Florian Schierlinger-Brandmayr, BSc

Engineering of a Toolkit to Support Ecosystem Analysis and Design

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Wolfgang Vorraber

Department of Engineering- and Business Informatics

Dipl.-Ing. Birgit Mösl, Bsc

Graz, May 2019

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

Acknowledgment

I would like to thank everyone who has supported me during the generation process of this thesis as well as during my studies.
Especially, I would like to thank Ass.Prof. Dipl.-Ing. Dr.techn. Wolfgang Vorraber for his supervision, great ideas, numerous advices and my introduction to the lived field of business informatics in the department of Engineering- and Business Informatics.
I would like to highlight my second supervisor, Dipl.-Ing. Birgit Mösl, Bsc, for the various update meetings, instant guiding answers concerning content and organization.
Without my two supervisors, my created thesis would not exist as it does now.

I would like to thank my family, especially my parents, Andrea and Karl, sisters, Christina and Anna-Lisa, and grandparents, for enabling my studies and supporting me during my entire study program.

Finally, I would like to thank Julia Buchmayer for the honest feedback sessions, for finding errors in my first version of this work and for the support within all the semesters of our studies.

Abstract

In this thesis, a newly developed visualization tool for value networks is introduced. The tool visualizes the V^2 value network notation and is designed to be individually extendable in the future. Leading to the actual implementation of the visualization tool, selected research areas as well as main concepts, which emphasize the need for value networks, are discussed.

The disciplines dealing with complex systems, considering their design, development and maintaining phases, are a widely discussed topic. Requirements engineering is a mainly used approach to identify and document systems' needs. The key phases during the requirements engineering process include the system analysis, the formulation of requirements and the appropriate documentation, requiring detailed knowledge about the stakeholders' needs. As important is the expertise about using precise documentation techniques to express the core idea behind each requirement. There are various possibilities, from pure linguistic representations until graphical representations. Requirements engineering tries to identify the (stakeholders') needs of a system. To emphasize the role of stakeholders and to arrange the engineering process according to their needs, stakeholder-centered engineering is discussed. Used methodologies in this area range from pure stakeholder analysis techniques, like the power versus interest grids and stakeholder influence diagrams, to philosophies, e.g. service design thinking. The stakeholder analysis techniques emphasize the need for understanding the stakeholders in their environments. Service design thinking is an approach, which uses the insights of the stakeholder analysis to design a service fulfilling the needs of the stakeholders. Systems engineering is the last big picture, which shifts the focus to the overall engineering process to be able to deal with complex systems. It involves basic philosophies, such as system thinking. Top-down approaches as well as black/white-box-oriented thinking are used to design systems in a systematic and structured way. The research field also references requirements engineering and the stakeholder centered engineering.

To meet the needs of analyzing systems and their environments, while focusing on the stakeholders' perspective, a browser-based visualization tool for the V^2 value network notation has been implemented. The focus of this notation lies on the value-based exchanges in a stakeholder network. Legal as well as motivational aspects get modeled, too. The visualization tool offers a client-sided browser tool for smoothly generating stakeholder networks and looking at the same network from different perspectives. Built-in functions, as layer switching and inline text-editing, enable an easy creation process of a linked stakeholder network with its corresponding value exchanges. The tool itself is the base for further research investigations, including a future simulation model, based on the created graphs.

Zusammenfassung

In dieser Arbeit wird ein neu entwickeltes Visualisierungstool, welches eine graphische Darstellung der V² Wertschöpfungsnetzwerk-Notation ermöglicht und in naher Zukunft individuell erweiterbar sein soll, vorgestellt. Hinführend auf die Implementierung des Visualisierungstools wird ein Überblick über ausgewählte Forschungsgebiete und Konzepte, welche die Notwendigkeit von „Value Networks“ unterstreichen sollen, geschaffen.

Die immer komplexer werdenden Systeme, welche sich mittlerweile aus interdisziplinären Gebieten zusammensetzen, stellen die Forschung vor neue Herausforderungen. Um diesem Trend entgegenzukommen, bietet der erste Teil dieser Arbeit einen Überblick über wichtige Modelle in der Systementwicklung. Zu Beginn wird das „Requirements Engineering“ analysiert, wobei die Systemanalyse, die qualitative Definition von Anforderungen, sowie deren entsprechende Dokumentation von großer Bedeutung sind. Wichtige Aspekte der Anforderungsdokumentation, wie beispielsweise sprachliche Barrieren, werden besonders hervorgehoben.

Um den Fokus von den Anforderungen eines Systems auf die gezielte Entwicklung in Bezug auf die Stakeholder zu legen, wird der stakeholderzentrierte Gestaltungsprozess genauer betrachtet. Analysemethoden, wie die Stakeholder-Einfluss-Diagramme, sollen die Abhängigkeiten zwischen den einzelnen Personengruppen aufzeigen. Ein gesamtes Konzept der Stakeholder-zentrierten Systemgestaltung wird in der Philosophie „Service Design Thinking“ gelebt. „Service Design Thinking“ bietet eine Reihe von Methoden und Werkzeugen, welche die Gestaltung und Entwicklung von Serviceprodukten voll und ganz den Bedürfnissen der Stakeholder verschreibt. „Systems Engineering“ hingegen legt den Fokus auf die gesamteinheitliche Gestaltung von (komplexen) Systemen und deren interdisziplinäre Vernetzung. Die Philosophie beschreibt unter anderem strukturierte Vorgehensweisen, wie etwa die Schritt-für-Schritt-Verfeinerung von Systemteilen. Auch „blackbox“-Ansätze, wo bewusst detaillierte Einblicke in interne Systemverhalten ignoriert werden, gehören zu der Grundidee.

Um die Stakeholder-zentrierte Möglichkeit einer System(umwelt)-Analyse zu unterstützen, wurde ein Visualisierungstool für die V² Wertschöpfungsnetzwerk-Notation geschaffen. Dieses Werkzeug ist browser-basiert und ermöglicht eine rasche Visualisierung eines sogenannten „Value“-Netzwerkes. Es ist zudem möglich, einzelne Ebenen des Netzwerkes ein- bzw. auszublenden, um verschiedene Sichtweisen mehrerer Graphen in einer Gesamtübersicht zu vereinen. Das Visualisierungstool ermöglicht eine rasche und detaillierte Analyse der wichtigsten Einflussfaktoren in (großen, komplexen) Systemen. Die aktuelle Version des Tools bietet eine hervorragende Ausgangsposition für zukünftige Erweiterungen, inklusive einer angedachten Simulationsschnittstelle.

Content

List of Abbreviations

CSS	Cascading Style Sheets
DOM	Document Object Model
DSGVO	Datenschutz-Grundverordnung
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
INCOSE	International Council on Systems Engineering
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
MPL	Mozilla Public License
MVC	Model View Controller
PNG	Portable Network Graphics
RUP	Rational Unified Process
SysML	OMG Systems Modeling Language
UML	Unified Modeling Language
XP	eXtreme Programming

List of Tables

List of Figures

1. Introduction

The emerging complex systems represent a networked and interdisciplinary research field. Multiple system parts, coming from different practical fields, interact with each other and need to be designed to work hand in hand. Embedded in their environment, key aspects of stakeholders mainly influence the system itself. This work tries to give an overview of baseline techniques needed for a structured work on such systems, including their environment.

Beginning with requirements engineering in chapter two, the key concept of researching, documenting and maintaining stakeholder requirements is discussed. Core principles, like defining qualitative requirements, whereas potential linguistic problems and unclearness is removed, aligning requirements engineering with various project procedure models until documenting and maintaining the generated knowledge are some of the main points at the start of this work.

Chapter three focuses on stakeholder analysis and its importance to the engineering process with the help of an emerging philosophy, called service design thinking. (Stickdorn and Schneider, 2013; Stickdorn *et al.*, 2018) This technique focuses on human factors and allows generating various artefacts to support the engineering process. To complete the picture of the overall engineering of systems, in chapter four the research field of systems engineering has been further investigated. This field offers the possibility to combine software and classical systems engineering as an interdisciplinary approach. It references the field of requirements engineering as part of the complete system designing and analyzing process. The main aspect of current systems engineering research fields is about using model-based approaches to better deal with complex, networked subparts. (Haberfellner *et al.*, 2018)
For analyzing those complex systems, the stakeholder-centered analysis in the very beginning is a substantial part for a successful project, business or service.

A useful tool to investigate stakeholders and their interactions is the value network notation, or shortly V^2, proposed by Vorraber (2019) and Vorraber and Vössner (2011). By now, there was no visualization tool, which fulfills the criteria for the intended visual elements. In order to close this gap, the practical part of this work was the implementation of a visualization tool for the V^2 value network notation. The tool has been programmed to enable the user a quick and easy way to generate a viewpoint representation of stakeholders, build a network and analyze these networks concerning different layers. The practical part of this work in chapter five offers a step by step explanation, how the tool has evolved, what the key intention behind it is and what main functionalities it offers. With the help of the visualization tool, value networks can be smoothly created. This offers the possibility for detailed insights into the requirements and development issues of a system.

2. Requirements Engineering

Requirements Engineering is a substantial part in the documentation process of what a system should be capable of. Before analyzing the system's environment, the discussion, definition and documentation of the system's requirements is a necessary starting point.

2.1. Definitions

This chapter mainly focuses on the reason why requirements engineering is necessary and what is covered by it.

2.1.1. What is Requirements Engineering?

There are several definitions in the field of requirements engineering, but most of them focus on the same core factors. Requirements engineering is a discipline which builds bridges between the (product) creator and the (product) user. The main aim is to establish a well-defined requirements analysis as well as a working requirements management. (Rupp, 2004, pp. 498–501; Institute of Electrical and Electronics Engineers, 1990)
The core of requirements engineering is the establishment of a well-suited requirements analysis. Taking a deeper look at the requirements analysis itself, there are differing opinions about what is part of the requirements analysis task. Institute of Electrical and Electronics Engineers (1990) and Rupp (2004, pp. 498–501) talk about the process of defining and formulating the user's requirements. There are also some sources which want to include the task of validating those generated requirements. (Rupp, 2004, pp. 498–501)

The definition of requirements engineering shows that it is all about describing the needs or expectations of customers for a certain system. But that is solely a certain task in the big project of doing requirements engineering. (Rupp, 2004, pp. 2–3)
For that reason, the whole process will be rolled up. Some important steps/questions will be (Rupp, 2004, pp. 2–3):

- Why do we need requirements engineering?
- What are requirements?
- How can requirements be described and formalized?
- How can requirements be documented?
- How can requirements be managed and maintained?
- Are there good practices how to do requirements analysis?
- Outline for practical usages.

2.1.2. Reasons for Requirements Engineering

The reason, why a systematic requirements engineering should be part of every project, is an obvious one: The later changes in specification have to be done, the more expensive and time exhausting they get. Additionally, defining requirements in a structured and reasonable way offers a guided path from the project idea to the concrete product or service with a clear description what should be part of it. An important task of requirements engineering is also the description of quality measurements, which should be fulfilled by certain requirements. (Rupp, 2004, pp. 10–11)

2.2. Requirements

Requirements have a broad definition. They should precisely define what shall be done by the product or service. Another important aspect is to describe what a certain person, who is involved in the process, should get delivered. (Rupp, 2004, p. 138)
It is always a good idea to describe requirements in the form, what a system should be capable of. The question, how the system is delivering the result, is not part of a typical requirement. (Sommerville and Sawyer, 1997, p. 4)

2.2.1. Types of Requirements

Concerning types of requirements, there are various possibilities to form groups. One of the most used criteria are: (Rupp, 2004, p. 138):

- The field of the requirement
- The level of detail the requirement is formulated
- The priority levels

2.2.2. Fields of Requirements

Talking about different fields of requirements, there are various types which can be distinguished. Some examples are (Rupp, 2004, p. 140)

- Requirements describing functions
- Quality-related requirements
- Technical requirements
- Requirements considering legal rights

2.2.3. Functional Requirements vs. Non-Functional Requirements

As mentioned in the chapter before, a widely used criteria to differentiate between requirements is the distinction between functional and non-functional requirements. Functional requirements describe the system's direct requirements concerning its abilities and the interactions with users. (Rupp, 2004, p. 140)

Non-functional requirements are simply all which are not part of the functional requirements. (Rupp, 2004, p. 140)

Mainly, functional requirements describe pure functionality of a system while non-functional requirements determine boundary conditions or constraints which must be fulfilled while delivering the functional requirement. It is not always a clear boundary between functional and non-functional requirements. There may also be the possibility of splitting a requirement into a functional and non-functional part. (Sommerville and Sawyer, 1997, pp. 7–8)

Quite often, non-functional requirements are represented as quality requirements, constraints relating the implementation process or the system as a physical object. (Partsch, 2010, pp. 27–30)

Partsch (2010, pp. 27–30) also talks about considering the requirements for the project organization as non-functional requirements.

2.2.4. Tasks of Requirements

Requirements are the core of the whole requirements engineering process as they are used to describe the target product or service in a structured way. When looking at the way how these requirements influence the process of system development, primary and secondary tasks for requirements can be figured out. (Rupp, 2004, p. 11)

Primary Tasks of Requirements

Primarily, requirements directly influence the development process of the product/service. The requirements form the base for several important decisions during and after the product/service development (Rupp, 2004, p. 11):

- They are a meaningful tool for communication. The requirements make internal discussions easier as every team member has the same information and exactly knows, what is the content of the discussion.
- They also serve as the external communication material, which is needed to create announcements, as well as parts of the contract, where the exact specification of the product/service is included.
- The two mentioned points beforehand solely describe the usage of the requirements during the primary development process, but another very important issue is the fact that maintenance tasks and possible integrations also must be done based on the described requirements.
- Lastly, requirements have a big effect on the system architecture, as they strictly determine, what the system should be aware of.

Secondary Tasks of Requirements

Secondarily, requirements also affect people and organizations outside the primary development process, e.g. during the planning phase or after the development process. The way the effects take place differs from sector to sector. (Rupp, 2004, p. 12)
Some classical secondary effects of requirements are (Rupp, 2004, p. 12):

- Detecting potential for future rationalization
- Increasing customer/employee satisfaction when directly considering customer/employee wishes.

2.3. Project Structures

As it is defined now what a requirement is and how they can be categorized, the question arises, how they are defined during a project. To answer this question, it must be clear how the project structure itself is organized. There are some basic models which are widely used to describe the main procedure way in a project. (Rupp, 2004, p. 43)

Three categories can be identified (Rupp, 2004, p. 43):

1. Iterative-incremental procedure models
2. Use-case driven procedure models

   As already mentioned in the name, use-cases are used to determine the system development process. Use-case driven procedures are discussed in chapter 2.6.1.
3. Agile procedure models

2.3.1. Iterative-incremental Procedure Models

Iterative-incremental procedure models divide the project work into several incremental parts, which build up on each other. The development itself is organized as temporal iterations. For each following segment (increment), findings from previous steps can be integrated. In combination with the repeating iterative activities the procedure model type is a very widely used one. (Rupp, 2004, p. 43)

Rational Unified Process

The Rational Unified Process – shortly RUP – defines a model of how to develop object-oriented software. Basically, it can be used also in different areas than software engineering, but then it has to be adapted at some points. (Rupp, 2004, pp. 43–46; Kruchten, 2007, pp. 17–33)
RUP is based on six best practices, which are widely used in the software development sector (Kruchten, 2007, pp. 17–33; Rupp, 2004, p. 44):

1. Iterative development of software products
2. Requirements engineering and management
3. Usage of architectures, which are based on components
4. Modeling of software with the help of visual methods
5. Definition and verification of the developed software's quality – throughout the whole development
6. Embedded change management

As it is clearly visible, RUP also defines requirements engineering as a very important step towards successful software development. RUP besides mentions other main parts, as iterative - use-case-driven - design, design-driven development and dynamic changes in design and architecture as important steps for a modern software project. But RUP itself is not just a description of important points, it is a real product and process, which evolves, changes over time and can be directly applied. (Kruchten, 2007, pp. 17–33)

V-Model

The V-Model is an incremental-iterative approach which tries to give a guideline how to structure a project, how to consider user requirements, transform these into technical requirements and document them in an appropriate way. (Allgemeiner Umdruck Nr. 250/1, 1997; Allgemeiner Umdruck Nr. 250/3, 1997; Rupp, 2004, pp. 46–49)
The level of detail increases with each incremental step as shown in Figure 1. The “V” in the name of the model is based on the structure of the temporal process. As already mentioned, the level of detail in combination with the definition and appropriate documentation of user and technical requirements is the first milestone. Each of the previous steps gets mapped to a certain quality testing step. (Allgemeiner Umdruck Nr. 250/1, 1997; Allgemeiner Umdruck Nr. 250/3, 1997; Rupp, 2004, pp. 46–49)

The V-Model offers a basic structure how to define and document systems, which is not solely limited to software, e.g. it is also used in the VDI 2206 by Gausemeier and Moehringer (2002). It is not a fixed, static procedure. A better explanation is a rough check list which tries to offer a framework for avoiding most common mistakes, like missing documents, unnecessary documents, missing quality as a result of a lack of testing. (Allgemeiner Umdruck Nr. 250/1, 1997; Allgemeiner Umdruck Nr. 250/3, 1997; Rupp, 2004, pp. 46–49)

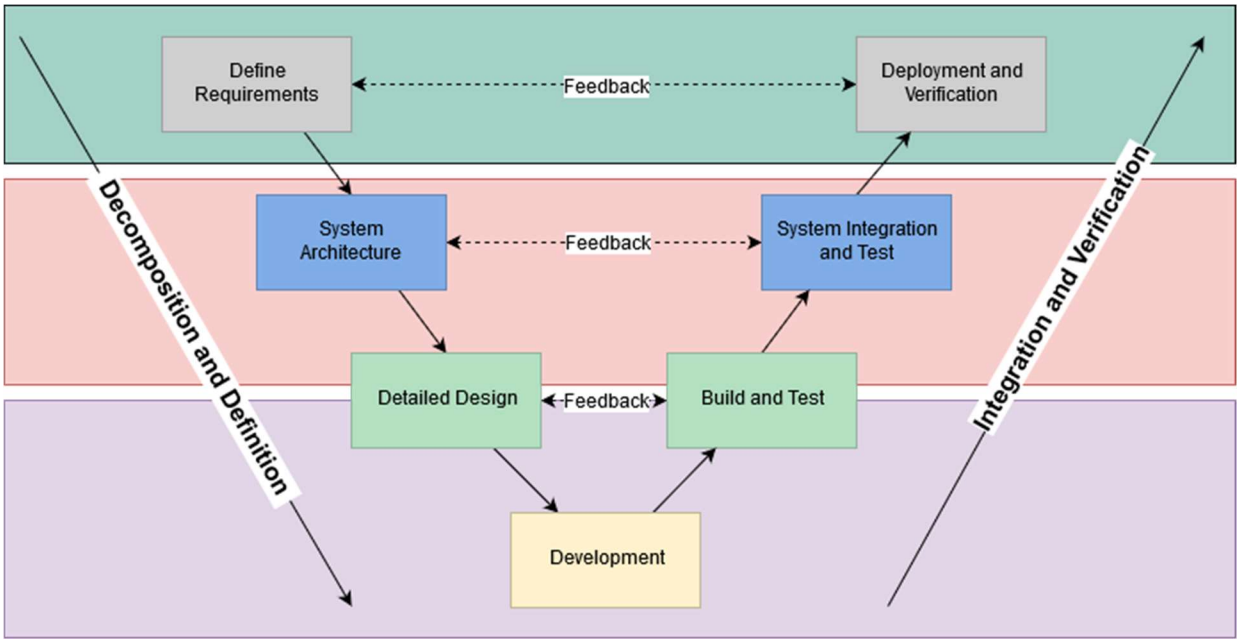

Figure 1: V-Model (Ruparelia, 2010, p. 10)

2.3.2. Agile Procedure Models

There may be some projects for which the detailed, fixed pattern for problem solving is not the most appropriate one. In this case, agile procedure models may be preferable. They use a broad toolset of available methods and focus mainly on four basic patterns (Beck *et al.*, 2001):

- The individuals and their interactions involved in a project are more important than the used tools or patterns.
- The delivery of a working (software) product is more important than the documentation step.
- The collaboration with the project customer is more important than the negotiations with the customer.
- Reacting to changing requirements is more important than strictly following one way.

The core field of agile methods is software development, but basically the agile methods can be executed in nearly all fields of project working.

eXtreme Programming

eXtreme Programming – shortly called XP – has evolved to a very famous agile (software) development procedure. The core thinking behind is producing running software as soon as possible. (Beck, 1999, pp. 7–10)
Various fundamentals have evolved which describe XP best (Beck, 1999, pp. 7–10):

- Working in pairs (four eyes principal), often described as pair programming
- Starting with a simple design and add further functions later
- Refactoring the system to stay flexible
- Minimal effort for the first running system, evolve it in the direction which is most valuable
- Write test cases before evolving the system
- Testing of the system several times a day

Scrum

Scrum is an agile development method, mainly used in the software development branch. It does not require a strict predefined definition of procedures or team structure, it rather defines a rough methodology to enhance the software developed by the most important features very fast. (Schwaber and Beedle, 2002; Haberfellner *et al.*, 2018, pp. 103–104)
Elements, which have to be implemented, are basically prioritized according to their importance. Out of these elements, a certain number of features is selected to be implemented in the next iteration, which is called a sprint. Sprints should have a pre-defined duration, typically below the maximum of a month. (Schwaber and Beedle, 2002; Haberfellner *et al.*, 2018, pp. 103–104)
Scrum uses mainly three different roles (Schwaber and Beedle, 2002; Haberfellner *et al.*, 2018, pp. 103–104):

1. Product Owner

   The product owner is responsible for prioritizing the elements to be implemented, sets the main target of the project and is also responsible for providing the budget.

2. Team

   The team is responsible for implementing the elements from the prioritized list, communicates directly with the product owner and can set limits how many elements are realistic to be met within the next sprint.

3. Scrum Master

   The scrum master is overviewing the whole development process, tries to guide the team through their working tasks, supervises roles and guarantees a smooth project progress.

2.4. System Analysis

System analysis is a main part in requirements engineering as it should reveal the needs of different stakeholders. As it is simply not possible to document every single aspect of a system's requirements in one model, multiple system models are used to express the dynamics of different viewpoints. With the guidance of the developed system models it should be clear for the stakeholders, what the system is aware of and if their requirements are met. These models are well-suited tools for guiding the system analysis task. (Sommerville and Sawyer, 1997, pp. 164–165)
For further supporting the step of system analysis, there are some basic guidelines which should teach, how to best proceed during the analysis step. (Rupp, 2004, p. 53)

The procedure guidelines are not very different to the project procedure models, they should be understood as a more detailed approach to design the step of system analysis. Some system analysis approaches are dependent on how the overall structure of a project looks like, e.g. there may be some approaches which perfectly fit into an agile project architecture or some which fit in other models better. (Rupp, 2004, p. 53)

2.4.1. Object Engineering

Object engineering is an approach which is used to systematically describe a system with its requirements in a way that they can be maintained and managed in the future. (Rupp, 2004, p. 55)
The object engineering approach defines (Rupp, 2004, pp. 55–56):

- The activities which are necessary to fulfill the above-mentioned approach.
- The persons who should be aware of fulfilling the activities.
- The dependencies between certain activities.
- The results which are generated by certain activities – called artefacts.
- The methods for planning, doing and controlling the activities and generating the artefacts.
- Methodologies for describing certain artefacts.

The components of the object engineering approach can be combined and weighted differently, depending on the prerequisites of the project, which highly depend on the project structure itself. (Rupp, 2004, p. 55)

Methods in Object Engineering

Methods are used to give the applying company/person(s) a set of techniques for doing the activities. They vary from investigation, over definition to documentation and quality assurance tools. (Rupp, 2004, p. 56)

Notations in Object Engineering

Notations are a required tool to create a suitable documentation for the created artefacts. The way, artefacts are written down, can differ – depending on the artefact itself. Notations can be e.g. natural language, program code, use-case diagrams, activity diagrams. There is a huge number of different possibilities, the main point is to describe the artefact as detailed and clear as possible. (Rupp, 2004, p. 56)

Artefacts in Object Engineering

Artefacts are defined as a physical result of a prior activity in the object engineering approach. They are used as input for next phases. (Rupp, 2004, p. 55)
In object engineering five main artefacts are defined as (Rupp, 2004, p. 55):

1. Targets, including stakeholder definition and boundary conditions
2. Requirements, written down in natural language
3. Analytical model
4. Conditions for fulfillment of requirements
5. Simulation model

Target Definition

Targets are a certain form of requirements, but on a more generic level as they are defined on a wider base. Right at this high abstraction level it is very important to know the stakeholders of the project and to analyze their dependencies and influences. Otherwise, it will be very hard to deliver an acceptable product. (Rupp, 2004, p. 57)
Identifying the stakeholders of a system, talking with them, examining their needs and considering them while implementing will not only make the system a better one, but also underline the stakeholders' importance. This will in fact lead to more satisfied customers, improving the system usage and acceptance in the end. Additionally, it will help the system engineers to get a better feeling, how different groups of people see the system and what they desire. (Sommerville and Sawyer, 1997, pp. 72–74)
A fact, which is often neglected in this early time of a project, are the boundary conditions restricting the project as a whole. Very likely, the main target(s) and the boundary conditions also influence the choice of the project procedure model. (Rupp, 2004, p. 57)

Requirements

Requirements are the central artefact in the object engineering approach. Typically, they are defined in natural language as it is simply understood and accepted broadly. The downside of natural language is a sometimes imprecise and incomplete understanding. (Rupp, 2004, p. 57)

Analytical Model

In order to get a quick overview of what is newly defined in the requirements, an analytical model should be created. This model should emphasize the dependencies and influences between different parts of the system, usually realized with graphical modeling. A widely used modeling language is UML (Unified Modeling Language), which offers various diagram types, for example class diagrams. In these UML class diagrams, an object-oriented approach is used to generate classes with corresponding attributes and operations. (Rupp, 2004, pp. 58–60)

Requirements should be “translated” into a suitable notation for modeling. Very often, such a translation into the model reveals some problems concerning different requirements or conflicts within one requirement. This possibility is used to further refine the definition of the requirements and update those definitions in the requirements specification document. (Rupp, 2004, pp. 58–60)

Fulfilling Criteria for Requirements

Besides the definition of the requirements, criteria for the fulfillment of each requirement has to be defined. This ensures that the system can be tested against the defined requirements and on the other hand that the quality of the requirements can be ensured. (Rupp, 2004, p. 60)

To enable a precise control mechanism for requirements, it is advised to use quantitative values for fulfillment criteria wherever applicable. This will result in a lower risk for discussion between the customer and the developers during the development phase of the project. (Sommerville and Sawyer, 1997, p. 157)

The fulfillment criteria play an important role during the verification and validation step. The proofing techniques can be simple manual checks with e.g. checklists or more formal techniques, which may also include mathematical proofs or software guided test cases. It is very important to precisely describe requirements and stay consistent in their description when using different words multiple times. (Partsch, 2010, pp. 51–54)

The definition of the fulfillment criteria is done in parallel to the definition of the requirements. This results in being able to see if the requirements are testable and if all corner cases are thought through and covered. The fulfillment criteria are often defined by another person, who has certain knowledge in testing (software) products. This offers another point of view and increases the quality of the requirement itself as unclear facts are identified very early in the development process. (Rupp, 2004, p. 60)

Simulation Model (Prototype)

Dealing with requirements in a written form can be exhausting for certain stakeholders. For this purpose, prototypes are introduced which should model a more or less detailed part of the final system to show certain functionality. (Rupp, 2004, p. 61)
Depending on the needs of the current project stage, prototypes can come in many different forms: paper drawn prototypes which should give also some graphical explanation, programmed tools which model just a single functionality or even bigger prototypes which include a broad number of functionalities. The purpose of prototypes is essential: some are made to be thrown away, some are made to be integrated later on in the final product. (Rupp, 2004, p. 61)
No matter which type of prototype is used, they always help the stakeholders to feel like really using the final system which introduces a completely new sight on the requirements. In the end, quality of the requirements will increase, some requirements will be thrown away and new requirements will arise. (Rupp, 2004, p. 61)

Usage of the Object Engineering Approach

The way the object engineering approach is used highly depends on the project structure itself. It is possible to use just a subset of the explained artefacts or combine them in different time steps. Investing time in (at least for a certain project) useless artefacts is not desired. (Rupp, 2004, pp. 63–77)
The decision, which exact procedure model with all its artefacts will be the right one, is not solely depending on the project structure, there are various factors – called project boundary conditions – which influence the way the final product will be implemented. (Rupp, 2004, pp. 63–77)
Basically, three main areas of influencing factors can be distinguished (Rupp, 2004, p. 68):

1. Human factors
2. Organizational factors
3. Content and complexity of the system area

Human factors are in practice one of the most important influences during a project. Common influencing factors include (Rupp, 2004, p. 68):

- Communication
- Knowledge level of stakeholders
- Knowledge level of developers/analysts
- Stakeholder homogeneity
- Culture

The other two influencing factors focus more on the system which has to be developed or extended and how complex the environment as well as the system's content is. (Rupp, 2004, pp. 63–77)

2.5. From System Analysis to a Real Requirement

It is a tough way from the target of a system, over the requirements of the stakeholders to a valid representation of all requirements. In this chapter common influences, which make system analysis a very hard job, will be discussed.

2.5.1. Continuously Increasing Quality of Requirements

Starting point is the system analysis task where one or more system analysts communicate intensively with the stakeholders to describe the desired features. The required information has to be provided by the stakeholders, the given information must be documented in a way that it represents the given facts. This is where first problems occur: Stakeholders know implicitly what they want, but there may be situations where they simply (Rupp, 2004, pp. 197–238):

- expect knowledge which is clear for them, but which is the other way around not clear for an outstanding person.
- cannot express precisely what the desired requirement is about. This is also caused by linguistic problems as the natural language is often not as precise as required.
- change or leave out details which are important.

The three main areas of linguistic effects, considered by Rupp (2004, p. 201), include deletion (facing selected aspects while leaving out others), generalization (considering examples as representative for a whole category) and distortion (changing memory details).

The main purpose of a systematic requirements description approach is to overcome these potential problems, gather the knowledge needed from different stakeholders, understand what they are really about (deep understanding of domain) and then document them in a way that the problems above do not occur for the next person reading the requirements. (Rupp, 2004, pp. 197–238)

This step also heavily depends on the accurate language usage. It is not recommended using complex sentence buildings with e.g. conditional dependencies. The better way is to make simple and straight forward sentences, which do not make a discussion necessary. This further leads to reduced costs as requirements do not need to be refactored and more people, especially stakeholders, understand the key message of the requirements. Better requirements with a higher stakeholder satisfaction will be the result. (Sommerville and Sawyer, 1997, p. 147)
There are very good guidelines in Rupp (2004) and Sommerville and Sawyer (1997) which show basic rules that should be followed to achieve a widely accepted, incremental, qualitative better solution.

In International Organization for Standardization *et al.* (2018, p. 12), additional language criteria for requirements are mentioned, which cover the need of requirement validation. To ensure that requirements can be proven fulfilled or not, unprecise language should be avoided. This means excluding text-fragments, such as (International Organization for Standardization *et al.*, 2018, p. 12):

- Subjective interpretable words
- Superlatives
- Vague comparisons (e.g. “better than”)

2.5.2. Systematic Approach for Good Requirements

The previous chapter focused on how to increase quality of existing requirements concerning mostly linguistic features. This chapter focuses on how to construct qualitative requirements with a pre-defined template. The knowledge of the previous chapter can also be applied afterwards to further increase requirements quality. (Rupp, 2004, pp. 239–269)
The template consists of six main steps (Rupp, 2004, pp. 239–269):

1. Defining the process which is the baseline of the requirement
2. Defining how the system does a certain activity
3. Defining the strictness of the requirement
4. Extend the requirements with needed objects and explanations
5. Define conditions under which circumstances the requirement should be fulfilled
6. Apply the approach of the previous chapter for increasing quality

Other resources, like International Organization for Standardization *et al.* (2018, p. 10), mention additional construct elements for good requirements, e.g. the fact that a requirement can be limited by a specific number of constraints. In International Organization for Standardization *et al.* (2018, p. 10) it is also highlighted that a requirement clearly determines the system's capabilities, when used by a pre-defined user, but the requirement does not tell anything about the user's skills.

Defining the Process

The center of each requirement is the raw functionality which should be provided. Further on, this is called the process which should be mainly described with a verb. (Rupp, 2004, pp. 242–243)

Defining the Characteristics of a Certain Activity

After defining the process, the question, how the system executes the certain process, arises. Three main types can be distinguished (Rupp, 2004, pp. 243–245):

1. The system executes the process totally on its own.
2. The system offers the functionality to the user.

   Whom should be given the possibility to do something?
3. The system executes the process in dependence of another system.

   What should the system be aware of? The requirement of the other system should not be part of this requirement as it is an encapsulated system.

Defining the Strictness of the Requirement

Depending on the importance of a certain requirement, it is necessary to express it with the corresponding linguistic baseline. There is a big difference between a system which

- shall
- should
- will

do something. It is very important to highlight must haves with the corresponding strict formulation. (Rupp, 2004, pp. 239–269)
Sommerville and Sawyer (1997, p. 149) also suggest using the above-mentioned words whereas "shall" expresses a must have requirement, "should" shows a nice-to-have requirement and "will" states external provision.

Extending the Requirements

Until this step, the raw functionality has been described. To give a more detailed explanation and insight, further objects and explanations can be added to clarify what is meant by the requirement. (Rupp, 2004, pp. 246–247)

Defining Conditions

In software development it is usual that things should be done under certain circumstances. For this reason, conditions are also embedded into the definition of requirements to express what must be fulfilled that a certain activity is executed. There may not only be logical conditions, temporal conditions are also another form which can be very useful. (Rupp, 2004, pp. 239–269)
As already mentioned earlier, it can be confusing when using conditions in natural language. Complex conditional aspects are better either divided into smaller, clear subparts or are visualized appropriately. (Sommerville and Sawyer, 1997, pp. 147–156)

Increasing the Quality of the Requirement

A basic template has been modeled for creating systematic requirements. Now the quality can be increased, whereas semantic unclearness should be erased. This is done using the methods described in the previous chapter. (Rupp, 2004, pp. 249–250)

Development of Semantic and Logical Tables

Besides using the template steps for creating meaningful requirements, definition tables are widely used in practice. Mainly, two different tables are created (Rupp, 2004, pp. 251–257):

1. Semantic tables in which used process words (the core of the requirement) are described precisely in the way what they mean, what type of system activity they express and how they can be used.
2. Logical tables which should translate classical logical operators (OR, AND, XOR, etc.) into a verbal representation to give an overview what a certain linguistic condition means.

2.6. Documentation of Requirements

In the previous chapters we focused on what a requirement is and how it can be described best using natural language, but as already mentioned, the natural language is not the only method for documenting requirements. (Rupp, 2004, pp. 156–157)
In some cases, it is not the best as other methods, like e.g. a graphical representation, mathematical functions or even program code can express the requirement in a more precise way. Engineers very likely already know many notations, which reduces the costs for introducing another way of requirements representation. (Sommerville and Sawyer, 1997, pp. 154–156)
The documentation methods can be explained in four different areas (Rupp, 2004, p. 157):

1. Behavior-oriented
2. Data-oriented
3. Context-oriented
4. Others

There are various techniques for each group of documentation techniques which are used in practice. This chapter should give a quick overview of the most important ones.

2.6.1. Context-oriented Documentation Techniques

Use-cases

A very famous representor of the context-oriented documentation techniques are the use-cases. They are used to describe the system from an actor's (user/stakeholder) view. They should show how the environment interacts with the system and what the result of an action is. (Rupp, 2004, pp. 161–166)
On the one hand, use-cases can be described semi-formally (organized, but with natural language) where each process (use-case) is described in terms of the actors interacting with the system, the preconditions, the postconditions and the main points which are done during the process. Table 1 shows an example. (Rupp, 2004, pp. 161–166)

Table 1: Example Use-Case of a Simple Login (Semi-Formal)

Use-Case Name	Log-In
Short Description	The user logs in into the existing system providing her/his username and her/his password.
Actors	User, System
Trigger	User clicks on login-button
Precondition	The user has already registered on the systems platform and knows her/his corresponding login data. The login page is already loaded.
Result	The user is successfully logged in, the user area is loaded.
Postcondition	The system has approved the identity of the user and is ready to start personal functions.
Main action plan	1. The user puts in her/his username in the username field. 2. The user inserts her/his password in the password field. 3. The user clicks on the login-button. 4. The system proofs the inserted username and password combination. 5. The system automatically loads the page for the user's personal area.

On the other hand, there is the possibility to utilize use-case-diagrams, where each use-case, pre-defined with the shown use-case specification, is represented, associated with actors (environment) and other use-cases (inter-system) to obtain a contextually view on the system and its environment. It is very important that the use-case diagram includes the important requirements in form of use-cases as it is shown in Figure 2. But with the increasing number of use-cases, the diagram can get unstructured and chaotic which should be avoided. Stakeholders should gain a quick overview over the system's process and interactions with its environment. (Rupp, 2004, pp. 161–166)

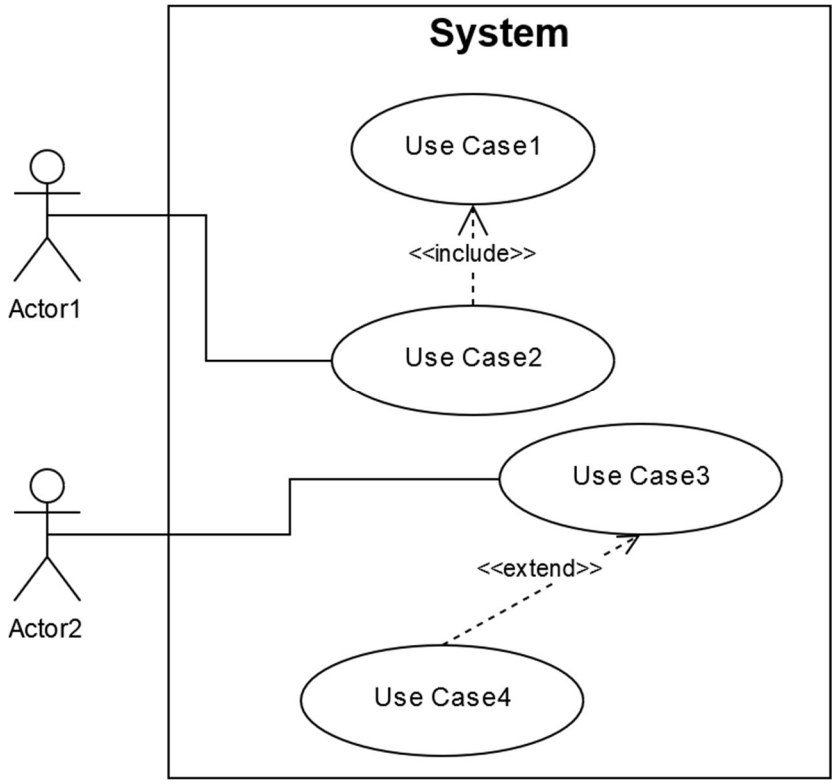

Figure 2: Example Use-Case-Diagram (based on (Rupp, 2004, p. 164)

2.6.2. Behavior-oriented Documentation Techniques

As described in Object Management Group (2017), the UML2 specification offers also possibilities to model activities to show systems behavior during a process. Basic core elements are (Rupp, 2004, p. 166; Object Management Group, 2017):

- Activities
- Actions taken by the system
- Object Nodes
- Control Nodes for expressing conditions
- Edges for connection

With these basic elements a system's procedure of a certain use-case can be described in terms of when does the system what under which circumstances, all graphically displayed.

2.6.3. Data-oriented Documentation Techniques

UML2 also defines in Object Management Group (2017) a possibility to arrange the system's components in a class diagram. It is an object-oriented approach which has the basic elements (Rupp, 2004, p. 186; Object Management Group, 2017):

- Classes
- Attributes which belong to certain classes
- Operations
- Associations

Classes define formal described structures of certain things in the system. All the instances of a class have the same structure which is defined in the class itself. Classes can contain attributes, which are fields for stored information of a certain instance of a class. Operations are used to show what a certain part of the system can do. Associations are used to express relationships between classes. Like in the object-oriented programming approach, looking at a class diagram the stakeholders can easily identify different parts of the system, the different abilities of the parts and the relationships between them in one single diagram. Needless to say, this is a data-centered approach as it solely describes what data is provided and saved for certain parts of the system. (Rupp, 2004, p. 186; Object Management Group, 2017)

2.6.4. Others

The other documentation techniques - which cannot be further specified - include for example the natural language description which was covered in detail in previous sections. (Rupp, 2004, p. 157)

2.7. Dealing with Non-functional Requirements

Non-functional requirements are a special part in requirements engineering as they represent one of the least approved and often neglected parts during the whole process. The previous sections described in detail how to form requirements and maintain or document them. Dealing with non-functional requirements, it is often a lack of structured possibilities or attitudes of under-prioritizing which leads to a bad description of those non-functional parts. (Rupp, 2004, pp. 269–288)

The value of non-functional requirements is often under-estimated. They can lead to (Rupp, 2004, pp. 270–271):

- Higher customer satisfaction

  Customer often focus on quality criteria more than on some small missing functions.

- Increased productivity
- A complete picture of the whole system, often leading to new functional and non-functional requirements

2.7.1. Problems Estimating Non-functional Requirements

As already mentioned, non-functional requirements are often forgotten in their importance as the structured description of direct functionalities seems to be the most important. In addition to this, the fact that non-functional requirements are influenced and demanded by different groups of stakeholders – not only the real system users – leads to a confused, helpless attitude. (Rupp, 2004, pp. 269–288; Partsch, 2010, pp. 27–30)
Even for stakeholders, their non-functional requirements may not be clear in the first sense. For this purpose, it can be very helpful to work with prototypes which help them realizing what is necessary. Another helping guide for stakeholders could be showing them examples of other non-functional requirements to give them a quick inspiration based on what others came up with. (Rupp, 2004, pp. 269–288)

2.7.2. Formulation of Non-functional Requirements

Non-functional requirements are a special task for formulation as they often cannot be visualized good enough. Textual representations are quite common for non-functional requirements. (Partsch, 2010, p. 71)
The formulation of non-functional requirements has some specific important issues apart from the classic description of requirements, which already have been discussed (Rupp, 2004, pp. 275–276):

- Writing down non-functional requirements, it is essential to give them some quality criteria which can be tested. It has to be absolutely clear, to which extend a non-functional requirement has been fulfilled afterwards.
- UML also offers possibilities for modeling non-functional requirements.
- Non-functional requirements should be referenced to functional requirements whenever possible.

3. Stakeholder Centered Engineering

The previous chapter covered in detail how requirements of an evolving system can be formulized and documented. In this chapter, the focus will be shifted to the needs of different stakeholders during a (software) project and how their requirements can be organized.
Stakeholders of a certain project can be understood as the group of individuals who influence the progress and result of the project or who are affected by the process or the outcome of a certain project. (Freeman, 2010)

3.1. Stakeholder Analysis

During a stakeholder-centered engineering process the stakeholders should be the main factor to think about. In later chapters tools, which help to model the stakeholder interrelations, will be introduced. The beginning should be a stakeholder analysis to get an overview of the certain stakeholders and how they can be described. After answering these questions, the interrelations between them in the big ecosystem of an organization and its environment can be modelled.

3.1.1. Basic Stakeholder Analysis Technique

The basic stakeholder analysis technique is widely used to identify stakeholders and to summarize their interests in terms of one's own organization. This can lead to first strategic plans concerning the stakeholders. Main steps involve (Bryson, 1995, pp. 71–75, 2003, pp. 13–14):

- Brainstorming of potential stakeholder candidates.
- Preparing an overview sheet for each stakeholder. For each one, writing down:
    - What the stakeholder desires from the organization
    - Judge how well the organization is doing now to fulfill the stakeholder's needs
    - Identify what can be improved to satisfy the stakeholder's needs
    - Identify how the stakeholder effects the organization
    - Write down what the organization may desire from the stakeholder
- Possibly rank the stakeholders according to their power.

3.1.2. Power Versus Interest Grids

The power versus interest grid was described in Eden and Ackermann (2004) and helps to organize the stakeholders in a matrix. The two dimensions, which define the matrix, are the interest of the stakeholders in the organization and the power of the stakeholders affecting the organization. The resulting matrix can be categorized into four different fields, each standing for a specific combination concerning interest and power. The result helps the organizations to get a feeling on whom to focus on, where to pay attention and to identify the most important players. (Eden and Ackermann, 2004, 121-125 344-346; Bryson, 2003, pp. 14–15)

3.1.3. Stakeholder Influence Diagrams

Stakeholder influence diagrams are used after completing the power versus interest grid. Within this method the goal is to draw influencing lines between the placed stakeholders in the grid. The process is used to be carried out in a group with various adjustments of the influencing lines. Basically, also a two way influence is allowed, but the main direction of the bigger influence should be clearly marked. (Eden and Ackermann, 2004, pp. 349–350; Bryson, 2003, p. 15)

3.2. Service Design Thinking

Service design thinking, described in Stickdorn and Schneider (2013) and Stickdorn *et al.* (2018), is a discipline which tries to enable service engineering. The stakeholders and their needs are put in the center of all considerations. Service design thinking focuses on understanding stakeholder requirements, leading to a unique user experience.

3.2.1. Core Principles

Service design thinking is based on, according to Stickdorn and Schneider (2013, p. 34), five core principles, named user-centered, co-creative, sequencing, evidencing and holistic.

Since 2010, these values have been slightly adapted which leads, according to Stickdorn *et al.* (2018, p. 28), to the following core principles, named human-centered, collaborative, iterative, sequential, real and holistic.

There are other sources which investigate the base of service design. Luo (2011) sees service design as a process, where people interact with service touchpoints. There are some different emphasizes between Luo (2011) and Stickdorn *et al.* (2018), but the core idea behind service design is the same. It is all about people, namely stakeholders.

User-centered/Human-centered

The core of a successful service creation is the deep understanding of the later users. Service design thinking requires more than simple statistical analysis, it is about seeing the service with the user's eyes. This is necessary as e.g. two statistic identical persons (same age, same social status) can have totally different opinions and attitudes. It is essential to know the users and their background intentions. (Stickdorn and Schneider, 2013, pp. 36–37)
Giving the users alternatives to make the service more flexible, is a widely used approach for involving different user groups in the service process. (Luo, 2011, p. 52)
In the updated version the understanding of the user is replaced with the understanding of all human beings involved in the design process which expresses the need for stakeholder involvement. (Stickdorn *et al.*, 2018, p. 28)

Co-creative/Collaborative

The service engineering process is a co-creative process, where different people have to establish a common understanding of how to fulfill the customers' needs. In this process it is advised to integrate not only customers, but all stakeholders, which are influenced by the future service. Real service design thinking includes all stakeholders and tries to deal with the problem of creativity transportation from one human being to others. (Stickdorn and Schneider, 2013, pp. 38–39)
The new collaborative core principle totally agrees on the previous one, but tries to emphasize the involvement of all stakeholders with different backgrounds in the service design process. (Stickdorn *et al.*, 2018, p. 28)

Iterative

The updated principles contain also the fact that the whole service design process should be executed fast and iterative. The final goal is the introduction of the service. (Stickdorn *et al.*, 2018, p. 28)

Sequencing/Sequential

Services can be seen as different, sequencing static pictures, which form a movie in the end. To design a successful service, all pictures have to be well thought through, they should even be prototyped like it is known in the product innovation process. Mainly, the service can be divided into three main periods (Stickdorn and Schneider, 2013, pp. 40–41):

1. The pre-service stage
2. The service stage
3. The post-service stage

All three stages have to be dealt with to give the customers an optimal service solution and meeting their needs. (Stickdorn and Schneider, 2013, pp. 40–41)
The whole service process is one of the core aspects for the service process innovation, where designers try to optimize the lived key elements for the customers. Thoughts, like when and where to pay during the service, are important examples during the service process innovation. (Luo, 2011, p. 53)

Evidencing/Real

Services often deliver their true value in moments where the customer is not really aware of consuming it. But when paying for the service, the customer should be sure why she/he is paying the bill. A widely used approach is to make the customer remember the positive effect of the service with some kind of memory: This can be an accurately designed bill, a small present or even a customer-oriented e-mail. If it is considered the right way, customer satisfaction and willingness to pay can be increased. (Stickdorn and Schneider, 2013, pp. 42–43)

The updated version of the core principles tries to emphasize the fact that all stages of the service design should be made real: From the research phase, over the prototyping phase until the values delivered by the service as mentioned above. (Stickdorn *et al.*, 2018, p. 28)

Holistic

Service design thinking should be done in a holistic way. It is very important to consider the environment where the service takes place (and what a certain customer might be sensing in this moment). Different service sequences with customers should be taken into account to get a feeling for the service in its macro environment. (Stickdorn and Schneider, 2013, pp. 44–45)

3.2.2. Procedure

The complex method of designing a service with all its core principles, its holistic viewpoint, can be described with a simple mindset which should be followed during the whole designing process. There may be resources suggesting some more steps, but the basic idea behind them should be the same. One of the most important facts about the procedure model is that it is organized iteratively which means that from each step during the process there may be one or more steps to be taken back to optimize the result. (Stickdorn and Schneider, 2013, pp. 124–127)

The four suggested main steps by Stickdorn and Schneider (2013, pp. 122–123) are exploration, creation, reflection and implementation.

Exploration

The first task is about designing the big picture. Service design is all about the customer. For the first step it is important to get a feeling for the companies' attitudes and if they know what service design is about. This ensures the overall goal and enables effective co-creativeness. (Stickdorn and Schneider, 2013, pp. 128–129)

The exploration task further references the field of system analysis and design, as every service is in fact a system consisting of several elements interacting with each other. The methods and philosophy behind systems engineering will be discussed in chapter 4. (Luo, 2011, pp. 52–53)

Afterwards, the service designers have to figure out where to start, in fact this is the definition of the problem itself. In this case there are various tools guiding the design team visualizing attitudes or mindsets of involved people to help understanding the big picture as well as describing some detailed viewpoints. The visualization of the findings is the final approach for this step. (Stickdorn and Schneider, 2013, pp. 128–129)

Creation and Reflection

This is the generating step of the whole process which is directly mapped to the next step of reflection. One of the basic thoughts of service design thinking is to find out many possible mistakes as early as possible. The two steps are mainly about doing progress, receiving feedback and iteratively do this again if needed. The main task of gaining response is how to test parts of the service as it is simply not tangible. Therefore, comic-strips, mock-ups or even stage are used for generating the emotional base for customers to give real feedback. (Stickdorn and Schneider, 2013, pp. 130–133)

Implementation

When all obvious problems are cleared and the service is ready to be activated, the implementation phase is reached. It is also mainly about how to introduce change, how to guide these changes and how to control them. Change management is the topic which deals with the whole procedure of how to do change best. One thing, which is often forgotten, is the fact that not only the customers have to be integrated into the change process, but also the employees which should be able to live and transport the ideas behind the service. (Stickdorn and Schneider, 2013, pp. 134–135; Cameron and Green, 2009)

3.2.3. Tools of Service Design

Tools of service design are prepared, systematic approaches for guiding the service-designer to model the real-world observations in a pre-defined way. (Stickdorn *et al.*, 2018, p. 37)

Research Data

Research data is probably one of the most important tools in service design as heavily influences the later process of finding real insights. Data is basically collected as raw- or interpreted-data. Raw data is collected without any obvious relations while in interpreted data, the researcher or data scientist tries to find patterns which should give detailed insights into the topic. (Stickdorn *et al.*, 2018, pp. 38–39)

Personas

Personas are used to describe certain stakeholder groups in a clear format, Figure 3 shows an example persona. There can be various forms, but mainly a persona is a kind of a one pager offering an insight towards the attitudes and interests of a certain stakeholder group. It should be mentioned that personas do not necessarily be equal to the marketing target groups, especially in service design, these two differentiations are not overlapping. Different needs at specific service levels define the personas. It is very important that personas offer a good understanding of the main needs and attitudes of a certain stakeholder group. For this reason, the following core elements are used (Stickdorn *et al.*, 2018, pp. 41–43):

- Image of a typical stakeholder in the certain group
- A representative name of the persona
- Demographic information (this should be chosen wisely to prevent stereotypes or misleading assumptions)
- Quotes, expressing the stakeholder's attitude in one sentence
- Descriptions of the interests, skills, attitudes, goals and so on
- Visualized statistics to give a quick overview of the most important quantitative data

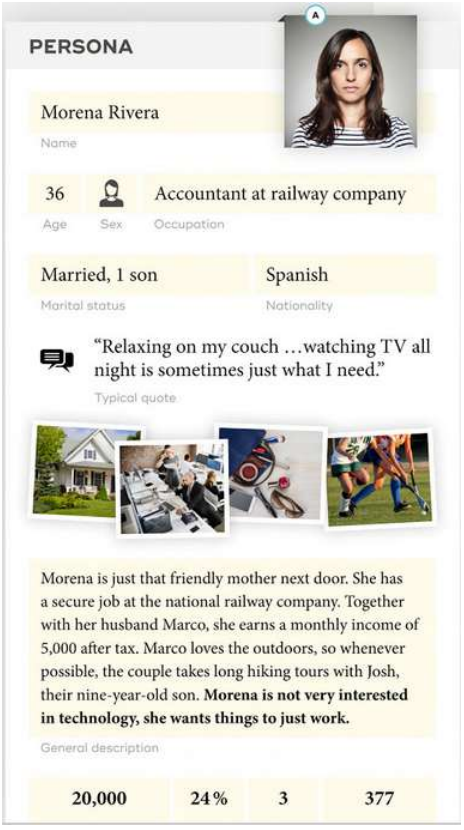

Figure 3: Example Persona (Stickdorn et al., 2018, p. 42)

Journey maps

Journey maps are used to visualize all key experiences during the service lifecycle, beginning from the basic need, over using the service until the post-service phase with certain customer support activities. (Stickdorn *et al.*, 2018, pp. 44–49)
Generally, journey maps are organized as timelines, which should lead through the whole journey of a certain customer. At each step, various types of information are added to the timeline (Stickdorn *et al.*, 2018, pp. 44–49):

- A short text based description of the phase (point B in Figure 4)
- Storyboards, which should graphically show, what the phase is about (comic style, point D in Figure 4)
- Emotional graphs, showing the emotional feeling at each step (point E in Figure 4)
- Communication channels used at each step (point F in Figure 4
- A list of stakeholders involved or responsible for the certain step
- The dramatic arc curve, showing the grade of engagement mapped to a scale from, e.g. one to five (point H in Figure 4)
- Backstage processes, which are often visualized as flow charts
- What if scenarios, about what could go wrong.

Journey maps can be made at different levels of detail for the same service. It is very common to draw a journey map for the whole service and then also make some journey maps for specific steps of the whole service in detail. (Stickdorn *et al.*, 2018, pp. 44–49)

Figure 4: Example Journey Map (Stickdorn et al., 2018, p. 45)

Service Blueprints

Service blueprints are designed to be an extension to the user journeys. They build up a timeline structure, which goes into detail about the frontstage and backstage processes getting triggered by user actions. Besides the frontstage and backstage processes, support processes also get integrated into the service blueprint overview. (Stickdorn *et al.*, 2018, pp. 54–55)

System Maps

System maps are used to visualize the system, embedded in its environment. There are various types of different system maps, which try to include different aspects. Classical elements in such a map are for instance (Stickdorn *et al.*, 2018, pp. 58–59):

- Stakeholders
- Channels
- Processes included in the scenario
- Places if important
- Platforms

Additionally to the various aspects, there are three main types of system maps, according to Stickdorn *et al.* (2018, pp. 58–59), named stakeholder maps, value network maps and ecosystem maps. They try to visualize the environment of a service in different levels of detail (Stickdorn *et al.*, 2018, pp. 58–59):

Stakeholder maps are often utilized to represent different scenario states of a certain ecosystem, including possible future states. (Stickdorn *et al.*, 2018, pp. 58–59)

Stakeholder Maps

Stakeholder maps show different stakeholders, embedded in the environment of the service. They are very useful for getting an overview of stakeholders influencing or being influenced by the service. Stakeholder roles and their interrelations are further part of stakeholder maps. It is always a very informative tool as it reveals facts during the generation process, which are often neglected or simply overseen. (Stickdorn *et al.*, 2018, pp. 59–60)
Stakeholders mostly get arranged in sectors by making a distinction between internal and external stakeholders or customers. Other used aggregation criteria are the importance of the stakeholders in the service environment. For completing the stakeholder map, relationships are added for modeling influencing powers. (Stickdorn *et al.*, 2018, pp. 59–60)

Value Network Maps

Value network maps are an extension of classical stakeholder maps, but instead of visualizing pure relationships between stakeholders they try to model the value exchange between them. Value exchanges e.g. can be simple money exchange for service, information or even things like trust. Icons are used to describe the value exchange while arrows determine the direction of the value exchange. (Stickdorn *et al.*, 2018, pp. 61–62)
Value network maps will be a core focus in the practical part of this work in chapter 5.

E3-Value Ontology

The e3-value ontology, in detail described in Gordijn (2002), offers the possibility to visualize value exchanges between different stakeholders. It consists of three different layers, each of them discussing special points of interest (Gordijn, 2002, pp. 46–47):

1. Global actor viewpoint
2. Detailed actor viewpoints
3. Value activity viewpoints

Global actor viewpoint

The global actor viewpoint is used to give a quick overview of the value exchanges between the different actors. Details, especially concerning the actors, can be derived from the detailed actor viewpoints. Figure 5 shows an example of a global actor viewpoint. The main elements of the global actor viewpoint are (Gordijn, 2002, pp. 48–58):

- Actors

  An actor is considered to be an economically independent participant.

- Value objects

  Value objects can be any form of value exchanged between the actors involved in the model. It can be a service, a product or any other form of value for at least one involved actor. In the e3-value ontology, the modeling of the value objects is not focusing on the real instances of values, but rather on the type of value objects. In some cases, it might be interesting to focus on the actual instance of the value object, then the ontology talks about value object instances.

- Value ports

  Value ports are used to interconnect the actors as they allow value object flows between them. A value object flow should model a change of ownership.

- Value exchanges

  Value exchanges are used for modeling the potential changes of ownership between actors concerning the value objects. The number of changes and the actual value object instances are not the point of interest here.

- Value interfaces

  Value interfaces are modeled to show the actor's willingness to provide some sort of value in expectation of a compensation. One actor can have one or more value interfaces, but usually one interface contains one in-going and one out-going value offering. The basic form of a value interface consists of exactly one offering. If one value port is occupied in a value interface, then all value ports have to do so.

- Value offerings

  Value offerings are a concept to model the willingness for an actor to offer something for her/his environment or to expect some kind of value from it. Value offerings are not explicitly visualized, they can be filtered when looking at all in-going or out-going value ports of a value interface.

- Value transactions

  Value transactions are used to union different value exchanges. They must occur together to fulfill the need of desired in-going and out-going value exchanges. Value transactions are visualized with lines intersecting the value exchanges they contain. The value exchanges are marked with points.

- Market segments

  Market segments are used to group actors which have a mostly common viewpoint on how to value different objects. Not all actors are modelled individually, but rather implicitly, which means actors are summed up concerning their equal viewpoints. If there is an actor, who has a completely different viewpoint on values than the others, there is the possibility to model her/him individually. Market segments are used to group actors, but mainly their value interfaces are joined. The visualization shows a stack of rectangles with the common value interface.

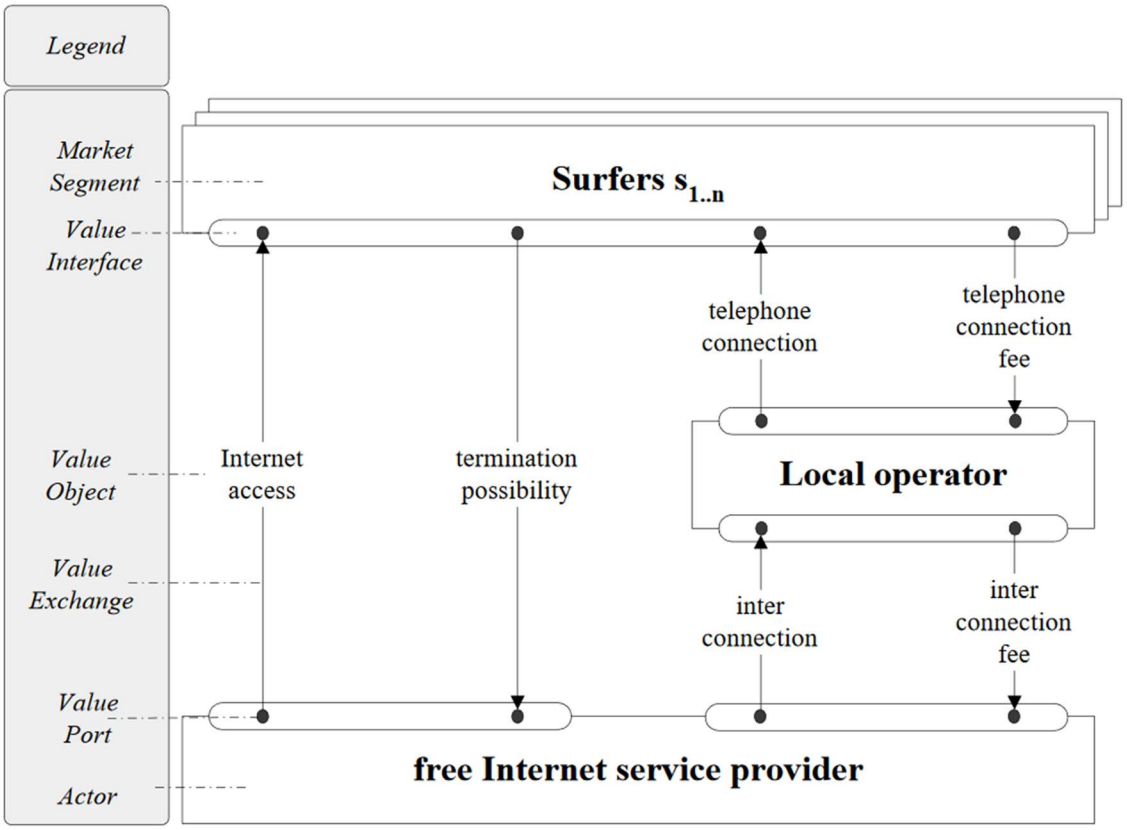

Figure 5: Example Global Actor Viewpoint (Gordijn, 2002, p. 49)

Detailed Actor Viewpoint

The detailed actor viewpoint, shown with an example in Figure 6, is used to give more detailed insights into the actors grouped in the global actor viewpoint. These actors are often united for terms of simplicity. Such types of actors are named value constellations. (Gordijn, 2002, pp. 58–62)

Value constellations are used to show commonly shared attitudes towards value interfaces. Therefore, value interfaces are grouped to express membership of different actors. In the detailed actor viewpoint these constellations are modelled in detail. (Gordijn, 2002, pp. 58–62)

Another reason for the detailed actor viewpoint is to represent partnerships of different actors. (Gordijn, 2002, pp. 58–62)

Composite Actors

In the detailed actor viewpoint, the role of the actors is split up into two different types. The first one is the composite actor who groups value interfaces together with other actors. It is used to express the above-mentioned reasons for the viewpoint itself, mainly for expressing unity of different actors. It is very important to highlight again that value interfaces are grouped, not actors. Actors may offer different value objects in addition to certain partnerships. Finally, a composite actor has its own value interface to connect to the environment. (Gordijn, 2002, pp. 58–60)

Elementary Actors

Elementary actors do not take value interfaces from other actors and are therefore the finest representative unit of the detailed actor viewpoint. (Gordijn, 2002, pp. 60–62)

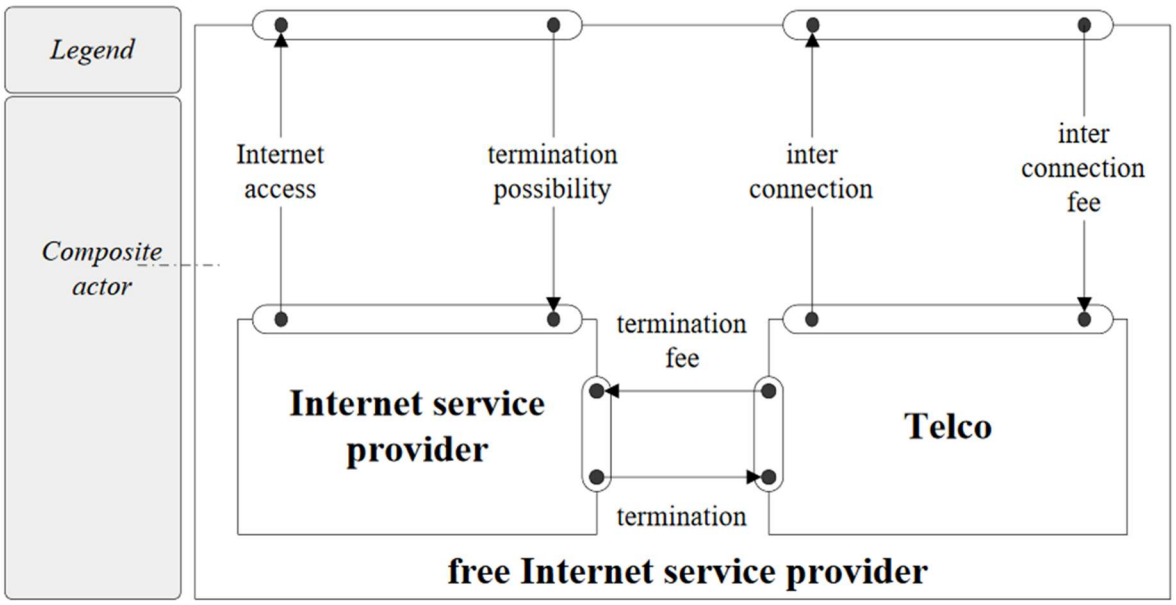

Figure 6: Example Detailed Actor Viewpoint (Gordijn, 2002, p. 60)

Value Activity Viewpoint

The value activity viewpoint is used to link actors with value activities, thus describing how the actors are doing their value creation, as the value activities should lead to profit. Each value activity can have one or more value interfaces, but a value interface can have zero or one value activity. The task of figuring out the value activities and assigning them to actors is often a very hard task during the generation of the model. An example value activity viewpoint is shown in Figure 6. (Gordijn, 2002, pp. 62–64)

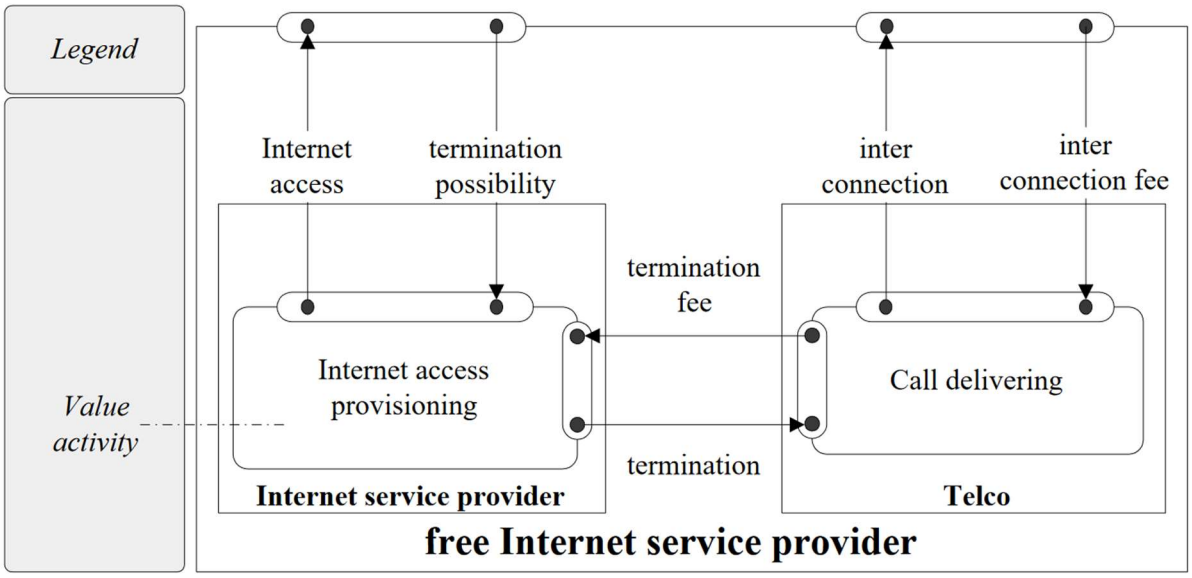

Figure 7: Example Value Activity Viewpoint (Gordijn, 2002, p. 63)

Ecosystem maps

Ecosystem maps are another extension of stakeholder maps or value network maps. To model the real world as detailed as possible, they also consider machines, interfaces and their communication besides the human interactions, even if the communication takes place between two machines. As the ecosystem map can get confusing very fast, a well-defined level of scope has to be preselected. (Stickdorn *et al.*, 2018, pp. 62–63)

4. Systems Engineering

Systems engineering is an approach, which tries to define methodologies for solving problems in different areas. The main assumption is that there is a current situation and a future situation, where the problem is solved, as it is shown in Figure 8. Systems engineering tries to deliver a methodology for supervising the solution finding and implementation process. (Haberfellner *et al.*, 2018, p. 9)

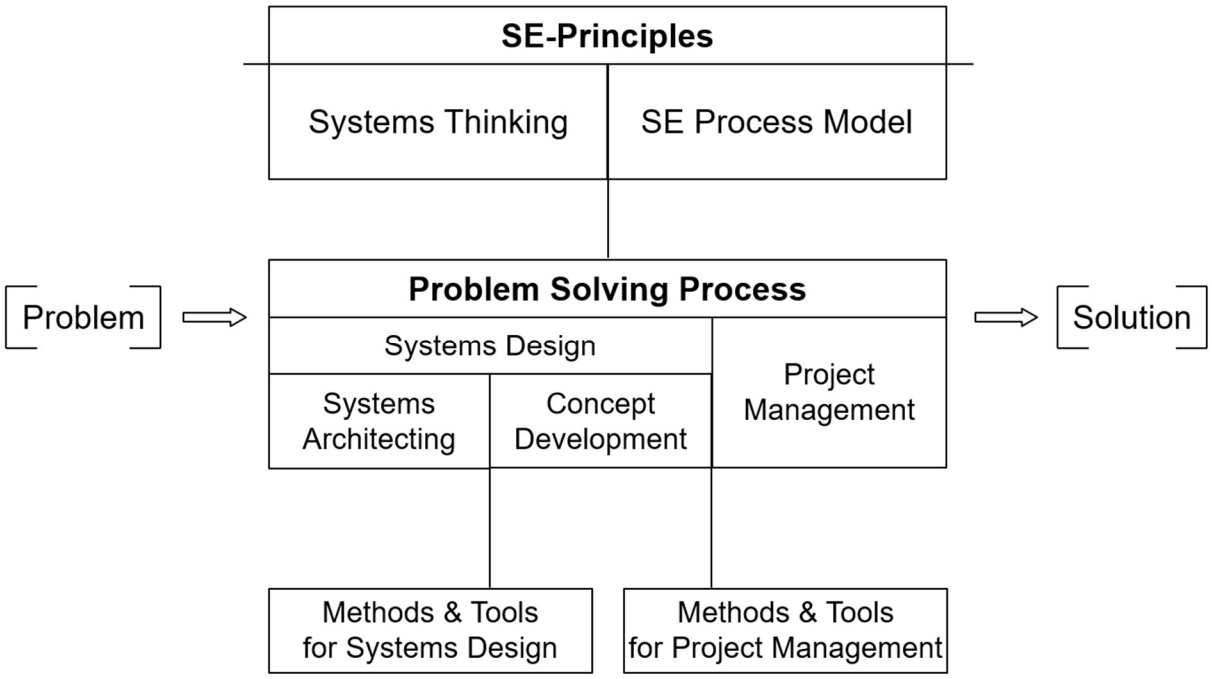

Figure 8: Systems Engineering Explained, Based On (Haberfellner et al., 2018, p. 10)

4.1. Systems Engineering Philosophy

The systems engineering philosophy is based on two main aspects (Haberfellner *et al.*, 2018, pp. 10–11):

1. System thinking
2. Procedure model

4.1.1. System Thinking

The system thinking describes a common understanding, how to describe systems, their elements as well as their environment. It should support modeling complex system appearances to show the big picture. (Haberfellner *et al.*, 2018, pp. 27–35)

System thinking furthermore offers definitions, how to describe basic elements, their relations as well as their environment. The most important definitions include (Haberfellner *et al.*, 2018, pp. 27–35):

- Basic definitions of what a system is, how it can be seen as a composition of different elements and their interrelations. Elements can also be further described in detail with their attributes and functions.
- System boundaries, especially where to see the borders of a certain system
- System structures
- Subsystems and main systems, which should allow describing smaller systems being part of another bigger system, but embedded as a standalone element. Systems engineering often talks about “system of systems”. Basically, this can be seen as a composition of main and sub systems. The main usage of introducing another definition here is to emphasize the fact the subsystem can be also seen as an independent system, which delivers its value or part of it even without the help of the main system. Classical subsystems depend on their main system and are just responsible for a part of the main purpose.
- System hierarchy
  Looking at the prior definitions of elements, subsystems, main systems and system of systems, the complex system itself can be described using a hierarchical order.
- Blackbox-thinking, offering the possibility to see a system just as a function which has an input and delivers a certain output. The step in-between in this approach is not important. The opposite thinking approach is called whitebox-thinking, which tries to look in detail at the way the output is generated according to the input. Typically, the procedure from the blackbox to the whitebox approach is used in top-down analysis methods.
- Differentiation between simple, complicated and complex systems, where the number of elements and the dynamic character play important roles. Figure 9 shows the characteristics of complex and complicated systems.

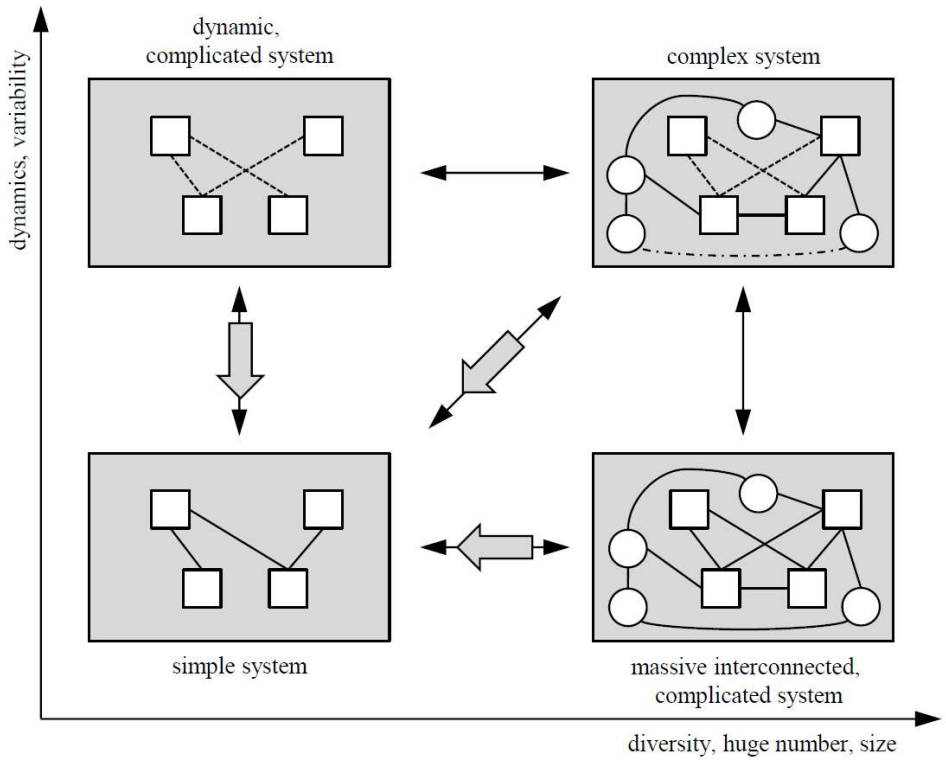

Figure 9: Complicated and Complex Systems, Based On (Haberfellner et al., 2018, p. 34)

The systems of systems approach influences other engineering disciplines as the boundaries between different types of systems get blurry. Through the complex, networked design of large scale systems, isolated components cannot be viewed as standalone parts anymore. This heavily influences certain assumptions, e.g. in requirements engineering, where systems are mostly planned concerning their desired internal functionalities. Integrated in another bigger system, this often gives totally new insights and changes the requirements. Requirements engineering as one of the affected disciplines needs to address these trends. (Easterbrook, 2007)

Viewpoints in System Thinking

System thinking offers, besides the definition of the above mentioned key elements, ways to look at the same system from different views. Every viewpoint shifts the focus on how the system with its elements is modelled. It is not possible, to model every single aspect in detail. It is very important to get a clear understanding which message a model should transfer. Models may also highlight different aspects considering the problem itself or the people, which should work with it. (Haberfellner *et al.*, 2018, pp. 35–42)

Common viewpoints are (Haberfellner *et al.*, 2018, pp. 35–42):

- Environment-oriented viewpoint

  This viewpoint considers the system itself as a blackbox and focuses on the environment and the interactions with the system. It is a good way to get a feeling, which external factors influence the system.

- Input-/Output-oriented viewpoint

  This viewpoint also handles the system itself as a blackbox, but focuses directly on how the input of the environment can influence the output of the system, considering its internal behavior. The concrete internal actions are not taken into account here.

- Structure-oriented viewpoint

  The structure inside the system with all its elements and relationships is the primary focus. Guided by the model obtained, it should be possible to retrace, how the output is generated. Mainly, processes and structures are used.

System Hierarchy Concept

The system hierarchy concept helps to prevent the system thinking task from getting lost in too many details. The goal to obtain detailed information where necessary is also fulfilled. The system hierarchy concept can be seen as a function of zooming in: At the beginning, the big picture is modeled, where the major relationships are displayed. If this model is not sufficient for the certain purpose, there is the possibility to get rid of the blackbox approach of the subsystems and inspect one level deeper, where subsystems get defined and are put in relation. This step can be done several times to get a more detailed view on certain aspects if needed. (Haberfellner *et al.*, 2018, pp. 40–41)

4.1.2. Systems Engineering Procedure Model

The procedure model of the systems engineering philosophy is based on four main aspects (Haberfellner *et al.*, 2018, p. 53):

1. Using the top-down approach to start from the big picture and go into detail further
2. Using different variants
3. Grouping system development tasks by time
4. Using a defined problem-solving guidance for every problem

Top-Down Approach

The top-down approach, as shown in Figure 10, is used in systems engineering to enable dealing with complex and big ecosystems. Overall, it is recommended starting with a rough view, including the environment and to lighten up levels of detail step by step. This way of thinking was already mentioned in the system hierarchy concept and uses the transition from black-boxes to white-boxes. The top-down approach helps to get an important overview at the beginning and to redefine the view for certain problems. It may also include redefinitions and direction changes if needed during (detailed) analysis. (Haberfellner *et al.*, 2018, pp. 54–57)

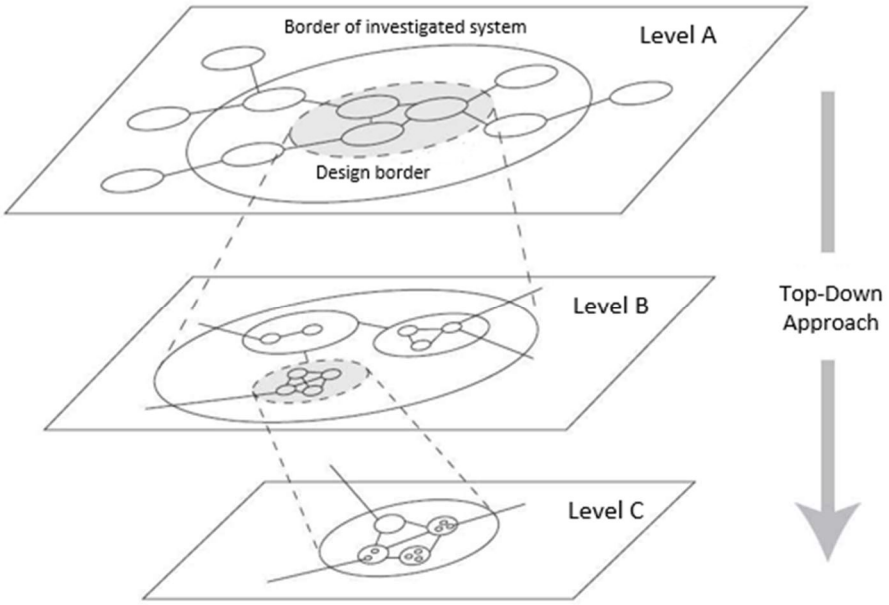

Figure 10: Top-Down Approach, Based On (Haberfellner et al., 2018, p. 56)

Principle of Variants

Systems engineering is designed to find the best possible solution for a certain problem structure. The principle of variants is used to consider alternatives at different levels of detail to ensure the best solution will be found. The focus should not be on the first plausible solution. (Haberfellner *et al.*, 2018, pp. 57–60)
Considering the top-down approach as shown in Figure 10, the optimal solution finding process would be (Haberfellner *et al.*, 2018, pp. 57–60):

- Starting at the top with a very low level of detail
- Investigating all possible variants for the level of detail
- For each found variant, work out all possible, more detailed versions (next level)

It should be clear that finding all different variants for every level of detail in advance would guarantee the best decision. Concerning time and invested resources, this is in fact not the preferred way in practice, as with every level of detail the number of variants will increase dramatically. (Haberfellner *et al.*, 2018, pp. 57–60)
Considering the pros and cons from the optimal solution finding process in systems engineering, the practical workflow would be (Haberfellner *et al.*, 2018, pp. 57–60):

- Considering (parts of) the system as a black-bock for investigating different impacts on input/output.
- Selecting the best variant
- Configuring the best structure for reaching the required impact from the level above, in fact, opening the black box
- Repeating steps from above if further refinement is necessary

Systems Engineering in Project Phases

Dealing with systems engineering and the prior mentioned methods, like the top-down approach and thinking in variants, it is very useful to part the overall process into separate project steps which offer opportunities for control, rearrangement and decision making. Popular project phases include (Haberfellner *et al.*, 2018, pp. 61–69):

- Kick-off

  Moment where the need for solving a certain problem is recognized and resources are available for potentially solving the problem.

- Pre-study

  Highly interrelates with the black-box-oriented step in the top-down approach where the big picture is modelled, figuring out different possible solution approaches, including the systems' environment. This step is not going into detail concerning exact structures and solution elements.

- Main study

  The main aim of this step is to make a decision about the core concept of the solution, thinking about structures and architecture to support the pre-study results.

- Detailed studies

  Detailed studies are carried out for certain, detailed system parts which have to be investigated further. Those parts should be concretized as far as possible to verify a riskless implementation and introduction of the system part.

- System implementation and testing

  Represents the actual building of the modelled system. The way implementation and testing is done can vary a lot between different project structures.

- System introduction

  It is the main introduction of the finished system, or parts of it. Always comes with a high risk that unknown side effects may occur.

- Finalization of the project

  The project gets terminated and the final solution is approved by the customer.

Problem Solving Cycle

Looking at the project phases provided in the previous chapter, another model can be introduced to support the problem-solving issues in every phase of a project. Considering the project phases as a macro cycle, this can be seen as a micro cycle. (Haberfellner *et al.*, 2018, pp. 70–81)
The classical problem solving cycle is basically described in Hall (1962) and includes the following steps (Haberfellner *et al.*, 2018, pp. 70–81; Hall, 1962):

- Kick-off
- Situation analysis

  Clarifying the starting point, mainly answering the questions, like, what the current situation and the actual problem is about. Side conditions, e.g. previous made decisions, the environment or time limits, should be collected.

- Target definition

  Based on the delivered information from the situation analysis, the target(s) should be formalized and documented. It is very important that the aims also consider other targets from higher levels. As in nearly every framework, the main aims should be realistic and reachable with the underlying resources. If there occur conflicts between different goals, it may be a good solution to prioritize them.

- Synthesis of solutions

  Considering the information from the situation analysis and the goals derived, it is necessary to come up with possible solution variants. The level of detail depends on the current project phase in which the problem solution cycle is started. Creativity techniques are important tools for finding various solutions for pre-defined problems.

- Analysis of solutions

  After the constructive generation of possible solution variants, solution analysis is applied to determine whether a concrete solution approach is suitable in terms of e.g. requirements of the entire system, integration, economic terms.

- Solution rating

  Rating methods are used to identify the best solution between all possible variants. During the analysis of solutions, the focus was on the suitability. Here, different approaches, like rentability proofing, cost-value interpretations, are important.

- Decision

  Based on the previous steps, a decision for a certain solution must be made.

- Result

  The final result of the solution finding process may be a decision for a certain solution or the insight that within the current boundaries, a suitable solution cannot be figured out. This can lead to different next steps, from redefining the problem, over rearranging resources or even to the cancellation of the project.

4.2. Problem Solving Process

The systems engineering philosophy is the key mentality behind the real problem solution finding process. The previous defined principles, e.g. the top-down approach, thinking in variants, the problem-solving cycle, will be enrolled during the actual system generation process. The main steps of the system engineering process include (Haberfellner *et al.*, 2018, pp. 129–130):

- System designing

  The definition and processing of the problem which must be solved with the help of a certain system. Definition of the system to fulfill the solutions' conditions.

- Project management

  All necessary activities to support the organization of the project. Contains various fields, like pure organizational tasks, decision making, power distribution, economical calculations.

4.2.1. Model-Based Systems Engineering

The classical way of representing and maintaining systems engineering processes with the help of documents is being replaced in the past years. The main reason behind that development is that systems today are not pure hardware or software systems anymore, they are more a complex network of integrated hardware and software aspects. Document-based processes are not able to reflect this linked and rapidly changing network as integration is hardly representable and maintainable with documents. (Haberfellner *et al.*, 2018, p. 162)

The International Council on Systems Engineering (INCOSE) defined and actively drives forward the model-based systems engineering approach. Model-based systems engineering is an approach, which tries to replace the document-based development by a model-centered approach. This enables integrated design, requirements engineering, but also other fields as verification and analysis throughout the whole development process. The model-centered thinking is not a completely newly invented approach, it has its roots in different other branches, such as mechanical or software engineering. (Object Management Group, 2019a; Haberfellner *et al.*, 2018, pp. 162–163)

The center of the approach is the system model, which can be understood as an eco-system with the results of the different development processes acting as elements, e.g. requirements, design patterns. These elements interact with each other, in fact they are linked which causes influences on change. (Haberfellner *et al.*, 2018, pp. 162–163)

OMG Systems Modeling Language (OMG SysML, or shortly SysML) is a defined modeling language which supports the (model-based) systems engineering process with its core components. It defines on the one hand graphical notations, but on the other hand also semantic tools for representing e.g. behavior or requirements. (Object Management Group, 2019b)

Generally, SysML can be seen as a part of UML2 with additional extensions which are necessary to fit the requirements of the systems engineering process. (Object Management Group, 2019b)

For data interchange between the concrete modeled parts of the system, the model-based systems engineering is mainly using two different interfaces (Object Management Group, 2019b):

1. The ISO 10303-233:2012 (International Organization for Standardization, 2018)
2. The OMG XML Metadata Interchange (XMI) which is compatible with the first one

4.3. Current Systems Engineering Tasks

As already mentioned in prior chapters, systems nowadays are not sharply limited standalone systems with hardly any interaction. Today they are more about highly networked systems with components from classical (hardware) systems and software parts. (Turner *et al.*, 2009)
This makes an integration of two related, but in many points differently executed disciplines necessary: Systems engineering and software engineering. Both disciplines share many applied tools, but their basic intentions and prerequisites are often completely different and need to be handled differently. (Turner *et al.*, 2009)
A first important challenge is the fact that the way of thinking is completely different in both engineering approaches. Software engineers are used to basically have an unlimited horizon of possible solutions where nearly everything is possible. System engineers on the other side are used to be bound to e.g. physical laws. Additionally, software engineers potentially focus on correctness of their implemented software whereas system engineers try to emphasize safety or reliability needs. (Turner *et al.*, 2009)
There are various methods, which have been adopted from either approach to be applied to another one. There are methodologies used in both approaches as they share a common baseline with analysis, requirements engineering, design and verification/validation. The ISO/IEC/IEEE 15288:2015 offers an overview of a framework which is usable with nearly all systems, including hardware, software, processes and so on. (International Organization for Standardization *et al.*, 2015)

4.3.1. Integrating Framework

An integrating framework for software engineering and systems engineering is called “Touchpoint”. It is based on four main components which have to be identified for solving the integration task of both disciplines (Turner *et al.*, 2009):

1. Processes
2. Touchpoints
3. Faults
4. Resolution Strategies

The first step is to identify the processes which are applied during the systems and software engineering disciplines. As a help for identifying those, the ISO/IEC/IEEE 15288:2015 can be used. (Turner *et al.*, 2009; International Organization for Standardization *et al.*, 2015)

Second, touchpoints between the applied processes have to be identified. A touchpoint is given if the corresponding activities and their interrelations affect the outcome of the whole project. (Turner *et al.*, 2009)
Third, faults are identified why the interaction between the two related processes negatively influences the value outcome. Faults can be (Turner *et al.*, 2009):

- Gaps where the interaction between the two engineering approaches is missing
- Clashes where the two applied processes' results are not compatible with each other
- Waste where resources are used in both disciplines for producing the same result

Lastly, resolution strategies need to be found for fixing the faults found. Resolution strategies can be further described in more detail (Turner *et al.*, 2009):

- Processes
- People
- Environment
- Technology

The Touchpoint framework offers a great opportunity to start working on the integration task between the two engineering disciplines of systems and software. A lot of time and resources have to be invested to define in detail where both disciplines interact with each other and where the interaction is not done in a valuable way in each project. (Turner *et al.*, 2009)

5. Implementation of a Visualization Tool

Nowadays business models are complex and intense networked buildings, whereas different influences with various interactors intersect. As already mentioned in chapter 4, systems engineering, systems are complex, highly networked structures with various influences. It is not an easy task to identify key players and main influences in complex graphs. For this reason, a value network notation, which enhances an existing notation from Biem and Caswell (2008), described in Vorraber (2019), is used as a basis for this practical part. The notation is based on different layers, which try to focus on numerous aspects of the business model network graph. Actors are introduced which influence each other by common links. The links get further refined by added labels and annotations. (Vorraber, 2019)

5.1. Notation

Different layers are introduced to look at the same graph from different perspectives and to mainly highlight different important aspects (Vorraber, 2019):

- Value exchange and resource layer
- Legal layer
- Values and needs layer
- Dynamics and motivation layer

5.1.1. Value Exchange and Resource Layer

The value exchange and resource layer helps to analyze different actors in a certain network, especially which values are exchanged between them. The value exchanges are modelled by links between the actors, as shown in Figure 11. These exchanges can carry different labels, each standing for a specific value type. The actors themselves are displayed as circles with sectors, describing their capabilities and assets. (Vorraber, 2019; Biem and Caswell, 2008)

Important here is also the differentiation between provided and received values. The first ones are represented with solid lines, whereas the second ones are modelled with dashed lines. (Vorraber, 2019; Biem and Caswell, 2008)

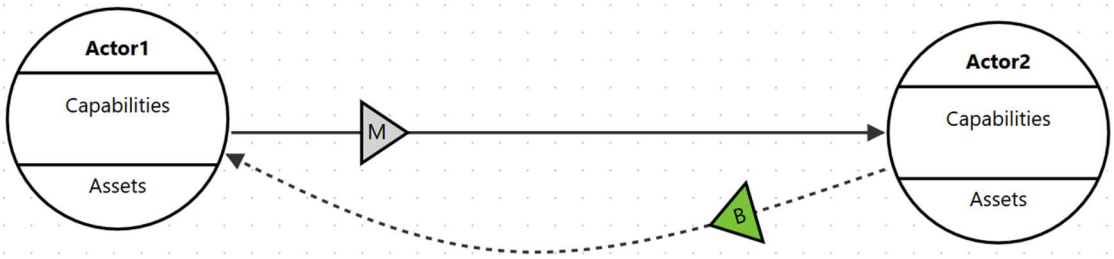

Figure 11: Sample Actors With a Simple Value Exchange Situation

5.1.2. Legal Layer

Legal regulations are always important and must be supervised during a business model development accordingly. For that reason, the legal layer offers the possibility to keep track of the legal compliance of the actors as well as of their value exchanges. For the legal obligations, square labels – appended on the links – are used. A simple traffic light system is introduced to show the state of legal compliance. Figure 12 shows example legal layer elements. (Vorraber, 2019; Vorraber *et al.*, 2016)

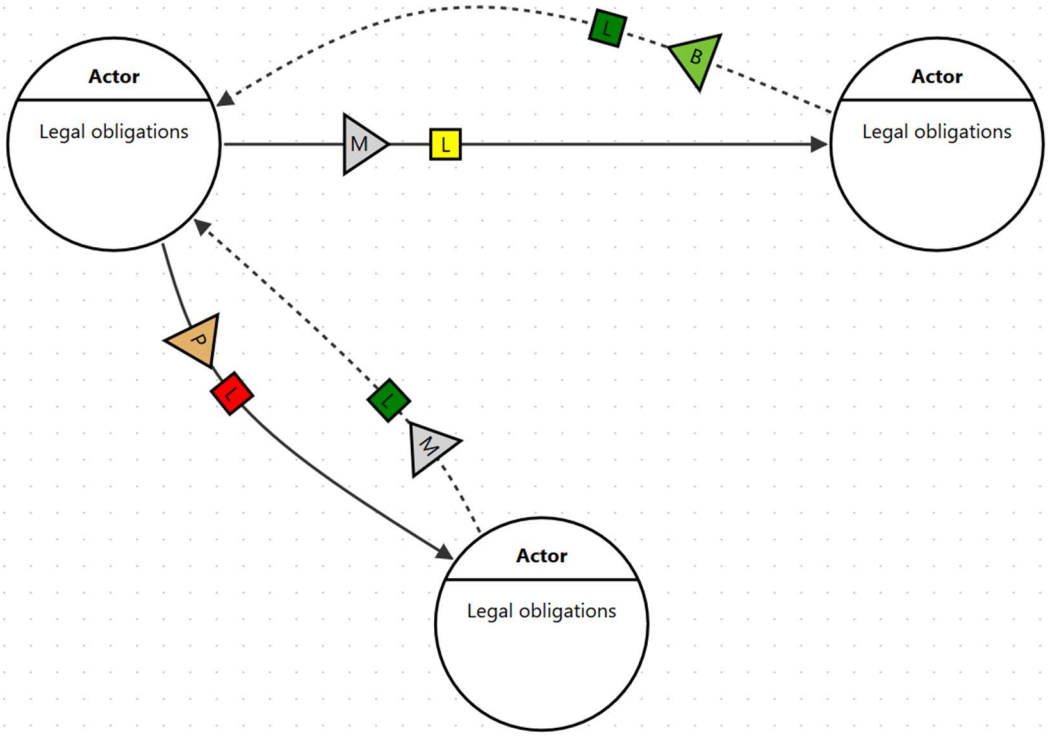

Figure 12: Sample Legal Layer Elements

5.1.3. Dynamics and Motivation Layer

The dynamics and motivation layer is based on the main works of Vroom (1967) and Porter and Lawler (1968), whereas actors underlie external and internal factors, which influence their activity in a certain network. The following elements, in enhancement to Biem and Caswell (2008), were defined (Vorraber, 2019; Vorraber and Vössner, 2011):

- Endogenous motivation, emphasizing the internal motivational factors for a certain actor within the entity
- Exogenous motivation, modeling the external forces the actor is exposed to when fulfilling the value activity (Kelman, 1961).

Each of the motivational elements can be marked as defensive, neutral or active. An example can be seen in Figure 13. (Vorraber and Vössner, 2011; Vorraber, 2019)
Additional concepts introduced are those of the value engines/breaks. (Vorraber *et al.*, 2019)
Value exchanges, which lead to positive benefits, may lead to positive values generated by the connected actors. In the end, considering positive influence in form of a cycle, value engines are depicted to express such a construct. The other way around, where negative benefits are identified, value breaks can be modeled, which express the overall, less supported value creation. The concept of value engines/breaks highlights the shift from looking at single entities for analyzing their business models to rather examining the whole interconnected network. This enables a deeper understanding of how dynamics change the way value is created/perceived. (Vorraber, 2019; Vorraber *et al.*, 2019)

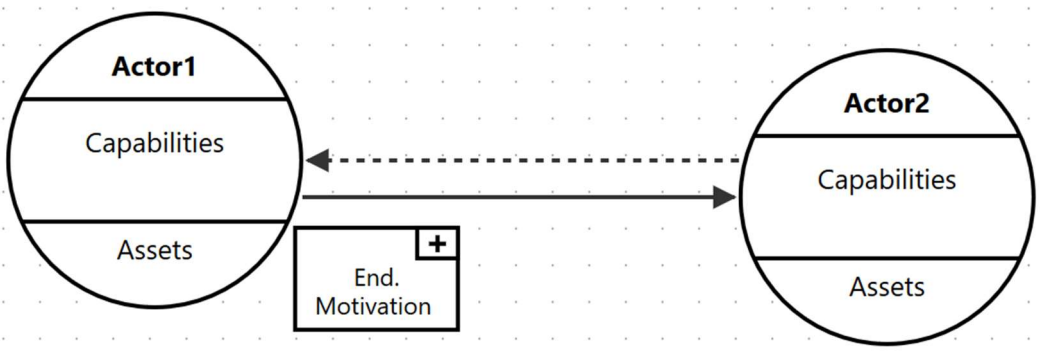

Figure 13: Endogenous Motivation Example

5.1.4. Values and Needs Layer

Business model tools like the business model canvas (Osterwalder and Pigneur, 2010) focus on value creation, considering especially customers and providers. Ethical values are not considered in detail. But for the purpose of socio-techno-economic systems, where all different viewpoints meet, ethical values can be an important part for modeling to understand the background of each actor. Other tools, like Tandemic (2019) and Breuer and Lüdeke-Freund (2017), try to fill that missing link. (Vorraber, 2019)
The following enhancements to Breuer and Lüdeke-Freund (2017) are intended to cover ethical and other values in a detailed and networked aspect (Vorraber, 2019):

- Functional needs
- Technical non-functional needs
- Social economic needs
- Social human needs
- Ethical needs
- Safety needs

The fact that another layer is introduced is definitely worth the time. One of these values can risk the whole business model if one of the actors introduces some negative value. (Vorraber, 2019)
Figure 14 shows a small example how the actual needs can be marked in a range between met and not met.

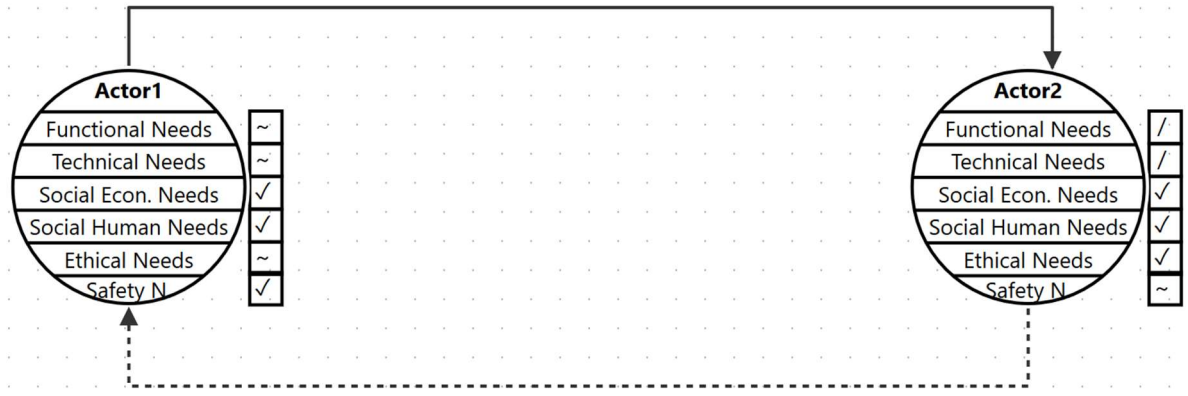

Figure 14: Sample Value Needs Elements

5.1.5. Overview

Figure 15 shows an overview of the basic elements, including actors, labels, links, etc., which are used in the notation.

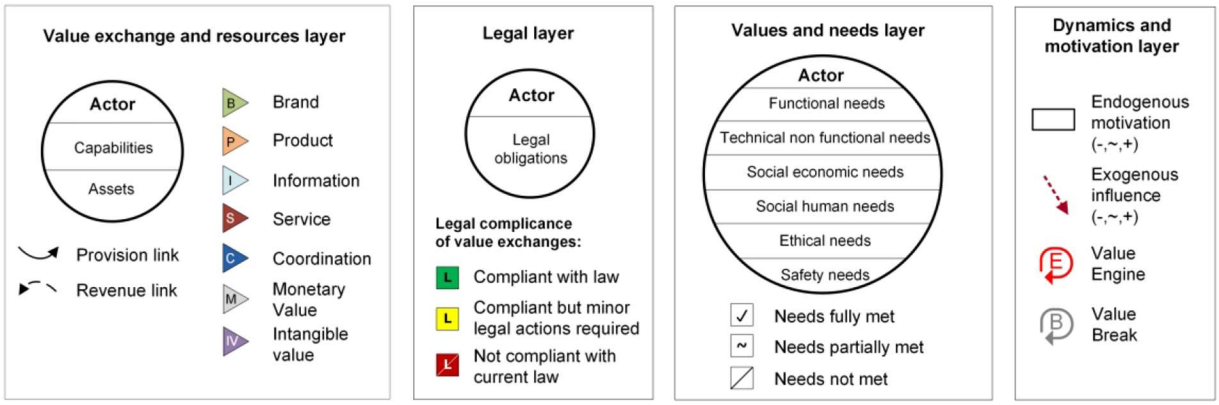

Figure 15: Overview of the Basic Elements of All Layers (Vorraber, 2019, p. 37)

5.2. Tool introduction

The need for modeling and evaluating the before-mentioned value network using the shown notation made a visualization tool, fitting exactly the purpose of the explained requirements, necessary. There exist various visualization tools, which are well approved and help visualizing different kinds of graphical elements, including desktop version programs, like Microsoft Visio, and browser-based applications.

The main problem with these existing graphical tools is not their implementation, but their purpose behind the scene. The graphical existing tools offer well-defined, approved and thought through user interfaces, which cannot be reached in a short period of time.

The main advantage of the newly generated visualization tool – in its current version - is the fact, that it offers:

- Design for supporting the above-mentioned value network notation with all its elements
- Support for the value network creation workflow
- Instantly switching between different layers to enable a quick editing and illustration of different views on one and the same graph
- Individual, direct support for future needs

5.3. Selection of Technologies

At the beginning of the project, decisions have to be made concerning the usage of technologies. The selected technologies will influence the whole project. The final product, including its user experience, is mainly dependent on the chosen alternatives.

5.3.1. Desktop-Version vs. Web Application

The first main decision is about whether to design a standalone desktop version program or a browser-based web application. Table 2 and Table 3 show a short comparison between both approaches which was very useful to find a decision.

Table 2: Pros and Cons of Desktop Version

Desktop Program Version

Pros	*Cons*
+Usage of mature programming languages, as C++/Java/C#	-More prerequisites for users
+Independency of browser applications and their different support levels	-Different OS systems
+Better resource management	-Collaborative work problematic

Table 3: Pros and Cons of Browser-based Application

Browser-based Web Application

Pros	*Cons*
+Nearly instant go for user (no installations mainly)	-Resource management in browsers
+Quick start from anywhere	-Probably more intense work with browser-based programming languages (error seeking, browser compatibility)
+Great technologies for graphical drawing (HTML5, JavaScript)	
+Reachability in web	
+Collaborative working straightforward	
+Easy server-communication via pre-defined protocols and languages (HTTP, PHP)	

The choice for an alternative was a very quick decision, as obviously the pros of the browser-based application definitely outweigh those of the desktop version, concerning the purpose of the visualization tool in this case.

5.3.2. Technologies for the Web-based Application

For the browser-based technology development, a web-server, a database and a client/server-side scripting language will be needed. In its current project version, the browser-based application does not require to implement server-side functionality. Therefore, technologies are simply selected on the client-side. Nevertheless, there will be a full extension of the server-side in the future. For that reason, the server-side technologies will also be dealt with. A full description of planned activities will be included in section 0.
The layout itself of the web page will be done with HTML5 (Hypertext Markup Language) and CSS (Cascading Style Sheets). HTML5 offers great extensions to its prior version which e.g. help to deal with inputs or drawing.

Client-side Software

The software decision on the client-side is in fact just about which browser is installed and used. The most common browsers, which are currently on the market, are (Refsnes Data, 2019):

- Google Chrome
- Mozilla Firefox
- Safari
- Opera
- Microsoft Edge/Internet Explorer

Client-side Scripting Language

The main part of the visualization tool will be the client-side scripting as for the current project step, the application will be fully executable on the client. The choice of the scripting language was made at the beginning of the project, JavaScript is the selected option. It offers easy possibilities to build a fully interactive web-page with DOM-manipulation as wells as event handling.

Server-side Software Tools

As already mentioned, the server-side will not be implemented in the current version of the project, but it is intended to be included in future works. Therefore, an overview of the main aspects or software tools will be worked out.

Node.js

Node.js is an event-driven, JavaScript based software for creating web-services. JavaScript can be directly used for programming the server functionalities. Events are used instead of threads to lower the risk of deadlocking or blocking threads. (Joyent Inc., 2019)

Apache HTTP Server

The Apache HTTP Server is a well-established web server, which can be easily customized and is used in a wide area. For writing server-side scripts, a programming language, like PHP, Python or Ruby, has to be used. (Apache Software Foundation, 2019)

Twisted

Twisted is a Python-based software, offering web-server functionalities. Twisted offers an event-driven web-server written in Python. (Twisted Matrix Labs, 2019)

Databases

MySQL

MySQL is very popular relational database system. There is a free version as well as a premium version available. In earlier times MySQL was an open-source project, maintained by a wide community. Since Oracle bought MySQL there has been a shift from MySQL to MariaDB in the open-source-community. Nevertheless, MySQL is a widely used database with high security features. (Oracle Corporation, 2019; MariaDB Foundation, 2019a)

MariaDB

MariaDB is a database server engine developed by the developers of MySQL as an open-source project which will stay open-source according to MariaDB Foundation (2019a). It is almost fully compatible with MySQL with some exceptions. (MariaDB Foundation, 2019b)

Scripting Languages

Various programming languages can be used to enable the server-side scripting. A broad used language is PHP. But like mentioned in the Node.js section, JavaScript is also a possible language as well as Python or Ruby. It is a more personal decision which language to use.

Complete Packages

XAMPP

XAMPP is a free package which includes Apache, MariaDB, PHP and Pearl. It is simple to set up and can be used for developing tasks like browser-based development as everything which is needed is included in the package. It is available for all common operating system platforms. (Apache Friends, 2019b)

WAMPP

WAMPP is, similar to XAMPP, a complete package which includes Apache, MySQL and PHP, but it is solely developed for Windows. It is also a subjective decision which one of the packages to prefer, WAMPP is especially developed for the use under Windows. One argument for WAMPP for the usage of the final software as a real product could be a more thought through system concerning security issues, XAMPP in its download version is emphasized to be used for development solely. (Alter Way, 2019; Apache Friends, 2019a)

5.4. Tool overview and Prerequisites

The visualization tool is designed as a browser-based application which can be simply used in a normal web-browser. In its current version there is no need of registering for using the system, everything can be made on the client-side (For further development activities, please refer to chapter 0).
All functionalities are tested under:

- Firefox (66.0.3, 64 bit)
- Google Chrome (73.0.3683.103, 64-Bit)

Basically, a browser and access to the source files is all that is needed to start working with the visualization tool.

5.5. Architecture

The basic software architecture is a very simple one: As already described in the previous section, the current version is absolutely fine with working on the client-side only. There is no need for a server-side handling, as no user registration nor a saving of a certain user session is needed.
The architecture, shown in Figure 16, can be explained in a few sentences: The basic HTML-page gets rendered by a user chosen web-browser. The CSS-files help to render the style. The JavaScript-files, which are also stored on the client dynamically, update the HTML rendering of the site according to individually programmed events. The main program logic is contained in the individually programmed JavaScript-files.

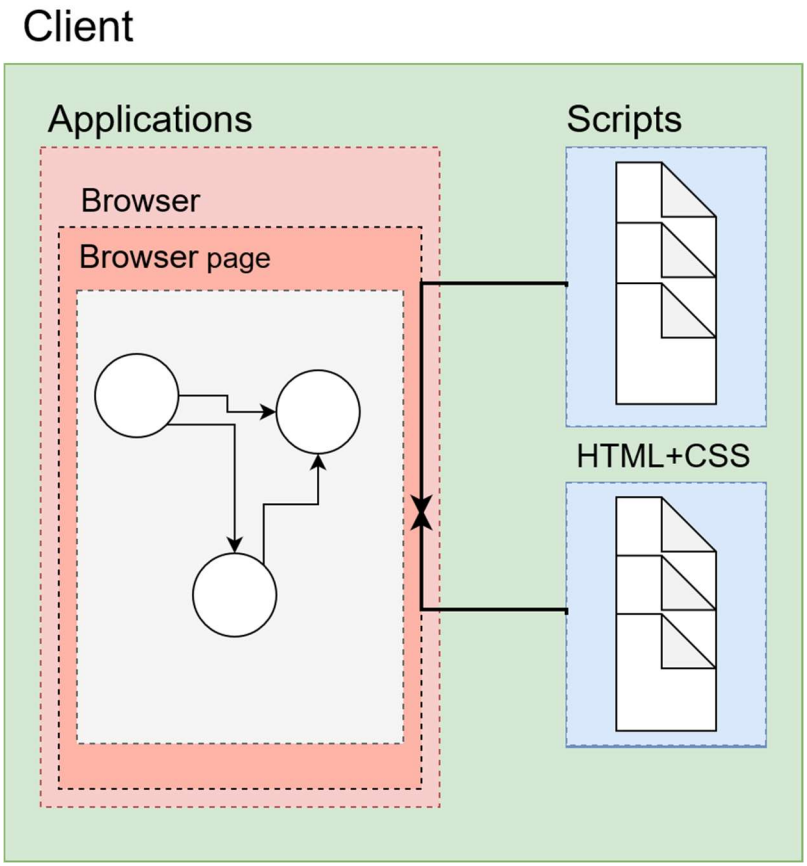

Figure 16: Basic Informal Architecture of the Visualization Tool

5.6. Design

The design of the implemented visualization tool is determining the way, how the background data is saved and how the visualization is rendered. This section will give an overview of the used patterns and software tools. The architecture in the form of an UML component diagram is shown in Figure 17.

5.6.1. Libraries

JointJS

The framework “JointJS”, described in client.IO (2019a), was used for basic administration of the graph model and view rendering. JointJS is a simple tool which supports building diagram visualization tools. There is also a commercial version, called “Rappid”. The JointJS is licensed under the Mozilla Public License (MPL) 2.0. For obtaining the MPL 2.0 please visit: https://www.mozilla.org/en-US/MPL/2.0/
The dependencies needed for JointJS (lodash.js, jquery.js, backbone.js) are released under the MIT-license.
JointJS is a simple library which helps to work with basic graphical elements. It uses SVG and guarantees a strict separation of model and view (client.IO, 2019b). JointJS does **NOT**:

- Replace individual programming work needed for this certain project.
- Include the main features of the value network visualization tool.
- Hinder from directly programming SVG elements or editing HTML elements.
- Overrule CSS styles.
- Limit current and further development activities.

What JointJS offers, is:

- Standard definition of lightweight, graphical elements (circle, rectangle, basic links).
- Separation of model and view.
- JSON-export and import of the model behind the graph.
- Great customization (elements, events) potential which was explicitly needed for this task.
- Very good documentation.
- Could be removed and replaced if potential conflicts occur in future.

Bootstrap

Bootstrap (Bootstrap, 2019) is used just for some minor layout properties, like pop-up-windows and small modals. The basic page layout is done in pure HTML+CSS. Bootstrap offers a huge selection of fixed designed web components. It is released under the MIT-license.

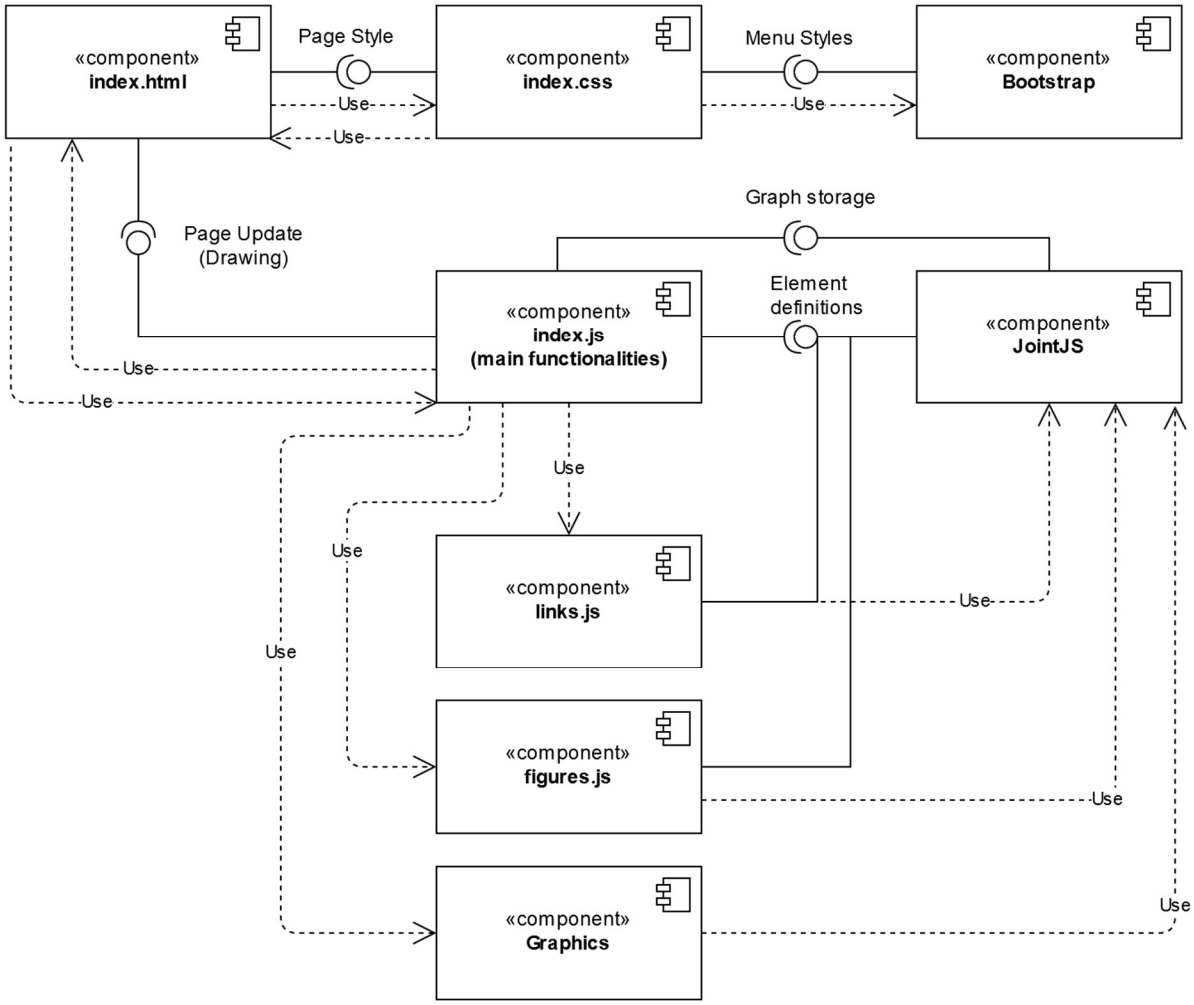

Figure 17: UML Component Diagram of Architecture

5.6.2. Design Pattern

As already mentioned, the basic design of this visualization tool is using the MV(C) design pattern (see Figure 18). MVC stands for Model, View, Controller and strictly enforces to separate the data from the actual graphical rendering. The controller part is responsible for reacting to events in the graphical user interface (the view) and change the underlying model accordingly. Changes in the model subsequently influence the view as the data is the base for the graphical representation. (Krasner and Pope, 1988)

It is a widely used concept to strictly separate data and view, as data should be stored in some kind of storage system, whereas the view itself gets rendered by a certain application. Mixing those approaches would result in an unstructured and unclean way of data processing. Furthermore, it is a commonly accepted and usual way to separate different needs of software in independent logical units, which could be easily exchanged or edited without the need of looking at the other part. (Krasner and Pope, 1988)

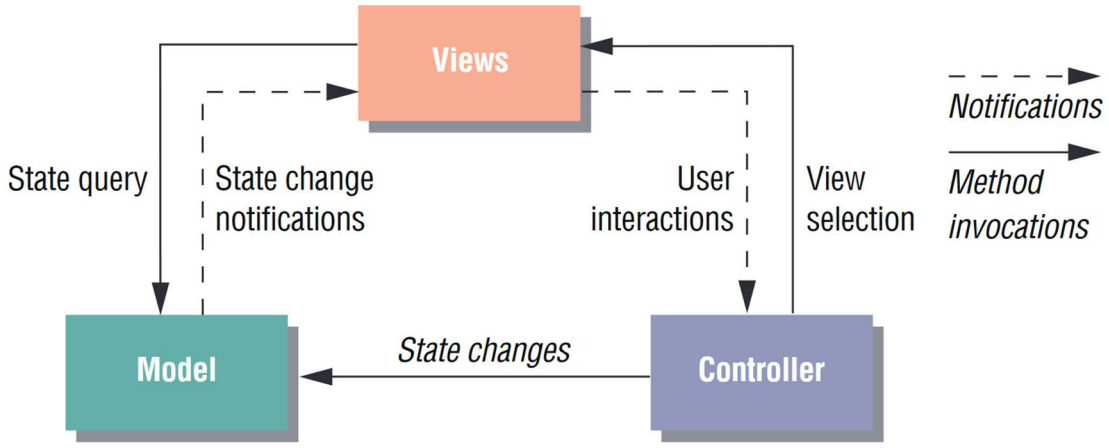

Figure 18: MVC Pattern (Curry and Grace, 2008, p. 88)

JointJS is using the MV-principle by simply completely separating the data, saved in the model, from the view, saved in the so-called paper. Events can be triggered on base of the model(s) as well as based on the paper(s). (client.IO, 2019b)

The controller part is not explicitly implemented (there are some exceptions concerning e.g. link creation). This has to be done from the person who uses the library.

5.7. Requirements

Table 4 to Table 7 should give an overview of the defined requirements, which have been written down during the project meetings. The requirements (present requirements as well as future requirements) are divided into the following categories:

- Functional requirements
- Graphical requirements
- Security requirements
- Legal requirements
- Quality requirements

Table 4: Functional Requirements

Requirement ID	Short Name	Description
F1	Element Drawing	The user shall be enabled to add layer elements to the drawing graph. Elements can be: • Actor elements • Link labels • Annotations • Text fields
F2	Drag and Drop	Drawn elements shall be movable via drag and drop.
F3	Element Selection	The user shall be enabled to select elements, which have been inserted in the graph.

F4	Element Deletion	The user shall be enabled to delete elements from the drawing. Deletion should be possible only for selected elements.
F5	Element Resizing	The user shall be enabled to resize the drawn elements (except links), the resize functionality should automatically be enabled when the user hovers the cursor over one of the corners of the element. The functionality shall be visualized by a changing cursor style. The outer corners are defined by 80% of the width and height of the corresponding object.
F6	Link Adding	The system shall enable the user to draw links between actor elements.
F7	Link Deletion	The system shall provide a possibility to delete inserted links.
F8	Label Adding	The user shall be able to add labels of all layers on to the connection links. Label positions should be automatically calculated.
F9	Label Rotation	The appended labels shall rotate according to the links direction to show the direction of the appended label.
F10	Link Intersections	The system should detect link path intersections and correct them.
F11	Link Selection	The user shall be able to select the link type, which is drawn when connecting elements. Link Types are: • Solid line links (black) • Dashed line links (black) • Dashed line links (red) (exogenous link)

F12	Actor Layer Switching	The user shall be able to switch between the actor layers. There must always be exactly one actor layer shown. Actor layers include: • Value exchange and resources layer • Legal layer • Values and needs layer
F13	Layer Selection	The user shall be able to select the layers for drawing labels and annotation elements. The layer selection is also defining, which labels and annotation elements are displayed in the graph. The selectable layers are: • Value exchange and resources layer • Legal layer • Values and needs layer • Dynamics and motivation layer The number of selected layers can be between zero and four.
F14	Text Editing	The user shall be able to edit text contents inside the actor elements and text fields.
F15	Text Arrangement	The system should automatically identify the needed space for the input text and arrange the font size and line breaks to fit the text inside the text area. The minimum font size should be 12pt.

F16	MVC	The system shall consider the MVC design pattern. It shall strictly limit the model from the user view. Every change in the model shall update the user view.
F17	Graph Download	The system shall enable the user to download her/his current work progress in form of a JSON-file.
F18	Graph Upload	The user shall be able to upload a JSON-file which contains a graph definition previously downloaded with the graph download functionality.
F19	Image Download	The system shall offer the user the possibility to download her/his current work progress in form of a PNG-image.
F20	Undo	The system shall offer the user the possibility to undo up to the last 15 changes, which have been made on the graph. The undo shall be activated either by a keypress combination (CTRL+Z) or by a button.

Table 5: Graphical Requirements

Requirement ID	Short Name	Description
G1	Basic Layout	The system should be arranged with the following layout: One menu header, one left-side menu for the model elements, one main drawing area and one footer element. (See Figure 19)
G2	Upload Pop Up	The upload of the JSON-file shall trigger a pop-up window, which offers the possibility to seek the local file system for finding an appropriate file.
G3	Draft Bar	On the left, the system should display a draft area. This area is for dragging draft elements onto the main graph. (See Figure 20)
G4	Draft Bar Subdivision	The draft area shall be divided into categorial subsections. The subsections should pop-up/down on click. The draft bar shall be divided into the following sections (see Figure 20): • Actor elements • Labels and Annotations • Links

Table 6: Quality Requirements

Requirement ID	Short Name	Description
Q1	Browser Functionality	The system shall offer the same functionality in both defined browser applications.
Q2	Layer Switching Time	The system shall operate the layer switching functionality within one second after the user has triggered the switch.
Q3	Invalid User Actions	The user should be notified by the system if the requested operation, triggered in the user interface, causes an invalid operation. The notification can either be a pop-up window or an undo of the requested operation.
Q4	User Interface Structure	The provided user interface should be sufficiently structured, such as the drawing process is smoothly supported. Smoothly supported means that a user who has used the system for drawing a graph once, can insert every drawing element within three seconds.
Q5	Upload Time	The upload of graphs included in JSON-format should not last longer than two seconds maximum. Small graphs (element number <100) shall be loaded within one second. Medium graphs (element number 100-500) shall be loaded within 1,5 seconds. Big graphs (element number >500) shall be loaded within 2 seconds.

Table 7: Legal Requirements

Requirement ID	Short Name	Description
L1	Licensing	The implemented software shall comply with the licenses of the used libraries.
L2	DSGVO	The system shall be compliant with the DSGVO in Europe.

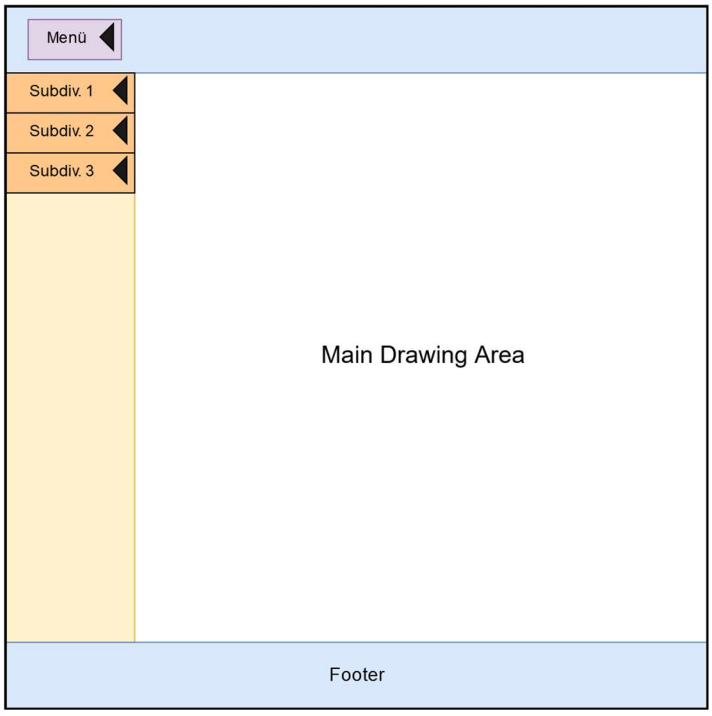

Figure 19: Graphical Draft for Requirement G1

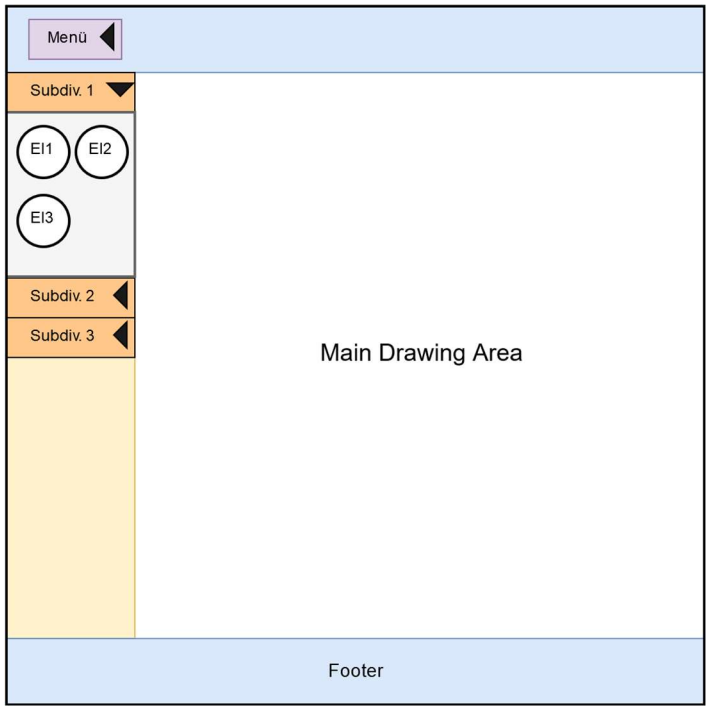

Figure 20: Graphical Draft for Requirement G3 and G4

Semi-formal and graphical explanations of requirements can be seen in Table 8 and Figure 21. Table 8 describes the requirement F12 with a use-case description, whereas Figure 21 uses the UML activity diagram type to show activity flows of requirement F12.

Table 8: Example Use-Case of the Actor Layer Switching Requirement F12 (Semi-Formal)

Use-Case Name	Actor Layer Switching (F12)
Short Description	The user selects an actor layer to be displayed. All inserted actor elements as well as the model elements in the draft area get filtered according to the selection.
Actors	User, System
Trigger	User clicks on selection field for layers.
Precondition	The user has fully loaded the page. There is only one actor layer visible. All previous inserted actor elements were inserted successfully.
Result	The user only sees actors for the selected layer.
Postcondition	The system has successfully hidden all actor elements, which are not from the selected layer type.
Main action plan	1. The system investigates all inserted elements. 2. The system proofs the types of the elements. 3. The system shows matching actor types and hides actors not matching the selected layer. 4. The system updates the affected links. 5. The system updates the draft area actor elements.

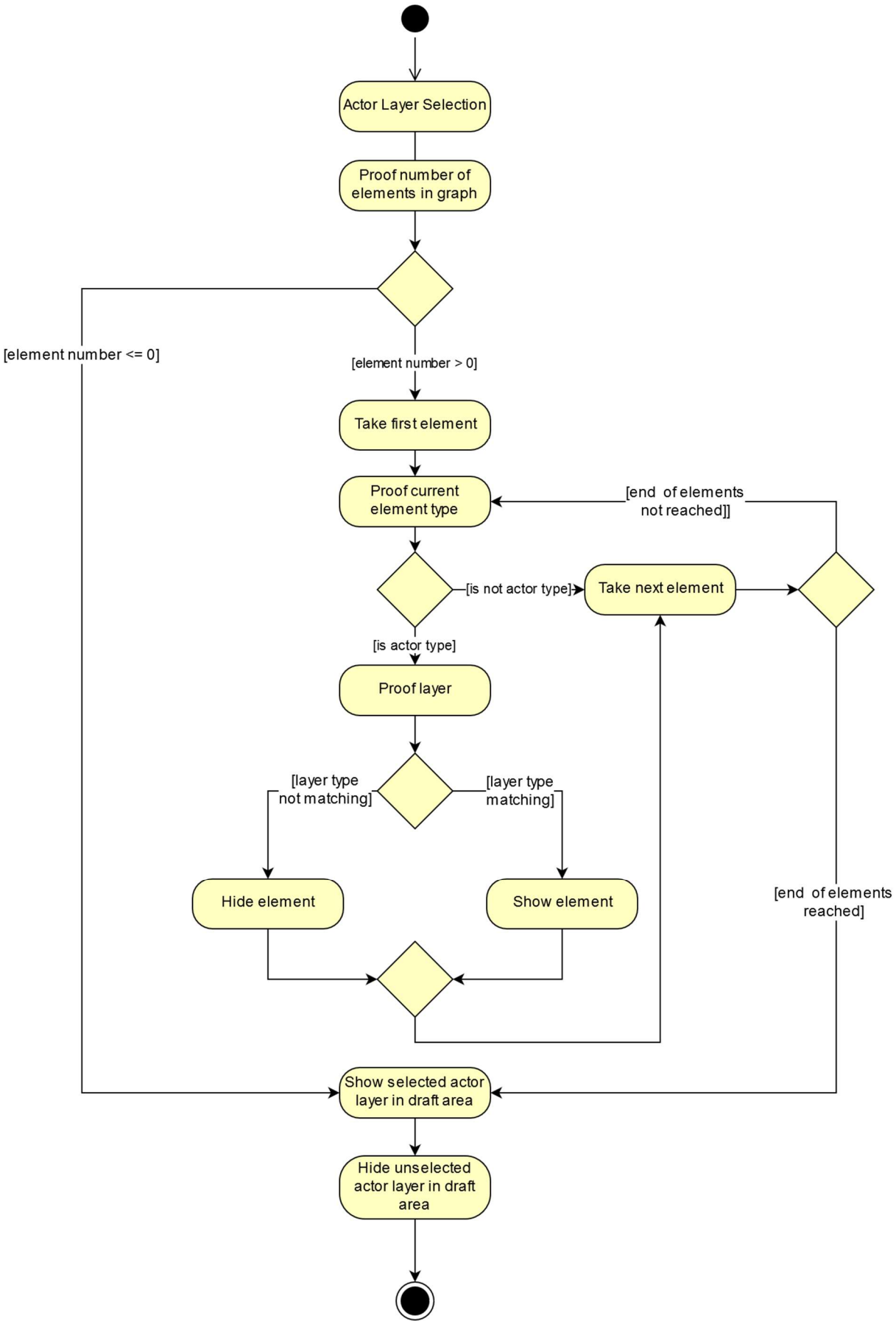

Figure 21: Example UML Activity Diagram for the Actor Layer Switching Requirement

5.8. Implementation Basics

HTML offers a wide area of pre-defined events, including mouse events and keyboard events. Additionally, JointJS offers the possibility to react on various events, either on basis of the graph (model) or paper (view). The program can also react on changes of single elements in the graph. For a complete list of possible events, look at (client.IO, 2019b).
Basically, all functionalities get triggered by the user with a certain event. The main work is to use the right events and alter the correct elements/data.

5.9. Custom Elements

One of the key constructs of the whole visualization tool are the individually implemented elements. Basically, a markup as well as standard attributes have to be defined to ensure an appropriate data saving in the model and to guarantee an exact rendering in the view. Mainly all defined attributes are SVG-attributes as JointJS is working with SVG-elements. There are some individual attributes which are used for the internal implementation itself. Arbitrary figure constructs can be implemented using a combination of different SVG-elements. (client.IO, 2019b, 2019a)

5.9.1. Legal Layer Actor Example Definition

In Figure 22 the definition of the legal layer actor can be found. This definition is used with the constructor to generate standardized legal layer actors. There are various attributes, from defining the circle radius, over defining straight lines for the circle sectors until individual attributes, which help for internal implementations, e.g. “maxTSpans” or “children”.
Further definitions can be found in the full project implementation as the example should be sufficient to give an overview on how the individual components are realized.

```
joint.dia.Element.define('custom.LegalLayer',{
    attrs:{
        main:{
            refCx: '50%',
            refCy: '50%',
            refR: '48%',
            strokeWidth: 2,
            stroke: 'black',
            refWidth: '98%',
            refHeight:'98%',
            fill: 'white',
            textElements: 2,
            refWidthCalc: '100',
            refHeightCalc: '100'
        },
        magnet_helper:{
            refCx: '50%',
            refCy: '50%',
            refR: '49%',
            strokeWidth: 0,
            stroke: 'none',
            refWidth: '100%',
            refHeight:'100%',
            fill: 'white',
        },
        sector1:{
            offsetX: '6',
            endX: '94',
            d: 'M 6 0 L 94 0',
            strokeWidth: 2,
            stroke: 'black',
            refX: '',
            refY: '30%'
        },
        text1:{
            numberChildren: 0,
            //children: tspan html elements
            children: [] ,
            //max number of possible tspan html elements
            maxTSpans: 1,
            currTSpans:1,
            fontSize: 9,
            fontWeight: 'bold',
            fill: 'black',
            text: 'Actor',
            refX:'50%',
            textAnchor:'middle',
            refY: '15%',
            refWidth: '70%',
            refHeight: '15%',
            height: '15%',
            event: 'element:pointerclick',
            contenteditable: true

        },
```

```
        text2:{
            numberChildren: 0,
            //children: tspan html elements
            children: [] ,
            //max number of possible tspan html elements
            maxTSpans: 5,
            currTSpans:1,
            fontSize: 9,
            text: 'Legal obligations',
            refX:'50%',
            textAnchor:'middle',
            refY: '40%',
            refWidth: '70%',
            refHeight: '30%'

        }

    }
},{
    markup:[{
        tagName:'circle',
        selector:'magnet_helper'
    },{
        tagName: 'circle',
        selector: 'main'
    },
    {
        tagName: 'path',
        selector: 'sector1'
    },{
        tagName: 'text',
        selector: 'text1'
    },{
        tagName: 'text',
        selector: 'text2'
    }]
});
```

Figure 22: Example Definition Legal Layer Actor

5.10. Main Functionalities

Now that the basic design, the idea behind the tool and the requirements are defined, a short overview of the main implemented features will be given. For the full functionality, please refer to the implemented project.

5.10.1. Using Draft Elements

One of the first main functionalities to be implemented was the task to enable dragging a copy of the draft elements from the draft side-bar onto the main graph, as shown in Figure 23. The dragging starts with a pointerdown-event on the draft element and ends with a pointerup-event over the main graph. The important background information needed for this functionality is that the draft area(s) and the main graphs are separate models and views. This leads to the fact that elements cannot be simply moved between those areas as the “papers are the limit”.

To overcome this issue, a simple algorithm has been developed, which includes the following main steps:

1. React on cell-pointerdown event in the draft-sidebar areas
2. Create a completely new graph and corresponding paper
3. Clone the clicked element of the draft area
4. Add the clone to the newly created graph
5. Add mousemovelisteners to the newly created paper
6. With every mousemove, rearrange the position of the newly created paper
7. React on cell-pointerup event, which signals that the drop has to be done
8. Proof if element is really dropped on main graph
9. Insert clone of dragged element into main graph
10. Delete temporary created graph and corresponding models of step 2

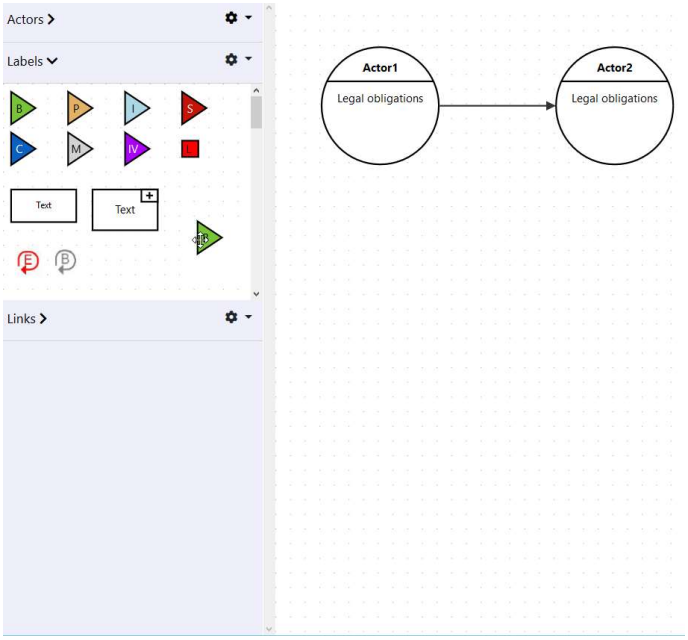

Figure 23: Dragging a Sample Label Element

5.10.2. Link Selection and Link Intersections

To fulfill the not permanently binding requirement of avoiding link intersections, the user has the possibility to choose different types of links. The user can choose between:

- Solid line links
- Dashed line links
- Red dashed line links (exogenous links)

In addition to the different graphical interpretations of links, the user can choose between links which differ in the way their route is rendered. The user can select

- Smooth links
- Manhattan links (orthogonal routing)

The manhattan-type link can detect intersections between different links whereas the smooth link cannot. Nevertheless, the user has the choice to decide which type she/he wants to use. The choice of the link type depends on the graph which is generated. There may be use-cases, where the orthogonal, non-intersecting links are more appropriate, but there may also be scenarios, which desire smooth connections between elements.
To activate the types of links, the user is able to select the different types in the draft-area side bar. The selected link type gets highlighted to show the user the current selection.

```
switch(cellView.model.attributes.type.toLowerCase()){
    case "custom.smoothlinknormal":
        second_paper.options.defaultLink = new
        joint.shapes.custom.SmoothLinkNormal();
        cellView.highlight();
        return;
    case "custom.smoothlinkdashed":
        second_paper.options.defaultLink = new
        joint.shapes.custom.SmoothLinkDashed();
        cellView.highlight();
        return;
    case "custom.manhattanlinknormal":
        second_paper.options.defaultLink = new
        joint.shapes.custom.ManhattanLinkNormal();
        cellView.highlight();
        return;
    case "custom.manhattanlinkdashed":
        second_paper.options.defaultLink = new
        joint.shapes.custom.ManhattanLinkDashed();
        cellView.highlight();
        return;
    case "custom.manhattanexogenousdynamicslink
        second_paper.options.defaultLink = new
        joint.shapes.custom.ManhattanExogenousDynamicsLink();
        cellView.highlight();
        return;
    case "custom.smoothexogenousdynamicslink
        second_paper.options.defaultLink = new
        joint.shapes.custom.SmoothExogenousDynamicsLink();
        cellView.highlight();
        return;
    default:
        alert("Wrong link type in draft paper3. Something is wrong!");
        return;
}
```

Figure 24: Short Code Excerpt For Setting Default Link

Figure 24 shows a small excerpt of the actual implementation code where the default link type is set according to the selected type. The different link types are individually created. A sample definition will be included later. Figure 25 explains the corresponding activity in the user interface.

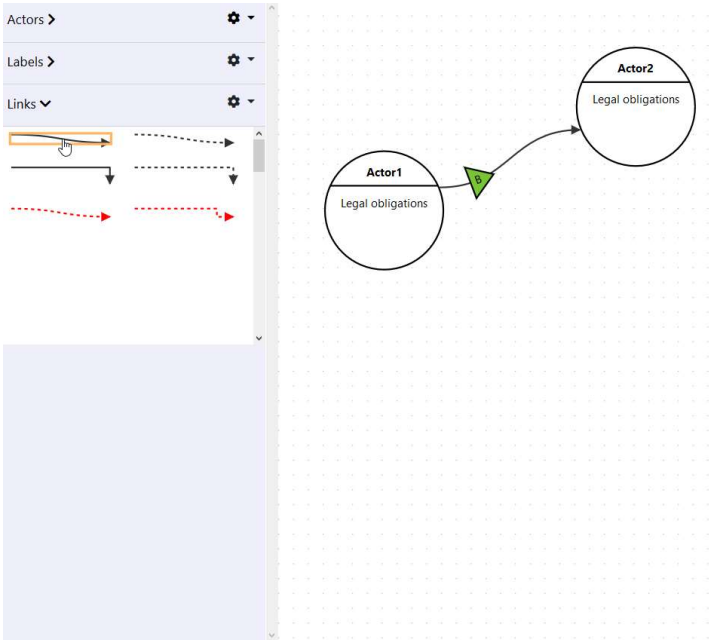

Figure 25: Example of a Link Type Selection

5.10.3. Label Adding and Label Rotation

The user gets the opportunity to drag and drop labels directly from the draft area of the graph onto a link between two elements. If the label is dropped over the link, the label is appended to the link itself, in fact becoming a part of the link. The label position is automatically computed, taking care of the number of already added labels onto the link. Afterwards the label is accordingly moving with the link, including rotations, to always point in the direction of the link. The rotation feature should improve readability of the graph (see Figure 26 and Figure 27).

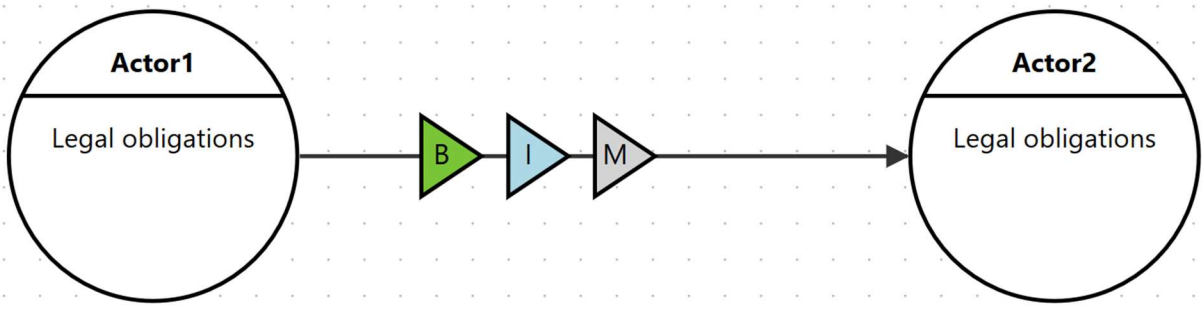

Figure 26: Example1 of Label Rotation

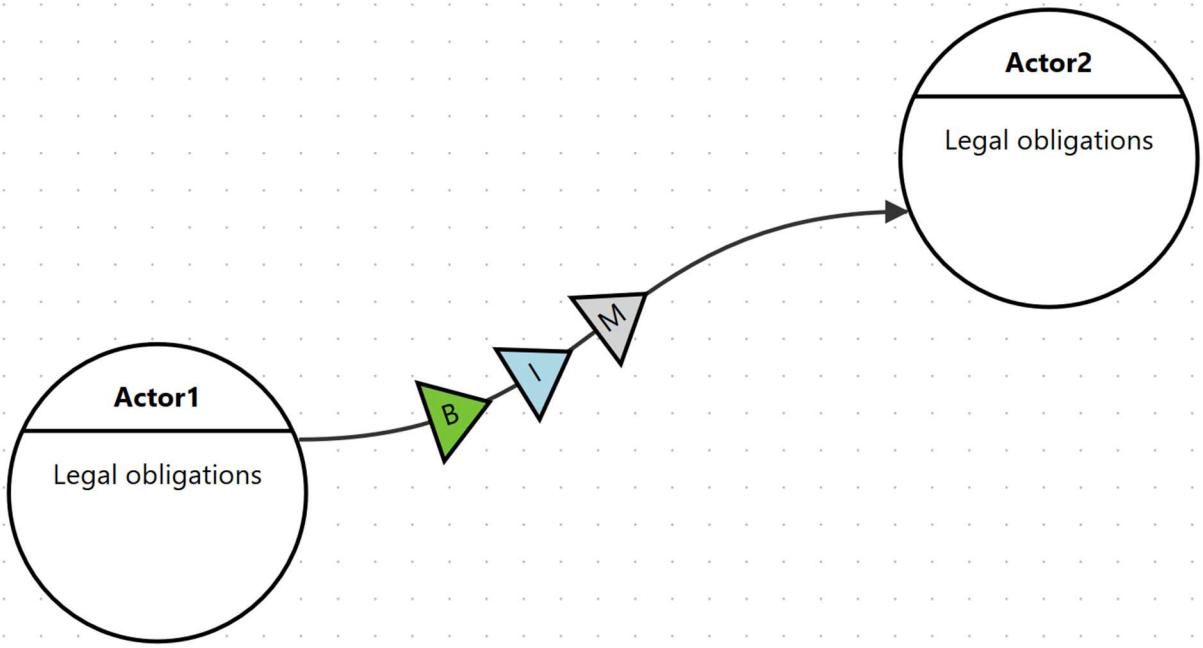

Figure 27: Example2 of Label Rotation

To have a look on the current implementation, the basic idea is to react on the same event as when inserting an element into the main graph. But when the element is from type “label” it should be checked whether it potentially, within a certain delta, crosses a link. If it intersects, the label is directly added to the link. Otherwise, it is inserted as normal element (it still can be dropped on the link later on via drag and drop). The code sample in Figure 28 shows the key part of the implementation code where the actual intersection check is done.

```
function proofAppendLabel(cellView){
    var bbox_dropped_element = cellView.model.getBBox();
    var links = jointjs_graph_main.getLinks();
    for(var i = 0; i < links.length;i++){
        var linkView = links[i].findView(second_paper);
        var closest_point = linkView.getClosestPoint(
        bbox_dropped_element.center());
        var diff = {};
        diff.x = Math.abs(closest_point.x-bbox_dropped_element.center().x);
        diff.y = Math.abs(closest_point.y-bbox_dropped_element.center().y);
        //inserting label intersects with link -->
        //insert label into link and delete old label element
        if(diff.x < 50 && diff.y < 50){
            var markup = Object.assign(cellView.model.attributes.markup);
            var attrs = Object.assign(cellView.model.attributes.attrs);

            //prevent double insert of same labels
            if(links[i].labels().filter(element => element.attrs.text1.text
               ==attrs.text1.text).length != 0){
                cellView.remove();
                continue;
            }
            //adjust position of inserted label (last label + delta)
            var pos = 0.2;
            pos = pos + links[i].labels().length/7.0;
            links[i].appendLabel({markup,attrs,position:{distance:pos}});
            cellView.model.remove();
            cellView = null;
            rotateLabels([links[i]],second_paper);
            break;
        }
    }
}
```

Figure 28: Code Example Showing the Label-Link Dropping

5.10.4. Text editing

Another important feature of the visualization tool is the direct editing of text elements. Triggered by a double-click-event on the text area, the user gets the possibility to change the text inside the field dynamically. The written text space arrangement algorithm checks the space available for the certain text area and rearranges the text with the help of:

- Font-size changes (minimal font-size is 12pt)
- Line Breaks
- Arrangement of words per line

With every change of the inserted text, the rearrangement algorithm is called (“oninput-event” fired). Figure 29 shows examples of text arrangements within different sizes of text areas.

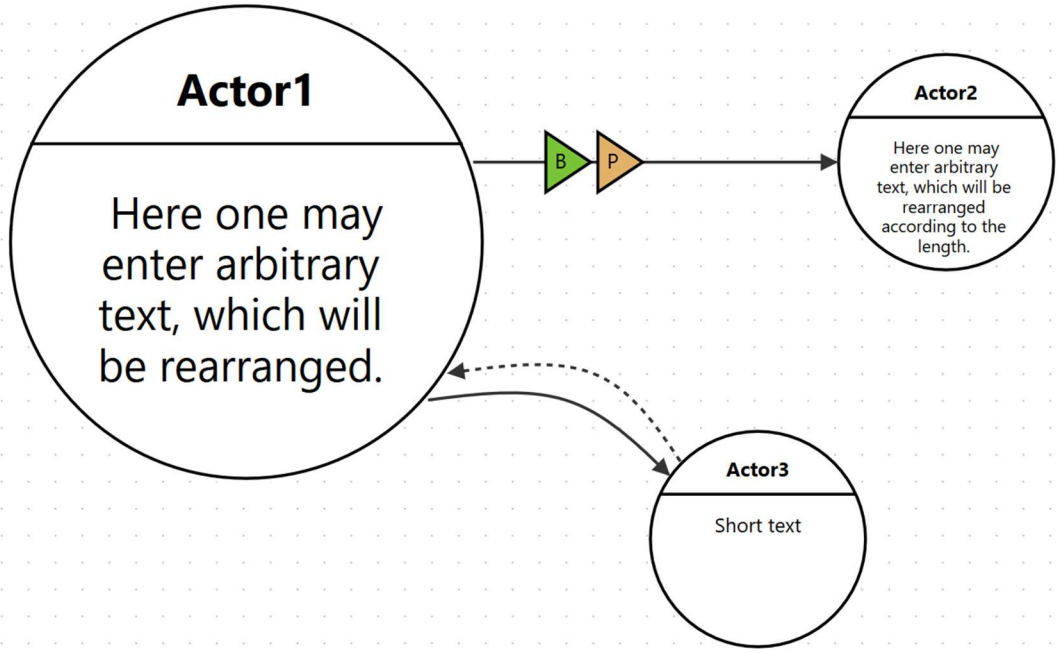

Figure 29: Example of Text Arrangement

5.10.5. Layer Switching and Layer Selection

One of the most promising and important features of the newly implemented visualization tool is the possibility to instantly switch between different layers. This allows the user to interactively design one and the same graph from different views, generating in fact multiple graphs at the same time. The insights generated with looking at different layers is one of the real values gained by the visualization tool.

Different layer options are offered to the user. The user can switch between the so-called actor layers, which determine the actor layout drawn on the graph. As shown in Figure 30, the user can select the displayed actor layer in a config drop-down menu. Only one selection is allowed at the same time. These layers include:

- Value exchange and resources layer
- Legal layer
- Values and needs layer

Figure 31 shows the same graph after selecting a different actor layer.

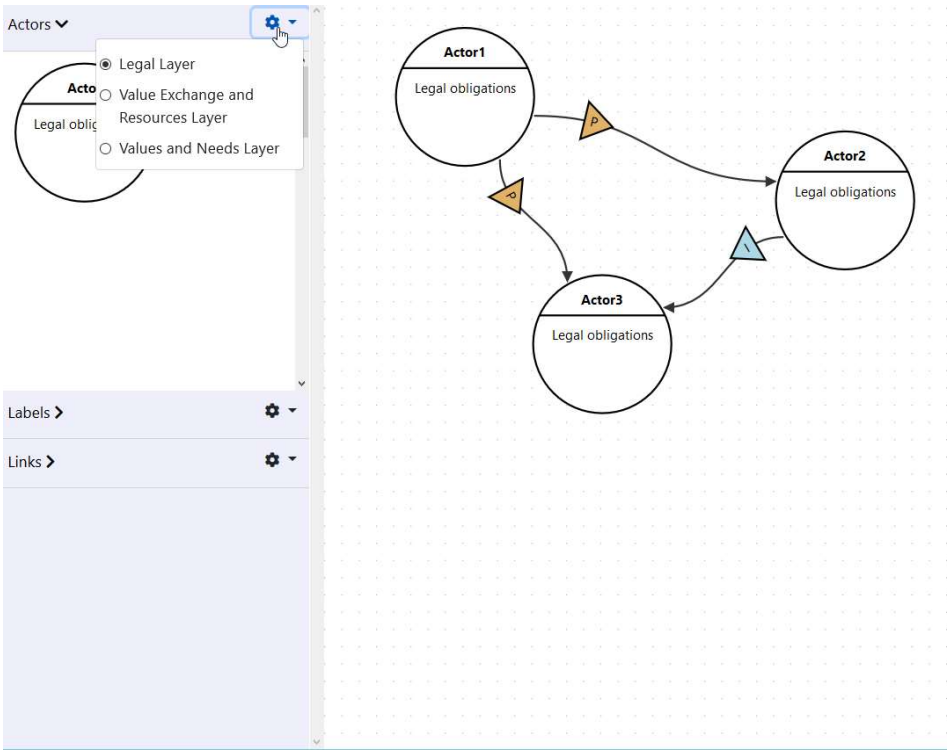

Figure 30: Actor Layer Selection Example Before Selecting a New Layer

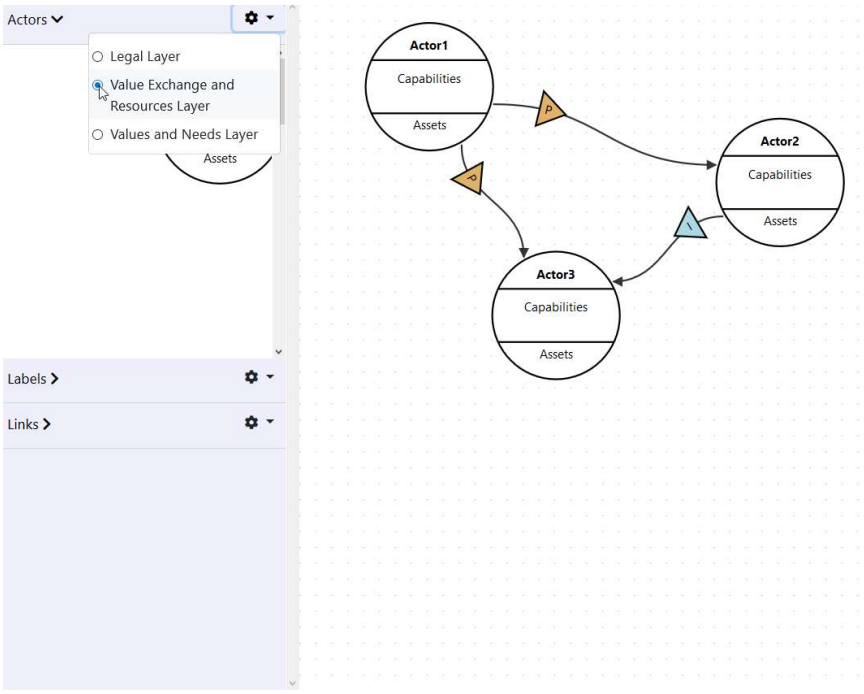

Figure 31: Actor Layer Selection Example After Selecting a New Layer

For the implementation each actor element has to save its corresponding element representations in the other layers to provide the instant switching between the layers. The called “sibling-elements” are saved by their Id in the model of each actor element.
The layer switch is triggered by the click on a radio-button field, which determines the selected layer. According to the selected layer, the actor elements’ siblings are used to find the corresponding sibling elements, which must be shown. Finally, all links are redirected to the shown sibling elements. All other actor elements, which have a different layer than the selected, can be hidden. The code example in Figure 32 shows the showing/hiding of actor elements as well as redirection of links according to the sibling ids.

```
//function for hidding/showing checked layer actors
function checkboxFunctionActors(text){
    var elements_draft = graph1Draft.getElements();
    for(var i = 0 ; i < elements_draft.length;i++){
        if(elements_draft[i].attributes.type.toLowerCase().includes(text)){
            elements_draft[i].attr("./visibility","visible");
        }else {
            elements_draft[i].attr("./visibility","hidden");
        }
    }

    var elements = jointjs_graph_main.getElements();
    for(var i = 0 ; i < elements.length;i++){
        //labels get handled by checkboxes!
        if(elements[i].attributes.type.toLowerCase().includes("labels")){
            continue;
        }
        if(elements[i].attributes.type.toLowerCase().includes(text)){
            elements[i].attr('./visibility','visible')

        }else {
            elements[i].attr("./visibility","hidden");
        }

        //filter out those siblings which are visible now and set target or
        //source to their cells according to their saved ids
        var links_outbound = jointjs_graph_main.getConnectedLinks(
        elements[i],{outbound:true});
        for(var j = 0; j < links_outbound.length;j++){
            links_outbound[j].source(jointjs_graph_main.
            getCell(elements[i].attr("./siblings").
            filter(element => jointjs_graph_main.getCell(
            element).attr("./visibility") == "visible")[0]));
        }
        var links_inbound = jointjs_graph_main.getConnectedLinks(
        elements[i],{inbound:true});
        for(var j = 0; j < links_inbound.length;j++){
            links_inbound[j].target(jointjs_graph_main.
            getCell(elements[i].attr("./siblings").
            filter(element => jointjs_graph_main.getCell(
            element).attr("./visibility") == "visible")[0]));
        }
    }
}
```

Figure 32: Code Example of Actor Layer Selection

Additionally, the user can select the layers for the displayed labels and annotations. The above-mentioned layers are hereby expanded by the dynamics and motivation layer. The user can select a number of different layers, which is between zero and four. According to the selected layers the added labels and annotations get shown/hidden accordingly. For example, if the user selects the legal layer, only the annotations concerning legal issues are shown in the left side menu, as illustrated in Figure 33. Figure 34 shows the same graph with additional labels shown by selecting more checkboxes.

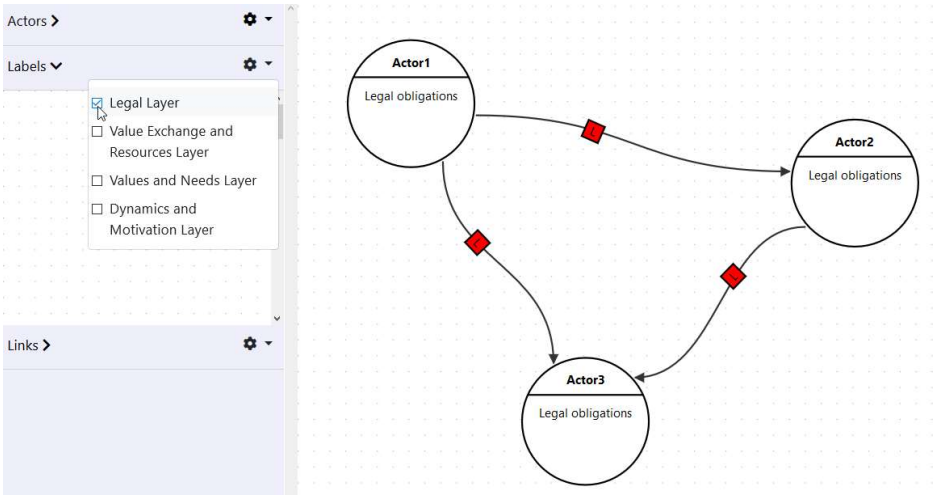

Figure 33: Label and Annotation Selection Example

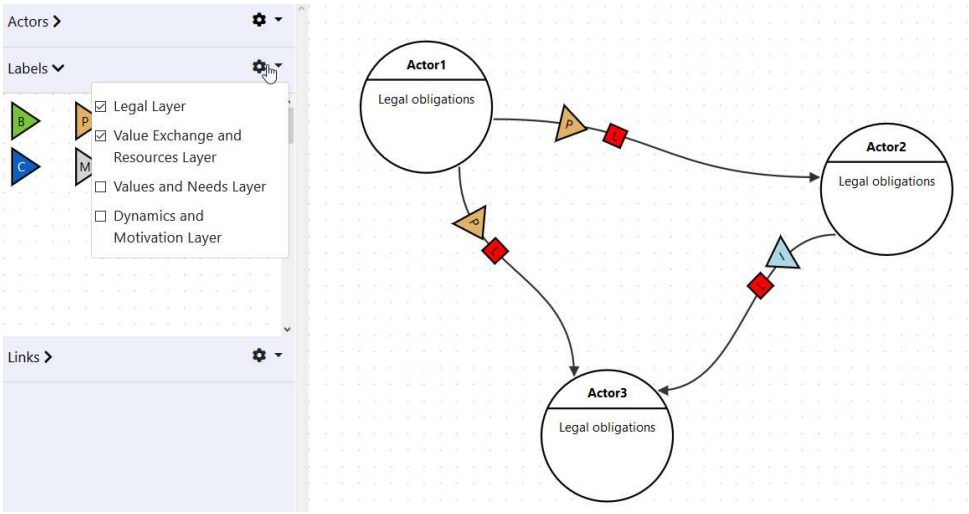

Figure 34: Label and Annotation Selection Example 2

5.11. Future Work

The current version of the visualization tool is a well-established project, which enables a fluent creation of the value network model described in Vorraber (2019). The workflows are very smooth and the key aspect of creating and analyzing value networks from different viewpoints is supported.
As in nearly every project, there is always space for improvement. Those improvements exceed the fixed scope at the beginning of the project and are therefore planned to be implemented in future project extensions.
This chapter will give a short overview of future features, which may be attractive to include them in the implementation.

5.11.1. Future Requirements

Requirements can be identified for some exploratory future implementation parts. The described requirements in Table 9 to Table 11 should give an overview of some extension examples where requirements can already be defined. There may be some additional extensions described in section 0, which are at the current point of time very vague and not precisely describable.

Table 9: Future Security Requirements

Requirement ID	Short Name	Description
S1	Availability	The System shall be available 24 hours a day, seven days a week. A downtime of one day per year is acceptable.
S2	Data Security	The system shall consider data security tasks to protect user-related data saved on server.

Table 10: Future Legal Requirements

Requirement ID	Short Name	Description
L3	DSGVO	The system shall stay compliant with the DSGVO in Europe. In case person related data will be stored due to future extensions, the DSGVO requirements shall be met.

Table 11: Future Functional Requirements

Requirement ID	Short Name	Description
F21	User Registration	The system shall allow the user to register and create himself/herself a user account. A unique nickname and a password shall be selected.
F22	Log In	The user shall be able to log in into her/his created user account by providing the nickname and the corresponding password.
F23	Session Handling	The created graphs shall be linked to the user account when logged in. Changes in the graphs are automatically saved on the server-side.
F24	Personal Working Space	When logged in, the user shall see her/his previous created graphs which were linked to her/his user account.
F25	Printing Function	The user shall be able to print her/his current drawn graph directly on her/his printer.
F26	Dynamic Page Update	The system shall support dynamic page update, no reload of the whole page should be necessary.
F27	Individual Layer Creation	The system shall support individual layer creation by the user. The user shall be able to select the actor's section number as well as the content of each section.

5.11.2. Future Functionalities (Extensions)

For the purpose of further improvements, some basic extensions have been thought of which might be useful in extended use-cases.
The first step for improvement could be the introduction of a server-side implementation for user handling and database storage. As already discussed in the section of technologies, a working server implementation has to be selected as well as a suiting database. The server-side should take care of the following activities:

- User registrations
- Log in/Log out (session handling)
- Graph storages

The server-side implementation would clearly bring the advantage of local file independency, in fact linking the created graphs to a certain user, which will then be available after login. It would also enable saving further configurations of the system per user and make the whole system a little bit more individual.
The server-side must be secured in an appropriate way as user-data is saved on the database-server. This causes administrative overheads and efforts.
The client-side functions as already implemented should not be touched and stay included in further releases. The lightweight, easy and fast client-side application is often a desired requirement for certain users.

Extension by individual elements can also be a further field of implementation, whereas the users are provided with an input mask, which allows customized declaration of individual components within a certain limit. This would offer the users a big potential if certain aspects of the included value network notation change over time or if different aspects have to be considered. To provide such an extension, further stakeholder analysis has to be done to get a deeper insight into the needs and expectations for such innovation processes.

Another potential use-case could be using the graph model as a starting point for simulation tasks to e.g. identify influencing powers of stakeholders concerning a certain viewpoint in a specific network. The simulation results could be graphically highlighted in the graph view. This extension would provide the value network visualization tool with the ability to analyze the underlying model in a deeper way and would lead to a more detailed result. The tool would then not solely be on the visualization side, but also act as a kind of data analyzation tool. The way how such a simulation can be implemented and integrated into the visual creation of the graph, was not part of this project and is therefore not included in this version. Nevertheless, it is an interesting point of thinking which shows the great potential of the created tool. The baselines (e.g. strict separation of view and data in the MVC pattern) already exist and offer a great possibility to build on for the suggested simulation process.

6. Conclusion

It has been investigated in detail, how important a structured engineering process is. Starting with the systems engineering approach, where complete systems are modelled using methods, like the top-down approach, different problems and solutions have been discussed to describe the dynamics of a system properly. Special engineering techniques, like service design thinking, have been described to highlight the focus on the stakeholders' interests and their visualization. For examining and documenting the stakeholders' needs, requirements engineering has been reviewed. The three main discussed theory chapters build a common understanding of how important the stakeholder centered, systematic design and analysis during an engineering process is. For this reason, a visualization tool has been developed to enable a value-network stakeholder map. It offers a detailed stakeholder analysis, using various viewpoints on one and the same influencing graph.

The implemented visualization tool provides great opportunities for a fluent value network graph generation. It is easy and quick to use, as no registration nor special software are required. The implemented functionalities guarantee a fast and easy creation of a value network, which can be analyzed using different viewpoints, while showing and hiding different types of elements.
It is, like with every other software product, possible to implement extensions if needed by the main users of the system. Future directions of implementations are already in the mindset and will be headed for in the near future.
Hopefully, the implemented tool helps to further strengthen the position of value networks in research and gives one or the other new user the possibility to easily get used to the topic by starting to work with it.

6.1. References

Allgemeiner Umdruck Nr. 250/1 (1997), *Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell.: Teil 1: Regelungsteil.*

Allgemeiner Umdruck Nr. 250/3 (1997), *Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell.: Teil 3: Handbuchsammlung.*

Alter Way (2019), "Wampserver. A Windows Web Development Environment", available at: http://www.wampserver.com/en/ (accessed 17 April 2019).

Apache Friends (2019a), "Windows Frequently Asked Questions", available at: https://www.apachefriends.org/faq_windows.html (accessed 30 April 2019).

Apache Friends (2019b), "XAMPP Apache + MariaDB + PHP + Perl", available at: https://www.apachefriends.org/index.html (accessed 17 April 2019).

Apache Software Foundation (2019), "What is Apache?", available at: https://wiki.apache.org/httpd/FAQ#What_is_Apache.3F (accessed 17 April 2019).

Beck, K. (1999), *Extreme Programming Explained: Embrace Change,* First Edition, Addison-Wesley, Boston.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. (2001), "Manifesto for Agile Software Development", available at: http://agilemanifesto.org/ (accessed 26 February 2019).

Biem, A. and Caswell, N. (2008), "A Value Network Model for Strategic Analysis", in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008), Waikoloa, HI, USA, 07.01.2008 - 10.01.2008*, Institute of Electrical and Electronics Engineers, New York, pp. 1–7.

Bootstrap (2019), "Bootstrap", available at: https://getbootstrap.com/docs/4.3/getting-started/download/ (accessed 16 April 2019).

Breuer, H. and Lüdeke-Freund, F. (2017), "Values-Based Network and Business Model Innovation", *International Journal of Innovation Management*, Vol. 21 No. 03, p. 1750028.

Bryson, J.M. (1995), *Strategic Planning for Public and Nonprofit Organizations: A Guide for Strengthening and Sustaining Organizational Achievement, The Jossey-Bass nonprofit sector series,* Rev. ed., 3rd print, Jossey-Bass, San Francisco, Calif.

Bryson, J.M. (2003), "What To Do When Stakeholders Matter. A Guide to Stakeholder Identification and Analysis Techniques", A paper presented at the National Public Management Research Conference October 2003, available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.514.2874&rep=rep1&type=pdf (accessed 2 March 2019).

Cameron, E. and Green, M. (2009), *Making Sense of Change Management: A complete guide to the models, tools & techniques of organizational change,* 2. ed., Kogan Page, London u.a.

client.IO (2019a), “joint.js”, available at: https://www.jointjs.com/opensource#Download-JointJS (accessed 16 April 2019).

client.IO (2019b), “JointJS Documentation”, available at: https://resources.jointjs.com/docs/jointjs/v2.2/joint.html (accessed 16 April 2019).

Curry, E. and Grace, P. (2008), “Flexible Self-Management Using the Model-View-Controller Pattern”, *IEEE Software*, Vol. 25 No. 3, pp. 84–90.

Easterbrook, S. (2007), “Scale Changes Everything: Understanding the Requirements for Systems of Systems”, in *Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007: ICCBSS '07 ; Feb. 26, 2007 - March 2, 2007, Banff, Alberta, Canada, Banff, AB, Canada, 2/26/2007 - 3/2/2007*, Institute of Electrical and Electronics Engineers Computer Society, Los Alamitos, Calif. [u.a.], p. 16.

Eden, C. and Ackermann, F. (2004), *Making Strategy: The Journey of Strategic Management,* Reprinted., Sage, London.

Freeman, R.E. (2010), *Strategic management: A stakeholder approach*, Cambridge University Press, Cambridge.

Gausemeier, J. and Moehringer, S. (2002), “VDI 2206- A New Guideline for the Design of Mechatronic Systems”, *IFAC Proceedings Volumes*, Vol. 35 No. 2, pp. 785–790.

Gordijn, J. (2002), “Value-based Requirements Engineering. Exploring Innovative e-Commerce Ideas”, Faculteit der Exacte Wetenschappen, Universiteit De Boelelaan, Amsterdam, 2002.

Haberfellner, R., de Weck, O., Fricke, E. and Vössner, S. (2018), *Systems Engineering: Grundlagen und Anwendung,* 14., überarb. Aufl., Orell Füssli Verlag, Zürich.

Hall, A.D. (1962), *A Methodology for Systems Engineering*, Van Nostrand, Princeton N.J. u.a.

Institute of Electrical and Electronics Engineers (1990), *IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990*, Institute of Electrical and Electronics Engineers, New York.

International Organization for Standardization (2018), “ISO 10303-233:2012. Industrial automation systems and integration -- Product data representation and exchange -- Part 233: Application protocol: Systems engineering”, available at: https://www.iso.org/standard/55257.html (accessed 29 April 2019).

International Organization for Standardization, International Electrotechnical Commission and Institute of Electrical and Electronics Engineers (2018), *ISO/IEC/IEEE 29148: 2018(E): ISO/IEC/IEEE International Standard - Systems and software engineering --*

Life cycle processes -- Requirements engineering, International standard, ISO/IEC/IEEE 29148-2018, Institute of Electrical and Electronics Engineers, New York.

International Organization for Standardization, International Electrotechnical Commission, Institute of Electrical and Electronics Engineers, IEEE-SA Standards Board and IEEE Xplore (Online service) (2015), *Systems and software engineering-- System life cycle processes: Ingénierie des systèmes et du logiciel-- Processus du cycle de vie du système, International standard*, ISO/IEC/IEEE15288:2015(E), Institute of Electrical and Electronics Engineers, New York.

Joyent Inc. (2019), “About Node.js”, available at: https://nodejs.org/en/about/ (accessed 17 April 2019).

Kelman, H.C. (1961), “Processes of Opinion Change”, *Public Opinion Quarterly*, Vol. 25 No. 1, pp. 57–78.

Krasner, G.E. and Pope, S.T. (1988), “A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk80 System”, *Journal of Object-oriented Programming - JOOP*, Vol. 1, pp. 1–34.

Kruchten, P. (2007), *The Rational Unified Process: An introduction, The Addison-Wesley object technology series,* 3. ed., 7. printing, Addison-Wesley, Upper Saddler River, NJ.

Luo, Q. (2011), “User-Oriented Service Design and Innovation”, in *2011 International Conference of Information Technology, Computer Engineering and Management Sciences ICM 2011: Proceedings 24-25 September 2011, Nanjing, Jiangsu, China, Nanjing, Jiangsu, China, 9/24/2011 - 9/25/2011*, Institute of Electrical and Electronics Engineers Computer Society, Los Alamitos, Calif., pp. 51–54.

MariaDB Foundation (2019a), “MariaDB About”, available at: https://mariadb.org/about/ (accessed 17 April 2019).

MariaDB Foundation (2019b), “MariaDB versus MySQL - Features”, available at: https://mariadb.com/kb/en/library/mariadb-vs-mysql-features/ (accessed 17 April 2019).

Object Management Group, I. (2017), “OMG Unified Modeling Language. Version 2.5.1”, available at: https://www.omg.org/spec/UML/ (accessed 26 February 2019).

Object Management Group, I. (2019a), “MBSE Wiki”, available at: http://omgwiki.org/MBSE/doku.php (accessed 20 April 2019).

Object Management Group, I. (2019b), “What is SYSML?”, available at: http://www.omgsysml.org/what-is-sysml.htm (accessed 21 April 2019).

Oracle Corporation (2019), “MySQL”, available at: https://www.mysql.com/de/ (accessed 17 April 2019).

Osterwalder, A. and Pigneur, Y. (2010), *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*, Wiley, Hoboken, NJ.

Partsch, H.A. (2010), *Requirements-Engineering systematisch: Modellbildung für softwaregestützte Systeme,* 2. Auflage, Springer Berlin Heidelberg, Berlin/Heidelberg.

Porter, L.W. and Lawler, E.E. (1968), *Managerial Attitudes and Performance, The Irwin-Dorsey series in behavioral science,* 1. print, Richard D. Irwin, Homewood Illinois.

Refsnes Data (2019), “Browser Statistics”, available at: https://www.w3schools.com/browsers/default.asp (accessed 17 April 2019).

Ruparelia, N.B. (2010), “Software development lifecycle models”, *ACM SIGSOFT Software Engineering Notes*, Vol. 35 No. 3, pp. 8–13.

Rupp, C. (2004), *Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis. 3., neu bearb. Aufl*, Hanser, München.

Schwaber, K. and Beedle, M. (2002), *Agile Software Development with Scrum, Series in agile software development*, Prentice Hall, Upper Saddle River, NJ.

Sommerville, I. and Sawyer, P. (1997), *Requirements engineering: A good practice guide,* Reprinted., Wiley, Chichester.

Stickdorn, M., Hormess, M.E., Lawrence, A. and Schneider, J. (2018), *This is Service Design Doing: Applying Service Design Thinking in the Real World A Practitioners' Handbook,* First edition, O'Reilly Media, Inc, Sebastopol, CA.

Stickdorn, M. and Schneider, J. (Eds.) (2013), *This is Service Design Thinking: Basics, Tools, Cases,* 3. printing, paperback edition, BIS Publ, Amsterdam.

Tandemic (2019), “Social Business Model Canvas”, available at: http://www.socialbusinessmodelcanvas.com/ (accessed 16 April 2019).

Turner, R., Pyster, A. and Pennotti, M. (2009), “Developing and Validating a Framework for Integrating Systems and Software Engineering”, in *2009 3rd Annual IEEE Systems Conference, Vancouver, BC, Canada, 3/23/2009 - 3/26/2009*, Institute of Electrical and Electronics Engineers, New York, pp. 407–412.

Twisted Matrix Labs (2019), “What is Twisted?”, available at: https://twistedmatrix.com/trac/ (accessed 17 April 2019).

Vorraber, W. (2019), “A Hybrid Service Analysis and Engineering Framework for New Business Models”, internal paper, under revision.

Vorraber, W., Lichtenegger, G., Brugger, J., Gojmerac, I., Egly, M., Panzenböck, K., Exner, E., Aschbacher, H., Christian, M. and Voessner, S. (2016), “Designing Information Systems to Facilitate Civil-Military Cooperation in Disaster Management”, *International Journal of Distributed Systems and Technologies*, Vol. 7 No. 4, pp. 22–40.

Vorraber, W., Mueller, M., Voessner, S. and Slany, W. (2019), “Analyzing and Managing Complex Software Ecosystems: A Framework to Understand Value in Information Systems”, *IEEE Software*, Vol. 36 No. 3, pp. 55–60.

Vorraber, W. and Vössner, S. (2011), “Modeling Endogenous Motivation and Exogenous Influences in Value”, *Journal of Convergence Information Technology*, Vol. 6 No. 8, pp. 356–363.

Vroom, V.H. (1967), *Work and Motivation,* 3rd printing, Wiley, New York, NY.