average familiarity with service-based systems of 4.1 is pretty good. Familiarity with scenario-based methods might be better, but it is still okay, since it is average, which means about three on a scale of one to five. We can separate our sample size into people with science and industry as an active field of work, people who use scenario-based method in real project and which do not. This way, we can tell if there are differences.

This section is divided into two parts. One describes our results on the estimation methods and another describes our results on the evaluation features.

6.1.2.1 Estimation Methods

In the first part, we describe how the participants evaluated our predefined effort estimation methods, then describe what other effort estimation methods the participants recommended, and in the last part, we looked for correlations in the responses given by the participants.

Figure 7 shows the mean of the answers to our effort evaluation methods. The story points were rated best by the participants in terms of both precision and usability at 3.26 and 3.46. The precision and applicability of the hours method was rated second best by the participants in terms of precision and applicability, but still mediocre at 2.98 and 3.28, which means about 3 on a scale of 1 to 5. The accuracy of the ordinal scale, the cosmic

function points and the code lines were rated below average, which means less than three. The ordinal scale has been ranked in the applicability as mediocre, which means about three. The cosmic function points and lines of code were rated below average in applicability, i.e. under three.

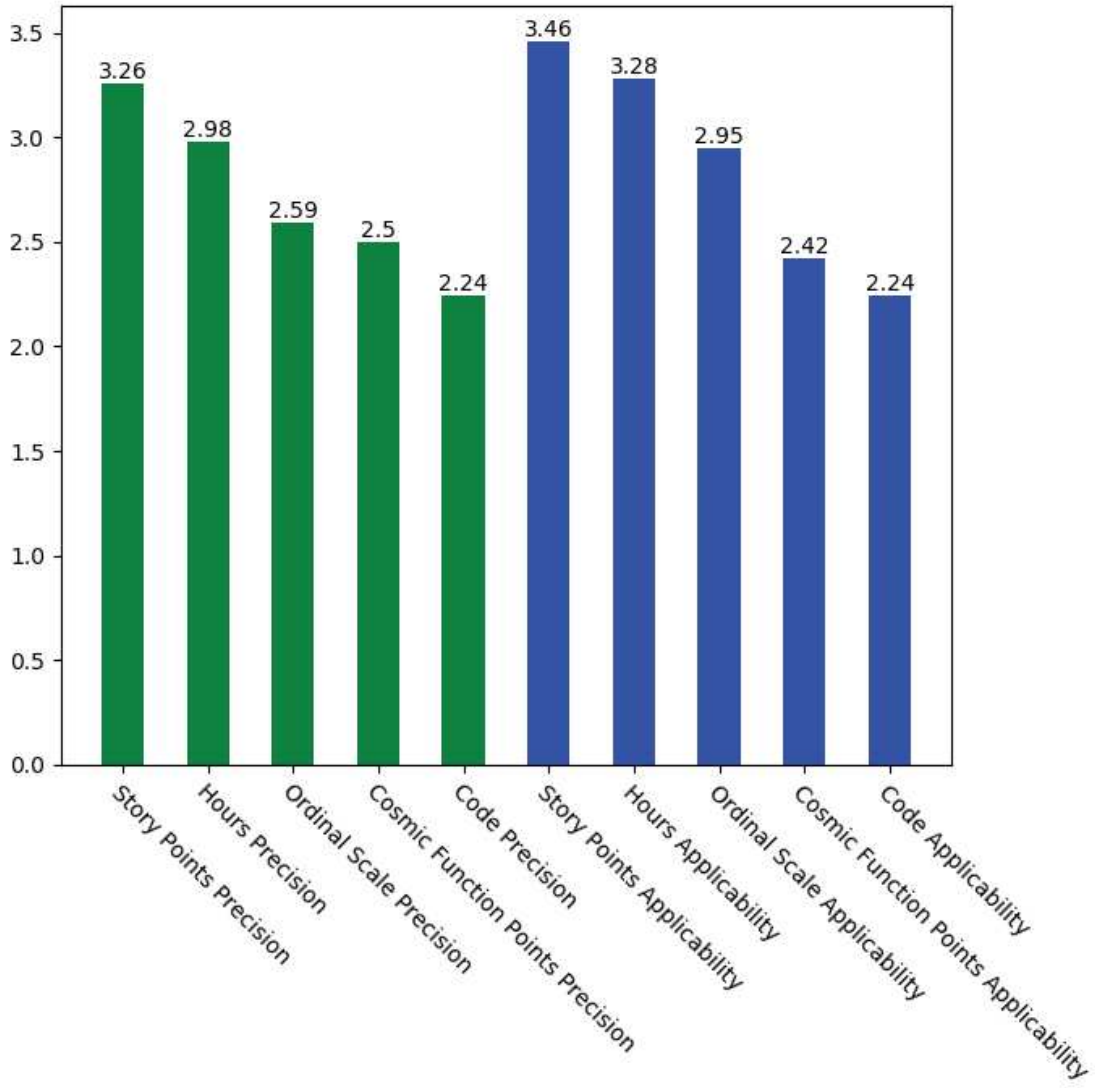


Figure 7 Mean of accuracy and applicability of the effort estimation methods unfiltered. Where green is the precision and blue the applicability.

Figure 8 shows the mean of the responses to our effort evaluation methods, filtered by familiarity of the method >= 3, on a scale of 1 to 5, to obtain meaningful results for the precision and applicability. The figure indicates that the accuracy of the story points was rated best at 3.55. The accuracy of the cosmic function points, the ordinal scale and the hours was evaluated with about three equal accuracies each. The accuracy of the lines of

code was rated worst at 2.56. The applicability of the ordinal scale and the story points is best, according to the participants, as the average rating of applicability is best at 3.83 and 3.81. The applicability of the hourly method was also rated as quite good at 3.47. The applicability of cosmic function points was rated as mediocre at 3.14 and the applicability of the lines of code as the worst of the five effort estimation methods at 2.75.

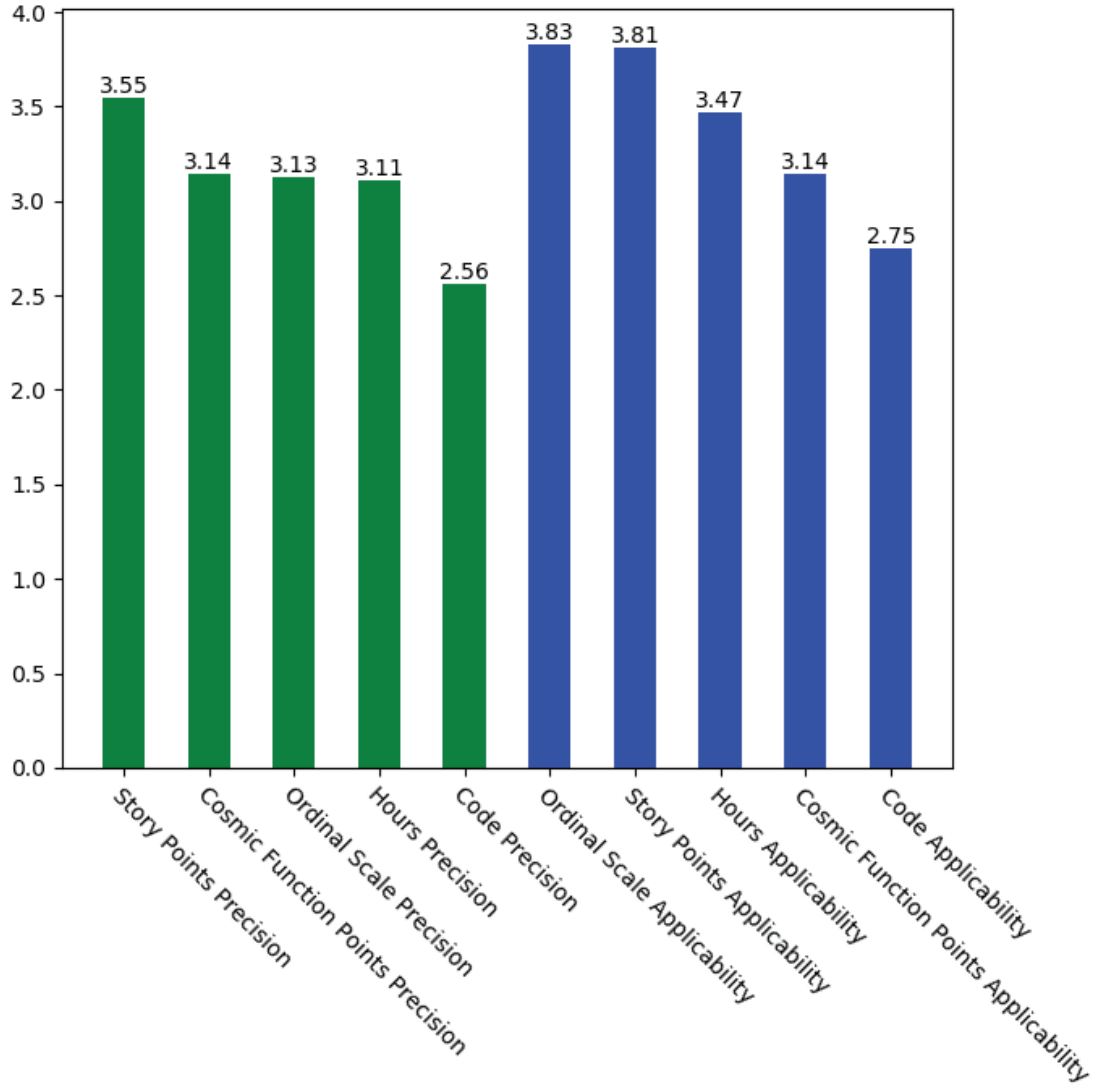


Figure 8 Means of precision and applicability of the effort estimation methods, filtered by familiarity of the method >= 3. Where green is the precision and blue the applicability.

This means that the story points are best suited according to the participants to estimate the cost of changing service-based systems as this method has been evaluated by the participants with the highest precision and second-highest applicability. The value of the

applicability is close to four, which means good. The ordinal scale was also judged by the participants to be good at assessing the cost of a change in service-based systems, as its applicability is best valued and close to four, which is good and its accuracy mediocre, which means three. The cost estimation in lines of code was rated worst because their applicability and accuracy was rated below average by all participants, i.e. under three.

Next, we divided the participants into nine groups. In group one are participants who used scenario-based methods in real-world projects, in group two are participants who have not used scenario-based methods in real-world projects, in group three are participants with above-average professional experience considering all participants, i.e. over 11 years of work experience, in group four are participants with below-average professional experience considering all participants, i.e. under 11 years of work experience, in group five are participants with an academic background, in group six are participants with an industrial background. Participants with an academic and industrial background are in both groups, five and six. In group seven are participants with above-average familiarity with service-based systems considering all participants, i.e. over 4.1 familiarity of service-based systems on a scale of 1 to 5, in group eight are participants with below average familiarity with service-based systems considering all participants, i.e. under 4.1 familiarity of service-based systems on a scale of 1 to 5. Group nine is an expert group, i.e. participants with above-average professional experience, above-average familiarity with service-based systems and above-average familiarity with scenario-based methods. In order to obtain meaningful results, we have considered answers to a method only if the familiarity was greater than or equal to three. We did this in every group.

Then we compared groups one and two, groups three and four, groups five and six, and group seven and eight, looking for differences.

Comparing groups three and four and comparing groups five and six, we found no significant differences in accuracy and applicability. Our interpretation is that it does not matter to a participant where his background lies and how many years he has been working. This probably means that a participant does not need much work experience or background to assess an effort estimation method according to its applicability or precision.

When comparing groups one and two, there are big differences in how the groups assessed the precision and applicability of each method. In the first group, participants rated the precision and applicability of each method half a point higher than the participants in the

second group. Except for the precision of the hours methods. There participants of the first group rated the method only with 0,34 points higher. Another big difference is the evaluation of the cosmic function points. Participants of the first group rated the cosmic function points in the applicability with 3.5 and in the precision with 3.5 as the second highest. In comparison, participants in the second group rated the cosmic function points in the applicability with 2.67 and in the precision with 2.67. Our interpretation is that participants who have used a scenario-based method in real projects can estimate the effort of a change with each effort estimate more accurately than participants who did not use a scenario-based method in real projects. The participants of the first group also believe that the effort estimation methods for service-based systems are more applicable than those of the second group.

When comparing groups seven and eight, there are also significant differences in how the groups assessed the precision and applicability of each method. Participants of group seven evaluated the applicability and precision of each effort estimation method in a range of 0.13 to 0.56 with an average of 0.33 points higher. Participants of group seven rated the cosmic function points 3.5 as the second highest method in the field of precision and the applicability also 3.5 as the third highest method in the field of applicability. Participants of group eight rated the cosmic function points to be the fourth highest accuracy with 3.0 and also the fourth highest applicability. Participants in group seven rated the applicability of the hours at 3.27 as the fourth highest in comparison to participants in group eight, which rated the applicability of the hours as the third highest with 3.62. Our interpretation is that the better a participant is familiar with service-based systems, the better he evaluates the precision and applicability of an effort estimation method.

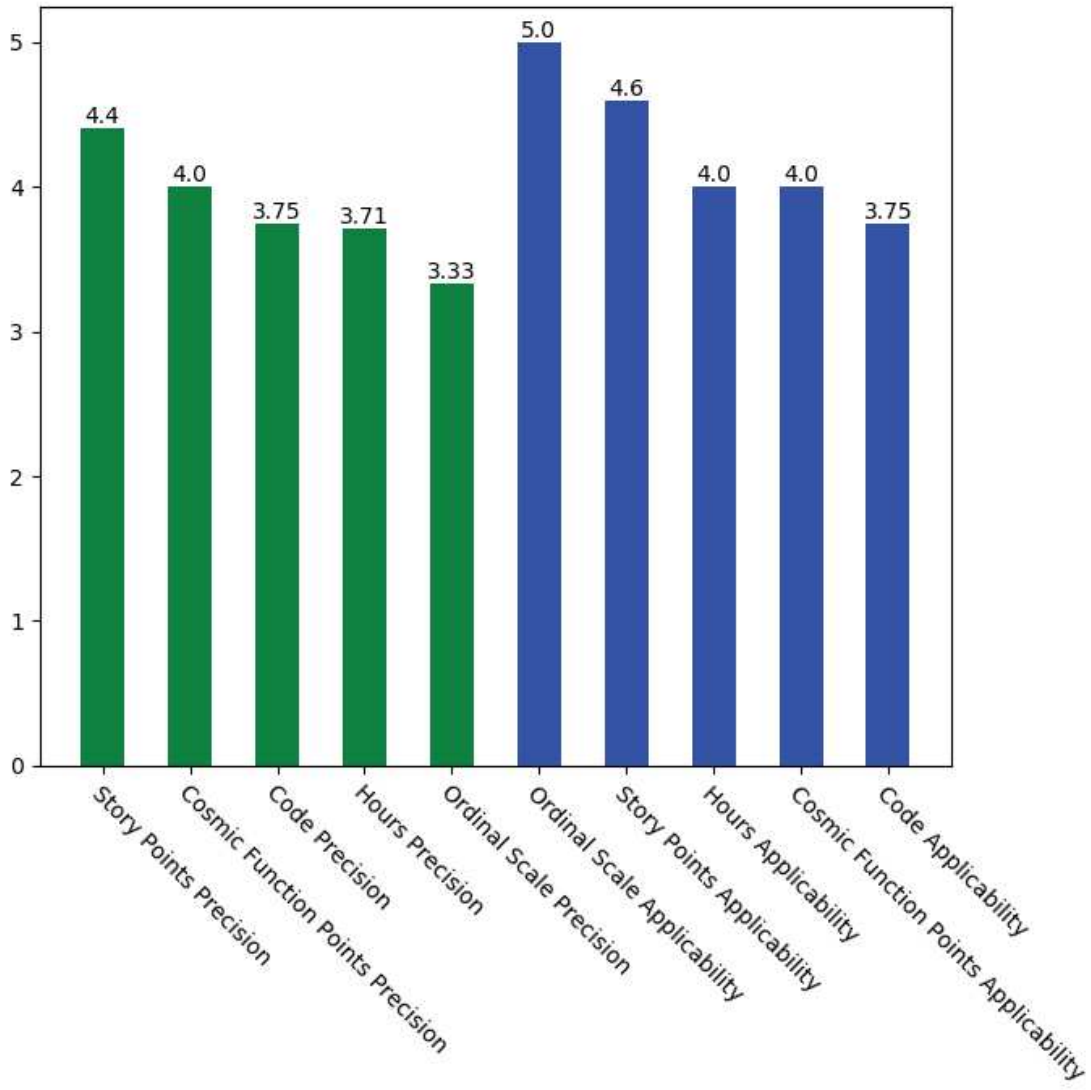


Figure 9 Means of precision and applicability of effort estimation methods from an expert group of participants (participants with above average professional experience, above average familiarity of service-based systems and above average familiarity of scenario-based methods, based on the mean of the value from all participants). Where green is the precision and blue the applicability.

Figure 9 shows the results of the ninth group, the expert group of seven participants. This group stands out from the other groups and the means of all participants. First, they rated the accuracy of the ordinal scale at only 3.33, but their applicability with 5. Second, they rated the precision and applicability of the lines of code above 3.2, which no other group did. Third, they rated the accuracy and applicability of each method above 3.7, which is quite good if we consider 4 good. Except for the accuracy of the ordinal scale.

Our interpretation is that the more familiar a participant is with service-based systems or scenario-based methods, the better abstract estimation methods such as the story points, cosmic function points, and an ordinal scale are judged in terms of precision and applicability compared to concrete effort estimation methods such as hours and lines of code.

We also asked participants what other cost estimation techniques they would find useful for a low-weight method based on scenarios. We found that they recommend abstract methods such as test complexity, affected libraries, affected modules, and the overhead associated with the service-interface complexity metrics.

Lastly, we searched for correlations between the methods and the background of the participants. To calculate the correlation, we used the Spearman correlation. As a threshold for significance, we use a p-value of <0.05. Is there a relationship between a particular estimation method and familiarity with scenario-based methods, personal experience, the active field, or when a person has experience with scenario-based methods in real-world projects? We have not found any such correlations.

We found a weak correlation between professional experience and familiarity with scenario-based methods with a correlation of 0.35. The more years of professional experience a person has, the more familiar a person is in general with scenario-based methods.

6.1.2.2 Evaluation Features

Again, in the first part, we describe how the participants rated our evaluation features and then describe which other evaluation features the participants recommended. In the last part we looked for correlations in the answers given by the participants.

Figure 10 shows the mean of all our evaluation features. The usefulness of the critical service that has modified the most was ranked the best by all participants of our five given evaluating features at 3.9. The usefulness of the critical service with the highest effort, the average cost per scenario, and the change that impact most services were rated average and good at just under 3.5 each. The usefulness of the number of services affected was rated as the worst of the five given features with 2.65.

Our interpretation is that most of the participants find four of the five evaluation features for rating service-based systems useful, as the averages show.

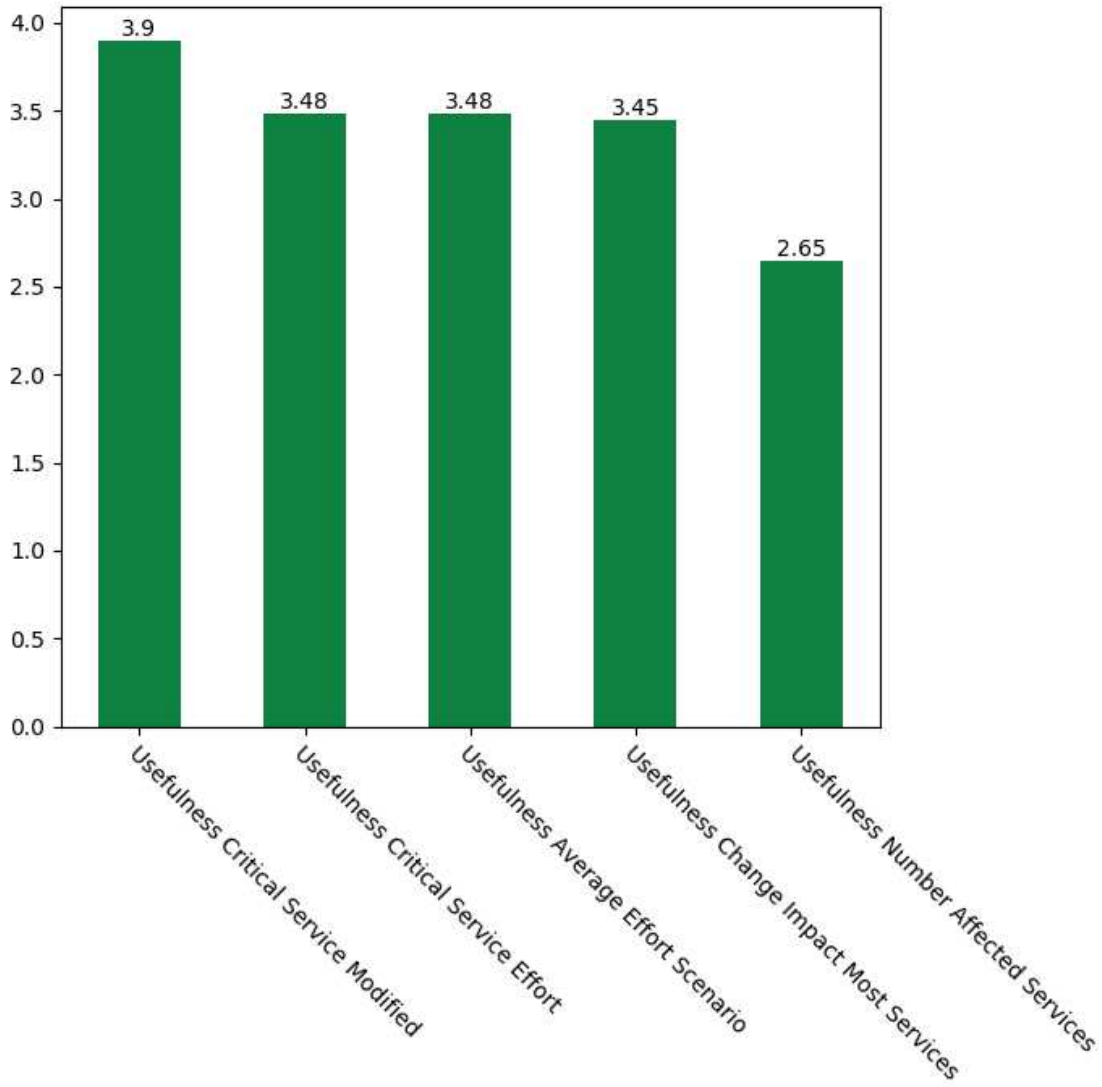


Figure 10 Average of respondents' responses to each evaluation feature

Again, we divided the participants into nine groups, which are the same as in 6.2.2.1, and compared the groups in the same way.

There were no significant differences between groups three and four, groups five and six, and seven and eight. This means that in comparison of the groups no rating feature stood out.

In comparison of the first and second group we found differences. First group participants rated the usefulness of the most critical service, which changed the most, to 3.6, compared to 4.2 in the second group. Another difference is the evaluation of the usefulness of the

number of affected services. Participants in the first group rated it at 2.8, while participants in the second group rated it at 2.5.

In group nine, the expert group, we found that they rated the value of the critical service with the highest effort and the number of services affected 0.21 and 0.23 points higher than the average of all participants.

Our interpretation of the search for differences in the groups is that the more familiar a participant is with the scenario-based method, the better he evaluates the usefulness of the number of affected services. This feature is still rated below average, which means three. The other evaluation features are rated pretty much the same.

In an optional question, we asked what other rating attributes participants would find useful in a lightweight scenario-based method. We found that participants wanted features that can be visualized well, such as the distribution of changes, e.g. 10% UI, 20% DB and 70% route calculation or test coverage of the existing functionality. They also wanted abstract functions such as algorithmic complexity or how many tests are available for a scenario.

Next, we searched for correlations between the personal background and the evaluation features. Again, we found no such correlations.

6.2 Hands-On-Interviews

This section is divided into the design of the hands-on-interviews and the results we found in them.

6.2.1 Hands-On-Interview Design

In this section, we describe how we conducted the practical interviews. We divided the interviews into four parts. Before starting the interview, we had the participant read the task list attached (**Hands-On-Interview: Evolution of a Video Platform**), while preparing a laptop where the participant uses the tool to perform the task. These were estimated five minutes. In the first part of the interview, we had the participant model a system in the tool based on a provided description, a YouTube-inspired video platform. As the participant created the system, we took notes of what we were observing, like what the

participant is doing and what struggles he was having. After the participant created the system, we asked him questions about the task. We did the next two parts of the interview in the same way. In the second part of the interview, we had the participant create scenarios describing changes in the system. Again, we watched the participant while the task was done and took notes. After the scenarios were created, questions were asked about how to create the scenarios. In the third part, we let the participant evaluate the system based on the scenarios created in our tool. As the participant did so, we took notes about what he was doing. After the participant finished the evaluation part, we asked him questions about it. In the last part of the interview we asked questions about the professional background of the participant.

6.2.2 Results

In total, we conducted seven hands-on-interviews. In this section, we describe our observations and the participants' answers to the questions we have asked. The questions can be found in the appendix under **Hands-On-Interview: Evaluation**. All participants are internal or external PhD students of the University of Stuttgart. In Table 5 and 6 we show their information.

Participant	Work experience	Familiarity scenario-based methods	Familiarity of service-based systems
1	19 years	3	1.5
2	17 years	4	3
3	6 years	2	3
4	8 years	4	2
5	5.5 years	2	2
6	6 years	1	3
7	3 years	4	2
Mean	9.3 years	2.86	2.43

Table 5 Professional background of participants in the hands-on-interviews. Familiarity scenario-based methods and familiarity of service-based systems is on a scale of one to five, with one is unfamiliar and five is very familiar.

Because we want to test how easy our tool and method is, they are good participants because they are less well-versed with scenario-based methods and service-based systems. The problem, on the other hand, is that we want to evaluate how useful our evaluation view is in our tool, and that it would be ideal for above-average, that is, over three, familiarity with service-based systems.

Next, we show participants' characteristics and results on our questions, which required a rating of one to five.

Participant	Duration for creating a system	Simplicity of creating a system	Duration of creating scenarios	Usefulness of creating scenarios	Helpfulness of the recommended ripples	Duration for the evaluation part	Helpfulness of the evaluation view
1	5 min	2	18 min	3	5	4 min	3.5
2	7 min	2	21 min	3	5	4 min	4
3	4 min	1	15 min	3	4	3 min	4
4	6 min	1	15 min	4	2	4 min	5
5	6 min	1	16 min	4	2	6 min	4
6	3 min	1	13 min	3	5	2 min	4
7	3 min	1	15 min	3	5	2 min	4
Mean	4.86 min	1.29	16.14 min	3.29	4	3.57 min	4.41

Table 6 Characteristics and results of the participants in the hands-on-interviews. The simplicity is on a scale of one to five, with one easy and five difficult. The usefulness of the recommended ripples is on a scale of one to five, one being unhelpful and five being very helpful. The same applies to the helpfulness of the evaluation view.

As already mentioned, we divided the interview into four parts. Therefore, this section is also divided into three parts because we are not dealing with the professional background in this section. For each part, we describe our observation while the participant has completed the task. Then we describe the answers to the questions without paying attention to the quantitative answers.

6.2.2.1 Creating System

We started each interview on the overview page in our tool, where the process of evaluating a system with the created scenarios is described and what you need to do to reach the evaluation goal. We observed that each participant reads through this introduction before beginning the actual task of modeling the system, a video platform. Although we abandoned the order in which a participant should create the system and its services, three out of the seven participants created the services in a random order and added the dependencies for the services later when editing the services. The edit button was always found very quickly. The other four participants completed the task in the given order and created the services with their dependencies at once.

Next, we describe what answers we got to the question, "What was easy or difficult for a scenario?" Participant two, three, four and seven responded that it is easy to create a system in our tool because the user interface is very intuitive, and the task is well structured. One participant found that the editing process was confusing at first glance but intuitive at the second, recognizing that it was the same process as creating the system. Another participant also criticized the editing. For him, it would be better to work on the system and the individual services individually and not together. He also criticized that the extension panel of a service is confusing. The expanded panel for a service displays its information and actions, which is to edit and delete the service.

"What do you think about defining dependencies in this way?" Five participants thought it easy to define dependencies in this way. Two out of the seven participants noted that a graphical representation of the services and their dependencies would be useful for reviewing a system in the system detail view or for the third step in the process of creating a system. Participant one and four thought the drop-down menu was too big, and it's annoying to click outside the menu to close it.

6.2.2.2 Creating Scenarios

Our most important observations in this part of the interviews were: Categories are selected through a drop-down menu in our tool. To display all categories, the user must scroll down. The user can also enter his own categories. Three of the seven participants did not recognize scrolling through the category drop-down menu. Only one participant realized that you can enter your own categories. Three participants would like to click on the potential ripples that come from the dependencies. Three attendees did not realize that a user must first click the "Save" button to save a change and then "Save Scenario" to save the scenario with its changes. Instead, they first clicked on "save scenario" and wondered why an error message was displayed. It appeared that two of the participants were confused by the fact they could see scenarios of systems they did not create. Two of the seven participants read the information icons. Other participants asked the interviewer, when they did not understand what they needed to enter in an input field. The workflow seemed to improve after two scenarios created for three participants in terms of getting used to the method and speeding up the process of creating a scenario. The dialog for canceling scenarios did not seem clear to one of the seven participants what it means.
In this paragraph, we describe the answers to the question "What was easy or difficult to create a scenario?". Two attendees responded that the “Save Scenario” button was drawing too much attention, so the “Save” button would be missed to save a change. Two of the seven participants answered that they would like to expand the scenarios in the overview so that they can edit a single change. Three out of the seven respondents said it would be helpful to separate the scenarios from the changes. The abort dialogue is not clear, what exactly it means, reported two participants. Five of the seven participants responded that it is not clear that you cannot click on potential ripples, although this would be very helpful. Three participants responded that the user guidance was good and easy. One said the workflow was pretty slow. Two responded that a tagging system would be more helpful than categorization scenarios. An introduction, taxonomy or a tutorial would be helpful, three participants replied. One answered that it would be great, if the table in the scenario overview remembered, how it was sorted.

6.2.2.3 Evaluation

We observed that three participants were confused by the second drop-down menu to compare systems. Four out of the seven participants analyzed the entire page before

answering the questions, so they could answer all the questions. Others, who did not first analyze the entire review page, could not answer all the questions because they either did not recognize the table or the information about the entire system.

Next, we list the most meaningful answers given to the question “Which evaluation criteria are helpful?”:

- The number of affected services. (Three participants said it)
- Average effort per scenario. (Three participants said it)
- The critical service that is modified most but knowing additionally it would be helpful to know the number of changes made. Also, it would be good to know, in which scenarios this service is affected. (Two participants said it)

Now we list the most meaningful answers to “Which evaluation features are not needed?”:

- The average effort per service. (Four participants said it)
- The average effort per scenario. (Three participants said it)
- The hardest change. (Three participants said it)
- The easiest change. (Two participants said it)
- The total number of changes. (Two participants said it)

In the last part of this section, we describe what feedback we received for evaluating a system in our tool. Two participants mentioned that it would be great to compare diagrams with the interaction of the services of the evaluated system. So, the hypothetical changed system could be compared with the original system. Another participant replied that a headline for the attributes of the entire system might be helpful in identifying this part of the evaluation. Two participants answered that visualizing the attributes would be helpful to the overall system, such as a dashboard.

6.3 Discussion and Implications

In this section, we discuss our findings and observations and give implications of what these results and observations could mean. We will explore each area of our stated objectives, the usability we examined through the hands-on interviews, the appropriateness of the estimation methods, especially in the context of service-based systems that we studied

through the survey, and how useful our evaluation features are we explored through the survey and the hands-on-interviews.

6.3.1 Usability

We break this section down into three parts, one for each area of our tool, one in which systems are modeled, one for creating a scenario, and one for evaluating the system based on the scenarios.

Our impression in modeling systems is that the process of creating a system in our tool is simple and straightforward, as the participants in the hands-on-interviews rated the ease of creating systems with a mean of 1.29. The editing process also seems intuitive as five of the seven participants found the button to edit a system and a service to edit the service dependencies, name, or description. Only attendee one criticized that the editing is confusing, because according to him it is not possible to distinguish between creating a new service and editing a service when the extension field is open. An improvement we found out would be a graphical representation of the system. Another improvement is to shrink the add dependency drop-down menu.

Our improvement in adding a category to a scenario was not as helpful as we thought, as one participant thought it unnecessary and two participants recommended replacing it with a tagging system. Since most participants did not realize that they could add their own categories, it should be a good idea to set a given list of categories/tags. The problem with this approach could be that there is no literature on standard categories or tags and therefore our default recommendations may not be appropriate for a scenario. A user would be confused if nothing fits, like it was for two participants in the hands-on interview. As with the add dependencies drop-down menu during the creation process of a service, the drop-down menu for adding categories should be smaller than the current one, because four attendees did not recognize that the drop-down menu is scrollable due to the size of the drop-down menu.

The possible ripples are helpful as their helpfulness was rated by the participants with a mean of 4. The way they are visible is good, as they were recognized by all participants. However, they should be clickable because four participants tried to click on them. Maybe they should have a different color, depending on whether they are selected or not, because a participant was confused, whether they were preselected or not.

Although the process and interface for creating a system and creating a scenario are the same, the simplicity of creating systems does not apply to creating scenarios. For the scenarios, there was a big problem that four out of seven attendees were confused that they must first save the change before they can save a scenario. That may have two reasons. One of them is that the “Save Scenario” button gets too much attention, so the “Save” button is not recognized for saving a change. This was observed in four hands-on-interviews. Another reason might be that participants may not be familiar with the method because one participant wanted a taxonomy or tutorial, and another asked what a scenario in our context is. Our conclusion is that an exact taxonomy of a scenario should be displayed in the overview, as most participants passed the overview. In this way, users can know from the beginning what a scenario is in our method. The change creation process should be visually enhanced to make it clear that a change can and must be saved first.

Three participants indicated that it would be helpful to separate the changes from the scenarios. They also specify that they want to edit one change at a time, not the entire scenario. The scenario creation process seems mediocre, as three participants found the user guidance good. However, three other participants found the workflow slow. The editing process must be improved so that only one change can be edited, as explained above. One participant suggested that an extension of the scenarios overview would solve this problem, since there a single change could be selected and not much clarity is lost.

Another improvement to the scenario overview is that a particular system should be selected because displaying scenarios from other systems that a user has not created is confusing. For two participants this was the case with the hands-on-interviews. This selection and possibly other configurations should be saved because the user does not want to select them every time he creates a scenario. The abort dialogue should be more meaningful, as this is not the case and causes confusion. One participant said it was not clear if the process would be stopped and resumed later or if all data would be discarded.

The user guidance of the evaluation view in our tool seems to be good as five participants were able to answer all the questions. One participant did not see the bottom of the view, which has a scenario information table, so he could not answer the questions about the scenarios. After the interviewer recommended that there is a table, the participant was able to answer the questions about the scenarios. Another participant did not recognize

the upper part of the evaluation view and therefore could not answer the questions about the whole system. He took it for a simple text and paid it no attention.
Another argument for the seemingly user-friendly guidance is that the average time to answer the questions for evaluating the system at a standard derivation of 1.39 minutes is 3.57 minutes. This means that all participants took a similar time to answer the questions. Probably because the evaluation view is clear, and the evaluation features can be found quickly.
To improve the usability of the evaluation view, it can be said that more visualization of the features is required because one participant of the hands-on-interview criticized that the first part is a text desert. It makes sense to create visualization features as they can be found in dashboards. A graphical representation of the current system and the change system would also be good for a quick comparison of what changes were made in the two service-based systems as suggested by two participants in the hands-on-interviews.

6.3.2 Effort Estimation Methods

In this paragraph we refer to the survey results we filtered because we want to get meaningful results. We only consider answers from participants whose familiarity with a method was above or equal to three for the method.
The method of estimating the effort of a change in lines of code is not appropriate for our tool-supported method, as the survey respondents ranked it on an ordinal scale of 1 to 5 in terms of precision and applicability less than average, meaning less than 3. It is noteworthy that a group of experts described in 6.1.2.1, which results are shown in Figure 9, evaluated the lines of code in terms of precision and applicability at 3.75 each. This may mean that a person needs above-average work experience, knowledge of service-based systems, and scenario-based methods, so that the lines of code method is applicable in service-based systems and the effort can be accurately estimated. Above average here means above average of the mean value of the respective value of all participants. On the other hand, this means that the majority of respondents think that this metric cannot be applied to service-based systems, as shown in Figure 8, as the lines of code method is worst for all participants in the survey in terms of precision and applicability.
The story points seem to be the best method of the five effort estimation methods, considering their precision and applicability, as the participants rated them with a mean of 3.55 and 3.83. In a group where the participants worked with a scenario-based method,

the story points were rated 3.87 and 3.93 in precision and applicability. The expert group rated story points with even 4.4 and 4.6. This shows that people who are familiar with scenario-based methods or have worked with them in real projects find that this effort estimation methodology for change is accurate and applicable to service-based systems.
The ordinal scale has the highest applicability as an effort estimation method in each group we consider useful. These are the total number of participants, the expert group and the group that used a scenario-based method in real projects. However, the precision of the effort estimation method is only mediocre since it was rated between 3.13 and 3.33 in the three groups studied.
The effort estimate in hours was also judged to be quite good in terms of accuracy and familiarity. We think that one reason is that most participants are well acquainted with the effort estimate in hours, since the familiarity of the method is 4.14. This is the highest familiarity value of the five effort estimation methods we rate.
Cosmic function points were evaluated by all participants, the group that used a scenario-based method in real projects, and the expert group, with the second highest accuracy. If a person uses a scenario-based method in a real project, the person judged the applicability of this method on average 3.5. Persons who are more familiar with scenario-based methods rated the applicability with an average of 4.0. It should be noted that the mean familiarity of all participants is 1.61 and we have considered only participants with a familiarity greater than or equal to three in the data of precision and applicability.

Our interpretation is that, according to survey respondents, story points are well suited as a cost estimation method for change, as they are well judged in terms of precision and applicability. Most participants in the survey also knew this method, as the mean of familiarity with 3.56 is second. The other cost estimation methods, with the exception of the lines of code method, each have their advantages and disadvantages. The ordinal scale was ranked as the most applicable method but has only a mediocre accuracy. Most participants knew the effort estimate in hours because the mean of familiarity is 4.14. However, this method was only judged between mediocre and good in terms of precision and applicability. Cosmic function points are pretty accurate, but the participants are not familiar with them and their applicability is mediocre. As mentioned earlier, the method of estimating the code overhead for our tool is not appropriate. Therefore, we may consider removing this method from our tool.

In the last paragraph of this section, we must note that there are no correlations between a participant's background and the five effort estimation methods. The differences in the group to people who worked in a real-world project with a scenario-based method, the expert group and all participants could therefore be a coincidence.

6.3.3 Evaluation Features

Our idea of comparing systems should be improved. At the moment it is confusing whether a second system can be selected in the evaluation view. This was the case with three participants. A quick fix would be to replace the selection with a button that displays the selection when a user clicks on it.

According to the survey results, the critical service that is changed most is most useful from the five rated attributes that we have rated by the survey participants. The mean is close to 3.9, which means good. The most expensive critical service, the average effort per scenario, and the change that affects most services are also useful, as they were rated nearly 3.5 each. The number of services affected does not seem to be particularly useful as it was rated 2.65, which is less than 3, which is below average. On the other hand, the participants of the hands-on-interviews rated the number of affected services as useful, as three participants indicated. The average cost per scenario was either useful or unnecessary, as three participants considered this useful and four participants unnecessary. We rely more on the survey results as the average familiarity of service-based systems was 4.1. The average familiarity of service-based systems of participants from the hands-on-interviews was only 2.43. Therefore, participants of the hands-on-interviews may not appreciate exactly which rating functions are well-suited for service-based systems.

These statements imply that the evaluation features that we rate in the survey are useful to evaluate a service-based system. We believe that the number of affected services could be useful, even though it was rated as below average by respondents, as three participants in the practical interviews considered it useful. We may consider removing evaluation features that deemed two or more participants of the hands-on interview unnecessary. These features are the average effort per service, the most difficult change, the simplest change, and the total number of changes. The average cost per scenario was also considered unnecessary but was rated as useful by three participants in the hands-on-interviews. This feature was rated as good in the survey. Therefore, removal is not considered.

The survey gave us valuable feedback for adding more evaluation features. From our point of view, the distribution of service changes, and the scenario that affects most services might be appropriate.

6.4 Derived Suggestions for Improvements

Based on the discussions and implications we can derive another improvement list. First, we describe the improvement and the reason for it. We then present a table in which we evaluated the improvement and its complexity for implementation.

Improvement 1: Adding a graphical representation of the created system in the third step of the creation process and in the system detail view
This enhancement helps to quickly understand the entire system and its dependencies. Although a large system can be obscure in one view. Two participants of the hands-on-interviews mentioned that this improvement is helpful.

Improvement 2: Shrink the drop-down menu to add dependencies
The clarity of the menu and thus the usability is improved. Two participants of the hands-on-interviews mentioned that this improvement is helpful.

Improvement 3: Replace scenario categorization with a tagging system
As the hands-on-interviews show, a tagging system would be helpful since it was mentioned by two participants.

Improvement 4: Shrink the drop-down menu for adding categories / tags
The clarity of the menu and thus the usability is improved. Two participants of the hands-on-interviews mentioned that this improvement is helpful.

Improvement 5: Structure the drop-down menu for adding categories / tags by adding sub-headings
One participant in the hands-on-interviews recommended that it could help clarify when the drop-down menu is better organized by sub-headings.

Improvement 6: Make the possible ripples clickable so they are added when clicked
In the hands-on-interviews, it was pointed out that the potential ripples look clickable and wondered why they are not. It improves workflow and ease of use.

Improvement 7: Design the second step to create scenarios to make the step more intuitive. Implement the improved creating process

In the second step of creating scenarios were a lot of confusion in the hands-on-interviews, so the usability of this step needs to be improved.

Improvement 8: Write the exact taxonomy and process for creating scenarios in the overview so that the user understands the method of creating scenarios

Since most participants in the hands-on-interviews carefully read the overview page, it makes sense to note down the exact taxonomy of the scenarios to make our method clearer.

Improvement 9: Improve the editing process of a scenario so that a user can only edit one change. This improvement requires a menu to select a change

Two participants in the hands-on-interviews said it would be useful if a change could be added and not the entire scenario.

Improvement 10: In the scenario overview, add a selection for a system so that not all scenarios are displayed for all systems

Since two participants in the hands-on-interviews were confused by the overall scenarios of all systems in the overview, it makes sense to select a system before displaying scenarios.

Improvement 11: Let the summary table remember how it is sorted by which attribute

For ease of use, it is helpful to remember the summary table as it was sorted. Also, one participant reported that in the hands-on-interviews.

Improvement 12: Improve the abort dialog to create scenarios and systems to make it easier to understand

Since two participants in the hands-on-interviews were confused as to what the abort dialogue means, the note text needs to be clearer.

Improvement 13: Remove the lines of code estimation method

As the survey indicates, this metric is not suitable for our purpose.

Improvement 14: Add a button that makes the comparison system selection visible or remove this feature completely

The system comparison is more confusing than helpful, as the hands-on-interviews have shown. It is not well implemented so it can be hidden or removed.

Improvement 15: Add test complexity as a cost estimation method

One participant in the survey recommended it and the general trend is that abstract effort estimation methods are well suited for our purpose, so this could be another good effort estimation method.

Improvement 16: Remove the average cost of services and the most difficult and simplest change from the evaluation view in our tool

As shown in the hands-on-interviews and the survey, these evaluation features are not helpful.

Improvement 17: In the evaluation view, add a better visualization of the data compared to plain text

As a participant of the hands-on-interviews criticized that the information of the entire system is a text waste, some graphics could solve this problem. For example, add a dashboard bubble for the most critical service that has changed the most and show the relative part how it has changed in proportion to the number of changes.

Improvement 18: In the evaluation view, add a graphical representation of the evaluated system and the hypothetical system

For a quick comparison of the changed system and the selected system, a graphical representation of the selected system and the hypothetical system may be a good way to realize this. This also mentioned two participants in the hands-on-interviews.

Improvement 19: Add the distribution of service changes in the evaluation view

This improvement makes it easy to see which services are changed most and which are not changed often. One participant mentioned this in the survey.

Improvement 20: Add test coverage of existing features in the evaluation view

One participant in the survey suggested that it might be helpful to know the test coverage of existing features.

ID	Improvement	Added value (1 to 5)	Complexity (1 to 5)	Is selected for implementa-tion
1	Graphical representation of the created system	5	5	X
2	Shrink the drop-down menu to add dependencies	2	1	X
3	Replace scenario categoriza-tion with a tagging system	3	3	X
4	Shrink the drop-down menu for adding categories	2	1	X
5	Structure the drop-down menu for adding categories	2	3	X
6	Make the possible ripples clickable	3	3	X
7	Redesign and implement the second step of the scenario creation process	4	3	X
8	Add an exact taxonomy of the method in the overview	3	1	X
9	Improve the editing process of a scenario so that a user can only edit one change	2	4	
10	Add a selection for systems in the scenario overview	3	3	X
11	Let the summary table re-member how it is sorted	2	4	
12	Improve the abort dialog	2	1	X

13	Remove the lines of code estimation method	3	1	X
14	Add a button that makes the comparison system selection visible or remove this feature completely	2	1	X
15	Add test complexity as a cost estimation method	2	4	
16	Remove the average cost of services and the most difficult and simplest change from the evaluation view	2	1	X
17	Add a better visualization of the data	5	4	X
18	Add a graphical representation of the evaluated system and the hypothetical system	4	5	
19	Add the distribution of service changes	3	2	X
20	Add test coverage of existing features	2	4	

Table 7 Derived suggestions for improvements.

We selected the improvements that should be implemented according to the reason that the improvement is either very easy to implement or provide high added value in exchange for their implementation cost.

7 Conclusions

This chapter presents a summary of the work, the threats of validity and future work.

7.1 Summary

We have shown how an existing tool-supported method for evaluating service-based systems works with a scenario-based assessment. This existing method has been systematically improved by us. We have shown what, in our view, can be improved and what we have implemented. We evaluated this improved tool-supported method by a qualitative assessment with hands-on-interviews and a quantitative assessment with a survey. Our most important finding in the evaluation is that the lines of code estimation is neither very accurate nor applicable and therefore not suitable for our tool-supported method according to the participants of the survey. The story points, according to the survey participants, are quite good at estimating the cost of a change, as they best assessed their precision and applicability among the five effort estimation methods given. We found no correlation between a participant's personal background, whether a participant worked in real projects with scenario-based evaluation, or his familiarity with service-based systems and our effort estimation methods and evaluation functions. Our evaluation view and the evaluation feature it contains seem to be helpful in evaluating a service-based system, as it was rated as helpful by the participants in the hands-on-interviews with 4.41. By evaluating our tool-based method, we have created a derived enhancement list that can be used for future improvements.

7.2 Threats to Validity

Our sample size of the survey and the hands-on-interviews was very small, so the results may not be meaningful. The user group of the hands-on-interviews consisted of Ph.D. students with professional experience in various sectors, but this did not cover a large number of industries. This means that the results in the hands-on-interviews are limited to the surveyed group and perspectives from several industries were omitted.

We tried to explain exactly how our tool-supported method works in the survey. However, there may be participants who do not understand a method or evaluation feature. So, the participant could answer weirdly.

The order in which we asked the questions was the same for all participants. In the survey, there could be an influence between the questions that lead to biased results. One way to avoid such interference is to randomize the order of the questions asked. Each participant sees a different sequence of questions.

Participants of the survey never interacted with the tool. We just showed them screenshots of the evaluation features and described the estimation methods. Because of this, a participant may not understand how our method works, and therefore estimates the estimation methods and evaluation features to be strange.

To assess the accuracy and applicability of an effort estimation method, only responses from participants whose knowledge of the method is at least three were considered. But how familiar a participant is, is valued by himself. He may not be as familiar as he thinks and therefore he cannot judge the precision and applicability well.

We asked respondents to rate an effort estimation methodology based on familiarity, precision, and applicability, and to limit answers whose familiarity in our assessment is less than three. The problem is that these properties may not be enough, and we could actually better narrow down the answers by asking the participant for more properties for the method, such as: For example, if you have estimated the cost of changing with this method in a real project.

We did the hands-on interviews at different times of the day. The participants may be influenced by the time we conducted the interview. For example, participants who conducted the interview in the morning may be more willing to give in-depth feedback than participants who did the interview at noon or in the late afternoon because they would like to finish quickly to go for lunch or end of work.

We conducted the survey online so that one participant could use different devices to answer the survey. The survey may have been displayed differently on each device, and the participant may have been affected.

7.3 Future Work

As we have worked out a further improvement list, this could be a good starting point for future work.

We believe that we can automate the process of creating systems such as the KAMP tool described in the related work. Probably the scenario creation process can also be automated. As with KAMP, a graphical representation of the system could be modified to generate scenarios with their changes.

Integrating software planning systems like JIRA to export changes to a task or a user story could be a good idea.

As the practical interviews have shown, the usability of the tool can be further improved.

If the tool is further improved, a further large-scale evaluation with a large and meaningful sample size should be performed to see how another set of participants rate the method, as the theory is very different from practice as our evaluation has shown.

In our tool-supported approach, we focus on modifiability, but other attributes of software quality attribute maintainability can be covered, such as: Testability, analyzability and reusability.

Bibliography

Babar, M., & Gorton, I. (2004). Comparison of scenario-based software architecture evaluation methods. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, 600-607.

Bengtsson, P. (2002). Architecture-Level Modifiability Analysis. *Department of Software Engineering and Computer Science Bleking Institute of Technology*.

Bengtsson, P., & Bosch, J. (2000). An Experiment on Creating Scenario Profiles for Software Change. *Annals of Software Engineering, 9*(1, 2), 59-78.

Boehm, C., Boehm, B., Clark, B., Horowitz, E., Westl, C., Madachy, R., & Selby, R. (1995). Cost models for future software life cycle processes. *Annals of Software Engineering, 114*(171), 64-94.

Coelho, E., & Basu, A. (2012). *ISSN : 2249-0868 Foundation of Computer Science FCS.* East Point College of Engineering and Technology, Bangalore, Karnataka.

Kazman, R., Bass, L., Abowd, G., & Webb, M. (2002). SAAM: a method for analyzing the properties of software architectures. 81-90.

Koziolek, H., Domis, D., Goldschmidt, T., Vorst, P., & Weiss, R. (2012). MORPHOSIS: A lightweight method facilitating sustainable software architectures. *Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012*, 253-257.

Matson, J., Barrett, B., & Mellichamp, J. (1994). *Software Development Cost Estimation Using Function Points.*

Matson, J., Barrett, B., & Mellichamp, J. (1994). Using Function Points. *IEEE Transaction on Software Engineering, 20*(4), 275-287.

Ostberg, J., & Wagner, S. (2014). On automatically collectable metrics for software maintainability evaluation. *Proceedings - 2014 Joint Conference of the International Workshop on Software Measurement, IWSM 2014 and the International Conference on Software Process and Product Measurement, Mensura 2014* (pp. 32-37). Institute of Electrical and Electronics Engineers Inc.

Richards, M. (2016). *Microservices Architecture.* (Nan Barber, & Rachel Roumeliotis, Eds.) O´Reilly Media.

Rick Kazman, Mark Klein, P. (2000). *ATAM: Method for Architecture Evaluation.* Pittsburgh: Carnegie Mellon Software Engineering Institute.

Romano, D., & Pinzger, M. (2012). Analyzing the evolution of web services using fine-grained changes. *Proceedings - 2012 IEEE 19th International Conference on Web Services, ICWS 2012*, 392-399.

Rostami, K., Stammel, J., Heinrich, R., & Reussner, R. (2015). Architecture-based Assessment and Planning of Change Requests. *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures - QoSA '15*, 21-30.

Schach, S. (2011). *Object-Oriented and Classical Software Engineering Eighth Edition* (8 ed.). New York: Raghothaman Srinivasan.

Tekinerdoğan, B. (n.d.). ASAAM : Aspectual Software Architecture Analysis Method Department of Computer Science , University of Twente ,. *Department of Computer Scien ce, University of Twente, P.O. Box 217 7500 AE Enschede, The Netherlands*, 10.

Usman, M., Mendes, E., Weidt, F., & Britto, R. (2014). Postprint Effort Estimation in Agile Software Development : A Systematic Literature Review. *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, 82-91.

Woods, E. (2012). Industrial architectural assessment using TARA. *Journal of Systems and Software, 85*(9), 2034-2047.

Independence declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Date and Signature:

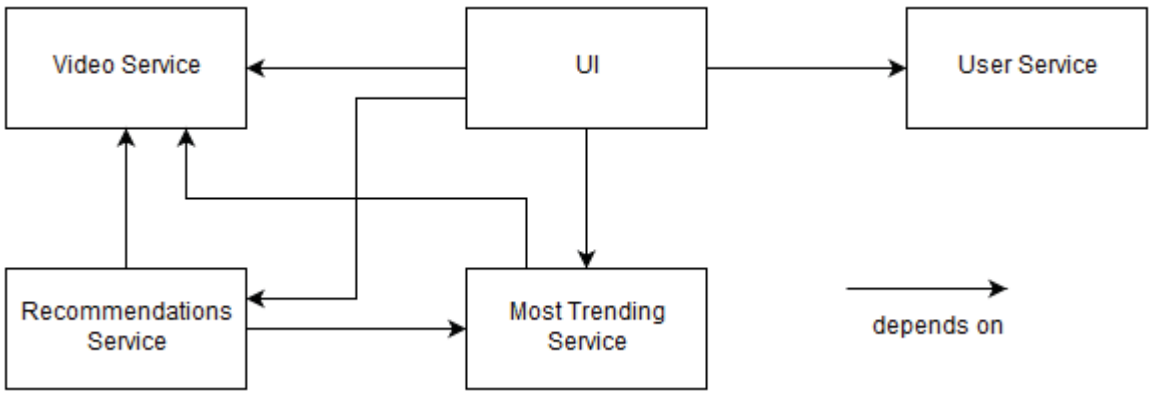
Appendix

Hands-On-Interview: Evolution of a Video Platform

System description:

Imagine a system where a user can watch videos and also upload his own videos. The system calculates the most popular videos and shows them in a tab in the UI. A user can browse through all videos in the UI. While watching a video, the system gives recommendations what other videos the user may want to watch based on the currently watched video and recent trends.

Service	Description	Dependency
User service	Through this service, a user can log in and manage his data.	-
Video service	The Video service is responsible for video uploading and showing videos to the user.	-
Most trending service	This service calculates the most trending videos based on recently watched and liked videos.	Video service
Recommendations service	The Recommendations service presents the recommended videos to the watched video based on likes and trends.	Video service Most trending service
UI	The UI takes care of everything the user sees.	Video service User service Recommendations service Most trending service



➔ Task: Create this system and its services in the tool.

Scenario description:

Market changes, security concerns, and new technology may require changes to your video platform in the future. Therefore, you thought of the following hypothetical evolution scenarios and the potential impact they may have on the system.

- To improve performance, the database management system in the Video Service should be exchanged.
- The system should be extended with instant messaging functionality. For that, a new service called Instant Messaging Service must be created.
- The complex Most Trending Service has become fairly large and unmaintainable. To increase its analyzability, the Most Trending Service should be rewritten in Python using a very convenient library.
- Due to security concerns and regulations, two-step verification should be implemented. This requires additional attributes in the user service and also affects the UI.

➔Task: Create these scenarios and their corresponding change(s) in the tool.

Scenario Evaluation:

After finishing scenario creation, go to the evaluation tab in the tool, select the created system, and try to answer the following questions.

- Which service is changed the most?
- Which service is associated with the highest scenario efforts?

- Which is the total potential scenario effort for the system?
- Which scenario is associated with the highest effort?
- Which scenario changes most services?

Hands-On-Interview: Evaluation

Creating system

Duration for creating the system:

Observations:

How easy was it to create the system on a scale of 1 to 5, with 1 being simple and 5 being hard?

What was easy or difficult to create a system?

What do you think about defining dependencies in this way?

Creating scenarios

Duration for creating the scenarios:

Observations:

Do you find it helpful to categorize scenarios on a scale of 1 to 5, where 1 is not useful and 5 is very useful?

What was easy or difficult to create a scenario?

Do you find the recommended ripples helpful on a scale of 1 to 5, where 1 is not useful and 5 is very helpful?

Created Scenarios:

Evaluation

Duration for evaluating the system based on the scenarios:

Observations:

Which valuation criteria are helpful?

Which evaluation features are not needed?

Is the evaluation view helpful for rating the system on a scale of 1 to 5, with 1 not helpful and 5 very helpful?

Feedback:

Personal information

- ID:
- Years of professional experience:
- Have you used a scenario-based method for real-world projects before?
- How familiar are you with scenario-based methods in general on a scale from 1 to 5, where 1 is not familiar and 5 very familiar?
- How familiar are you with service- or microservice-based systems in general on a scale from 1 to 5, where 1 is not familiar and 5 very familiar?

Survey: Scenario-based Modifiability Analysis of Service-based Systems

Section 1: Scenario-based Modifiability Analysis of Service-based Systems

To analyze how well a service-based system would adjust to changes in the future, we are working on a lightweight scenario-based evaluation method specifically designed for service orientation. We also develop tool support for this method and want to align it with software professionals' needs.
Within the scope of such a method, we'd like to hear about:
- your opinion on different ways to estimate the scenario effort (6 questions)
- your opinion on different metrics supporting the modifiability analysis of a system (6 questions)
- your personal experience with these topics (5 questions)

In total, there are 17 questions which should take approximately 5-7 minutes. Results are collected anonymously. Should you have any questions, or should you be interested in the results, please contact justus.bogner@reutlingen-university.de. Feel free to distribute to your colleagues.
Thank you for your participation!

Section 2: Change Effort Estimation

In our tool, you can create "Scenarios". Each "Scenario" consists of at least one "Change". For estimating the effort of these "Changes", different techniques can be used.
For each variant, we'd like to hear your experience concerning
- how familiar you are with this technique (Familiarity)
- how precise this technique is (Precision)
- how applicable this technique is for effort estimation in a lightweight scenario-based method for service-based systems (Applicability)

On the scale from 1 to 5, 1 represents "not precise", "not familiar" and "not applicable" while 5 represents "very precise", "very familiar" and "very applicable".

Effort in “Hours of development”:

A common way to estimate effort is by using development hours, i.e. the duration it takes to implement the "Change".

Your experience with “Hours”

	1 (not familiar, not precise, not applicable)	2	3	4	5 (very familiar, very precise, very applicable)
Familiarity					
Precision					
Applicability					

Effort as “Lines of Code”:

Another common technique to estimate the needed effort for a "Change" is lines of code (LOC).

Your experience with “Lines of Code”

	1 (not familiar, not precise, not applicable)	2	3	4	5 (very familiar, very precise, very applicable)
Familiarity					
Precision					
Applicability					

Effort as an “Ordinal Scale”:

With an ordinal scale from 1 to 10, the effort and complexity of a "Change" can be described, where 1 is easy and not time-consuming and 10 very difficult and very time-consuming.

Your experience with “Ordinal Scale”

	1 (not familiar, not precise, not applicable)	2	3	4	5 (very familiar, very precise, very applicable)
Familiarity					
Precision					
Applicability					

Effort as “Cosmic Function Points”:

The COSMIC method defines principles, rules, and a process for measuring the functional size of software. "Functional size" is a measure for the amount of provided

functionality and completely independent of the implementation.

More details about the technique can be found here: https://cosmic-sizing.org/cosmic-fsm

Your experience with “Cosmic Function Points”

	1 (not familiar, not precise, not applicable)	2	3	4	5 (very familiar, very precise, very applicable)
Familiarity					
Precision					
Applicability					

Effort as “Story Points”:

A common way to estimate effort is by using development hours, i.e. the duration it takes to implement the "Change".

Your experience with “Story Points”

	1 (not familiar, not precise, not applicable)	2	3	4	5 (very familiar, very precise, very applicable)
Familiarity					
Precision					
Applicability					

What other effort estimation techniques would you consider useful in a lightweight scenario-based method?

Section 3: Scenario-based Modifiability Analysis of Service-based Systems

Evaluation Attributes – Part 1

The following picture is a screenshot from the evaluation view of our tool. Several metrics and attributes about the entered system and "Scenarios" are displayed. We'd like to hear your opinion about the usefulness of these information in the context of modifiability analysis.

Evaluation View for the “Uber” Example System:

System name:
Uber

Critical service that is modified most: Passenger management

Critical service with highest overall effort: Driver Web UI

Service with lowest effort: Payments

Summarized effort for the total system: 557 Story points

Average effort per service: 61.89 Story points

Average effort per scenario: 92.83 Story points

How useful is information about the critical service that is most modified?

	1	2	3	4	5	
Not useful						Very useful

How useful is information about the critical service with the highest effort?

	1	2	3	4	5	
Not useful						Very useful

How useful is the average effort per “Scenario”?

	1	2	3	4	5	
Not useful						Very useful

Section 4: Scenario-based Modifiability Analysis of Service-based Systems

Evaluation Attributes – Part 2

The following picture is a 2nd screenshot from the evaluation view of our tool. A table of all "Scenarios" with some metrics is displayed. "Scenarios" consist of at least one "Change". We'd like to hear your opinion about the usefulness of the following information in the context of modifiability analysis.

Evaluation Table: List of all Scenarios

Scenario ↓	Most Impactful Change	# of Affected Services
More notifications	modification:Notification when driver has arrived / affects 5 services	5 / 9
Design and implement more user-friendly UI	modification:Change Web Framework / affects 3 services	4 / 9

How useful is information about the “Change” that impacts the most services?

	1	2	3	4	5	
Not useful						Very useful

How useful is the number of affected services for each “Scenario”?

	1	2	3	4	5	
Not useful						Very useful

What other evaluation attributes would you consider useful in a lightweight scenario-based method?

Section 5: Scenario-based Modifiability Analysis of Service-based Systems

Professional Experience:

In this section, we like to ask some general professional details about you.

In which field(s) are you currently active?

	Industry
	Academia
	Both
	Other:

How many years of professional experience do you have?

Have you used a scenario-based method for a real-world project before?

	Yes
	No

How familiar are you with scenario-based methods in general?

	1	2	3	4	5	
Not familiar						Very familiar

How familiar are you with service- or microservice-based systems in general?

	1	2	3	4	5	
Not familiar						Very familiar