Inclusive Systems Engineering

Dissertation

Thesis approved by the Department of Computer Science RPTU University Kaiserslautern-Landau for the award of the Doctoral Degree Doctor of Engineering (Dr.-Ing.)

to

Šandor Dalecke

Date of Defense	22.09.2025
Dean	Prof. Dr. Christoph Garth
Reviewer	Prof. Dr. Christoph Grimm
	Prof. Dr. Nil Hande Ergin

DE-386

Abstract

New systems are becoming ever more complex and interconnected. To develop these systems the expertise and collaboration of many stakeholders, including domain experts and systems engineers, is required. Usually these domain experts are not familiar with systems engineering, which is especially true with the adoption of the model-based systems engineering.
In order to identify systems engineering issues and importance of requested features this thesis presents data gathered from two surveys which have been conducted. The first survey had participants who were students of the RPTU Kaiserslautern with no prior systems engineering experience. This survey was used to review SysML, SysML v2 and SysMD on their entry bar and as how intuitive these modelling languages are experienced. The key insights of this survey were the need for simplification of modelling syntax and the clear preference for graphical model representations for visualisation and understanding a model and the preference towards a textual representation in order to create and extend a model. The second survey was shared with members of the Object Management Group and the International Council on Systems Engineering. The goal of this survey was to identify key issues of systems engineering. The study had participants from different levels of system engineering experience, making it possible to identify indicative trends on different needs depending on the experience. As expected the professional level participants perceive systems engineering as easier to get into than beginners. This survey supports the results from other studies while focusing on an individual level and participants from a variety of systems engineering expertise. In addition, the survey indicates a need for simplification of syntax of modelling languages and tools for model verification and validation.
To reduce the entry bar of systems engineering and enhance collaboration the notion of inclusive systems engineering is introduced as a means to make systems engineering more accessible. This extends systems engineering towards being more accessible for domain experts by focusing on a user-centric design and proposes the adoption of features aimed at user guidance, providing tool accessibility and features focused on enhancing the accessibility of the systems engineering process. The notion of user guidance as a means to enhance systems engineering is discussed in more detail, giving an introduction into human decision making and reviewing common user influencing techniques.

Acknowledgement

First of all I want to thank my wife for always being at my side, supporting my decision to stay abroad, taking on the offer to do this thesis and never doubting my abilities, no matter how much self-doubt I had. I want to thank my family for providing me the environment to grow and foster my curiosity. Furthermore, I want to thank Prof. Christoph Grimm for offering me the opportunity to work in his research group and pursue my areas of interest even when they did not fully align with his own research interests. Last, but not least I want to thank Prof. Nil Hande Ergin for agreeing to review this work and Prof. Randi Karlsen, who provided the foundations of this work by sparking my interest in nudging and related topics.

December 5, 2025, Šandor Dalecke

Contents

List of Figures

Chapter 1

Introduction

1.1. Motivation

New systems get ever more complex which requires ever more domain expert participation in systems engineering and poses higher risks of misunderstanding intricacies of a system parts. This requires domain knowledge of a multitude of domains from systems engineers. While AI techniques are explored which can mitigate the need for domain knowledge by acting as a domain expert [36] it is problematic to rely on AI exclusively.
A more entwined systems engineering process is the current approach, however this is very time and cost intensive. Additionally, communication is a process in which information inevitably gets lost, leading to incomplete or incorrect features being developed.
A different approach is to train domain experts in systems engineering. However, systems engineering requires knowledge of both syntax and semantic unique to systems engineering. Many of these particularities are based in computer science, a discipline comparatively young and thus not widely known by domain experts of other domains. Training domain experts in systems engineering is therefore time and cost intensive as well.
This poses a risk of newer systems engineering methodologies, like model-based systems engineering, only being adapted slowly making new systems take more time and money to create. The adoption of such a new methodology has to overcome challenges such as aversion towards up-front investment and the expected effort required compared to the anticipated benefits. Additionally, such a methodology has to be adopted by domain experts as well, which can be difficult depending on the tools and languages specific to their domain they have used before. Most studies on this are on an organisational level, giving important insights into the organisational adoption but lacking insight on the issues and features important to individuals with different level of systems engineering expertise. These studies focus on the adoption of model-based systems engineering from professional systems engineers, lacking insight on effects hindering adoption for beginners of systems engineering or students with no prior experience in systems engineering.

To gather insights into issues and features important to systems engineers of different level a survey approach can be used. Surveys provide a means to gather insights in a structured and asynchronous way, reducing the need to spend resources on interviews, creating focus groups or observe systems engineers in their work environment. The risk of sampling bias and under representation of certain domains needs to be considered when evaluating such a survey approach.
The insights collected from surveys provide tentative evidence of needs and issues to provide an extension to the systems engineering process which focuses on integrating domain experts and providing a lower entry barrier. These insights are complemented with insights into human decision making processes and user guidance techniques to guide new users towards adapting systems engineering as well as guiding experienced users towards adopting new approaches such as model-based systems engineering and provide more comprehensive but easy to understand solutions to reduce miscommunication.

1.2. Contribution

My work introduces inclusive systems engineering as a framework to make systems engineering more approachable. This framework focuses on user guidance and the reduction of complexity of syntax. Implementing this framework reduces overhead for domain experts to learn the model-based systems engineering approach, leaving more time to focus on semantics to provide semantically comprehensive and consistent models.
Inclusive systems engineering is based on leveraging theoretical knowledge human decision making, lessons learned from software development and data collected from two surveys aimed at recognising current systems engineering challenges and requested features.
The key contributions of this work are:

- A discussion on user guidance as a means to make systems engineering more accessible with a focus on why the methods work by giving a brief introduction into human decision making.

- A survey with students from the RPTU Kaiserslautern-Landau reviewing the intuitive use of SysML, SysML v2 and SysMD and the preference of textual versus graphical representations of models.

- A survey with members from OMG and INCOSE focused on issues present in systems engineering and model-based systems engineering. This survey gives insight on a personal level of systems engineers with professional experience, as well as intermediate, beginner and no experience in systems engineering.

- The proposal of inclusive systems engineering, aimed to make systems engineering more accessible for stakeholders, and domain-experts, in particular with little to none prior systems engineering experience. This

theoretical proposal includes a definition and description of features to support inclusive systems engineering. These features are structured as features specific to user guidance, which includes nudging, and recommender systems among others, features specific to process accessibility, such as support for graphical and textual representations of models, and features focused on tool accessibility, such as providing a version control system or facilitating notebook programming. This framework provides an overview of techniques, features and attributes to keep in mind when designing and implementing new systems engineering tools or reviewing current tools for future improvement.

1.3. Thesis Structure

Chapter 2 discusses the current problems faced in systems engineering and introduces document-based systems engineering, electronic document management system and model-based systems engineering followed by a brief discussion of surveys on systems engineering and the adoption of model-based systems engineering. An overview of systems engineering languages, such as UML, SysML and OPM is given with a greater focus on KerML, SysML v2 and SysMD. To complement this technical state of the art, chapter 3 an introduction into human decision making theory and common ways to influence and guide user behaviour. Chapter 4 reviews the data gathered in a preliminary survey with students from the RPTU Kaiserslautern-Landau and is followed by another survey with participants from OMG and INCOSE. The insights gathered from this chapter provide a basis for inclusive systems engineering which is discussed in chapter 5. This chapter introduces inclusive systems engineering by giving a formal definition and providing criteria to evaluate a given tool or language. Inclusive systems engineering is based upon three pillars to provide more accessibility into the process, which is user guidance, process accessibility and tool accessibility. For each a number of features are given that enhance accessibility. Chapter 6 gives the conclusion of my thesis and discusses future research opportunities.

Chapter 2

Systems Engineering State of the Art

Contents

2.1. Problem Definition

Systems are becoming ever more complex, connecting multiple domains and having to respond to new trends and requirements ever faster. New systems rely on domain experts to provide comprehensive requirements and their expertise to be successful. However, communication between systems engineers, stakeholders and domain experts is a bottleneck in many projects, leading to increased resource costs [85]. This issue is getting more severe the more stakeholders are involved. This is further amplified by systems engineers being required to have some domain expertise to be able to translate the domain expert input into suitable systems engineering requirements to fulfill these requirements.

The current move towards model-based systems engineering to realize complex systems aims to solve issues of complex systems such as scalability issues to manage extremely large systems and to provide a common basis for requirements, design and analysis to improve traceability and communication. However, adoption of model-based systems engineering requires redefining of traditional workflows and introduces it's own issues by introducing issues such as complex syntax, when SysML v2 textual representations are used, and complex semantics following programming conventions not necessarily well known for beginners and domain experts.
To address this, my work introduces inclusive systems engineering, an extension to traditional systems engineering and model-based systems engineering, which focuses on making the approach more accessible to non-experts by providing features to be followed in three main aspects.
1) By enhancing accessibility of the process by means of providing simplified syntax to lower the entry barrier, supporting incomplete models and related aspects.
2) By providing features for tools, ensuring standard compatibility or providing means for automatic code generation.
3) Putting a focus on user guidance to increase motivation and reduce workload by utilising concepts such as nudging or providing means to abstract unnecessary information.
This aims to provide domain experts with the ability to create comprehensive requirements and rudimentary system definitions themselves and to communicate solutions created in a domain specific language or tool easier. These requirements, definitions and part solutions can be used by systems engineers to create complex systems with a reduced risk of misinterpretation.

2.2. Systems Engineering

INCOSE defines systems engineering [67] as *"[...] a transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological and management methods."* Systems engineering has changed a lot in the last twenty years. Dealing with ever more complex systems, the need to change functionality during and after development, new policies or needs. This development results in new challenges for software and especially systems engineering. Communication between stakeholders is often a bottleneck in these instances [85]. This is especially true for more task specific, often highly complex, systems, tailored towards specific domains with unique requirements.

2.2.1. Document-Based Systems Engineering

Document-based systems engineering (DBSE) is the traditional method of systems engineering which focuses heavily on textual documentation to capture, communicate and managing system requirements, designs, and processes. DBSE typically follows a linear or waterfall model in which each step is com-

pleted before moving on to the next development phase. The waterfall model has been updated and replaced by the Vee-Model as shown in figure 2.1.

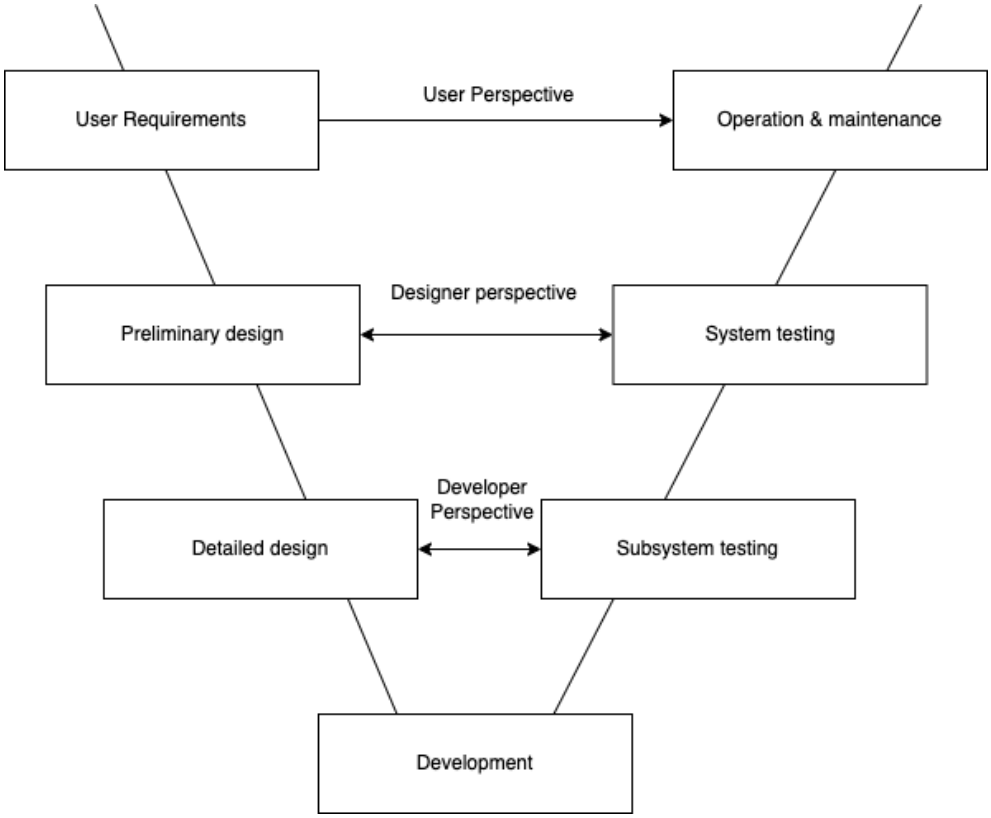

Figure 2.1.: *The "vee" developmental model according to Blanchard [20].*

The main source of truth is a collection of documentation consisting of requirements specification, design documents and verification and validation plans, among others. These documents are maintained throughout the systems engineering process and function as communication assets [20]. However, the increasing complexity of modern systems pose multiple challenges for DBSE, mainly requiring a high amount of flexibility to adapt to new challenges. The inherent linear process limits this flexibility which leads to increased costs and project delays when modifications are necessary. In addition, the early development phases often lack formal methods for verification and validation to ensure the specifications meet the requirements before implementation, which can lead to significant issues, especially considering the high complexity of today's systems. Furthermore, a collection of documents, which are usually interdependent, as source of truth can be cumbersome to manage and require high maintenance to keep up to date, which hinders collaboration, especially in concurrent engineering environments [4]. The document-centric approach often makes use of fragmented documentation, providing very limited traceability for design decisions.

2.2.2. Electronic Document Management System

Electronic document management system (EDMS) modernizes DBSE by digitizing the document workflows to improve accessibility and standardize the process [99]. It is still document-centric, using documents as the source of truth but addresses some of the limitations of traditional DBSE. Digitization reduces the physical storage needs an removes manual retrieval cost. The real-time information sharing capabilities of EDMS facilitate enhanced collaboration across teams and stakeholders [1]. By using indexing and version control systems a partial automation can be achieved. EDMS adepts DBSE, still being fundamentally document-centric, by taking advantage of the capabilities of digital documents. However, being document-centric with fragmented documentation it only provides limited traceability. EDMS can pose high costs and encounter stakeholder resistance while leaving gaps in addressing system-level complexity, providing mostly advantages to domains relying on structured documents such as construction projects [19] and the public sector [1].

2.2.3. Model-Based Systems Engineering

Model-based systems engineering (MBSE) replaces documents with interconnected models to enable a dynamic, data-driven representation of systems. According to INCOSE MBSE is the formalized application of modeling to support systems requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases [67].
MBSE also intends to reduce wasted time and money by implementing wrong or useless features [85]. The document-based approach is hard to manage with increasing complexity of systems. MBSE moves away from this towards an iterative approach with models as the key sources of truth which are more scalable and manageable for larger, more complex systems. The model-based approach should create clear, connected models which can be shared among shareholders, achieving better communication and reducing the risk of useless features due to miscommunication. These models needs to be updated in case of major changes, but is intended to be checked against throughout the whole engineering process.
However, large-scale models are hard to create perfectly, needing to be revised and updated continuously, which is time-consuming and can have large repercussions [26] [24]. In addition, policy changes or technological advances can affect the validity of a model throughout the development.
Furthermore, MBSE is still a rather new approach, lacking a wide-spread standardized methodology. Currently the OMG works to resolve this issue by developing the Kernal Modeling Language (KerML) [59] to provide a very fundamental, easily extendable language definition. Additionally, a more sophisticated language, introducing many important extensions, called Systems Modeling Language version 2 (SysML v2) [119] is developed simultaneously.
Estefan discusses the mathematical foundations described by Wymore [133],

as the foundations of MBSE are supposed to be grounded in the foundation of mathematics [68], and gives a comprehensive discussion of MBSE methodologies [49].

2.2.4. Domain Expert Integration

Domain experts play an important role in systems engineering, ensuring that technical solutions fulfill real-world requirements and constraints. In early development domain experts provide essential input by providing constraints and formulating requirements the future system needs to align with. These can be operational needs and industry-specific standards. During system design and modelling domain experts collaborate with systems engineers to validate models. Their domain expertise ensures that necessary abstractions don't misalign with domain-specific behaviours and interactions. Domain experts are integral to validate and verify developed models and prototypes as well as making sure training datasets are curated in case of AI training. With systems engineering being a continuous process, iterating upon systems by providing new features and responding to new requirements the need to integrate domain experts is continuous as well.

2.2.5. MBSE adoption

The adoption of MBSE in industry has been subject of studies in recent years. Kuhn, Murphy and Thompson [80] provided early evidence on the challenges and benefits of MBSE adoption. Their introduction of forces as indicators of innate complexity and friction as accidental complexity. Identified forces, such as a team working on different versions of an artifact, have one or more associated frictions, such as tools offering version control being limited in their abilities or not providing the version comparison in the desired way. The need for traceability in incremental releases is a force with the corresponding friction of inefficient and fragile workarounds to create traceability by naming conventions.
Aranda, Damian and Borici [10] presented findings from a case study of organisation consequences after a model-driven engineering approach was adopted. They highlight the benefit of bringing the development closer to the the subject matter and the domain experts, however they also point out the issue of models not being adopted by all groups, especially by domain experts who are unlikely to have software development training. They also highlight issues in adoption by cultural hindrance and by an aversion to adopting a new work methodology as it can disrupt team-work and requires additional training.
Vogelsang et. al. [128] further supported these findings by identifying hindering forces, such as different types of inertia and anxiety. The types of inertia include tooling inertia, not adopting new MBSE tools over the already existing in-house tooling environment, and context inertia, not believing the new methodology fits the current situation. The types of anxiety include issues such as not seeing the benefit of adopting the new methodology considering the resources spent for adoption and the fear of lacking necessary skills. How-

ever, they also identified external forces fostering adoption such as the growing complexity of systems and enforcement by stakeholders who request the use of an MBSE approach. Another set of internal fostering forces have been identified as pull forces, such the value provided by early feedback on correctness or easier handling of increased complexity.
Siddique [113] provides a literature review on the benefits of adopting MBSE over the traditional DBSE approach with the focus on small satellite systems. This highlights key MBSE advantages such as adaptability to evolving requirements and enhanced collaboration for domains such as satellite systems, which are very complex and need to adapt to changing environments. This further supports the findings from the analysis of MBSE integration within the aerospace industry by Pratt and Dabkowski [103]. They identify enablers and challengers for MBSE adoption in this domain. They have identified key issues such as performance expectancy, effort expectancy, social influence and facilitating conditions and rate these as challengers or enablers according to a literature review. The greatest challenge identified there was the effort expectancy, showing the aversion towards the up-front investment to adopt MBSE, while facilitating conditions, such as a reported lack of guidelines and best practises is also functioning more as a challenger than an enabler for MBSE adoption. The greatest enabler was the performance expectancy, highlighting the advantages of being able to experiment in small scales and the overall performance increase due to MBSE benefits such as increased stakeholder collaboration.
Temper et al. [122] provide a literature review considering the MBSE perception across sectors from commercial, contractor and governmental as well as systems such as astronautical, aeronautical and infrastructure. Overall, they report a positive perception of MBSE across systems while highlighting more variance in perception by sectors. Especially contractor and commercial sectors perceived MBSE more negatively and a reluctance to implement MBSE. This supports the findings from Vogelsang et. al., highlighting an aversion to change the curren processes when perceived as "working" and being averse towards an up-front resource investment due to short-term goals. The study by Henderson et. al. [64] discusses how organizational factors affect MBSE adoption by comparing data on organizational structure with adoption metrics. They report the strongest correlations to be flexibility and interconnectedness, showing that organisations that are flexibility and interconnectedness had strong positive correlation with MBSE adoption, with more flexible and organisations with highly connected members were open to adopt MBSE easier. Size and centralisation had a negative correlation, showing large and highly centralized organisations were less open to adopt MBSE while size and centralization had a negative correlation.

2.3. Systems Engineering Languages

2.3.1. UML

The unified modeling language is the widely spread de facto standard for specifying, constructing and documenting software systems [108]. This language has been refined by the Object Management Group (OMG) and currently exists in UML version 2 which will be referred to as UML for short as version 1 is not relevant to this thesis.
UML makes use of multiple different diagram types, each emphasising only certain aspects of a system which all together provide a full and complete description of a modelled system. The following list is based on the UML 2 Toolkit by Eriksson [48] and only provides a short introduction into each diagram type

- *Use-Case Diagram*: Shows external actors and their connection to the use case, which is the functionality a system provides.
- *Class Diagram*: Shows the static structure of classes in the system. Classes are related by association, dependence, one class being a specialization of another class, or being grouped together as a package.
- *Object Diagram*: Object diagrams show the object instances of classes and provides a possible snapshot of a system's education.
- *State Machine*: Class diagrams can be complemented by state machines, showing all possible states that the objects of a class can have and what events result in state changes. The subtype of a behavioral state machine describes all details of a class's life cycle whereas the protocol state machine only focuses on the transition of states and the rules defining execution order.
- *Activity Diagram*: Provides a visualisation of a sequential flow of actions. Typically describes the activities in a general process workflow.
- *Interaction Diagrams*: UML provides three diagram types which show interaction. In addition, UML provides a timing diagram to model real-time systems.
- *Sequence Diagram*: Illustrates collaboration between objects. It shows a sequence of messages and models dynamic interaction.
- *Communication Diagram*: Shows dynamic collaboration similar to the sequence diagram. In addition it shows the objects and relationships, modelling both interaction and the context in which it takes place.
- *Interaction Overview Diagram*: Shows the main flow of interactions at a high-level to provide an easy method for reviewing the interactions from a high-level point of view.
- *Component Diagram*: Components can be source code, binaries or executable. Components contain information about the logical classes it implements and provides a mapping from logical to component view.

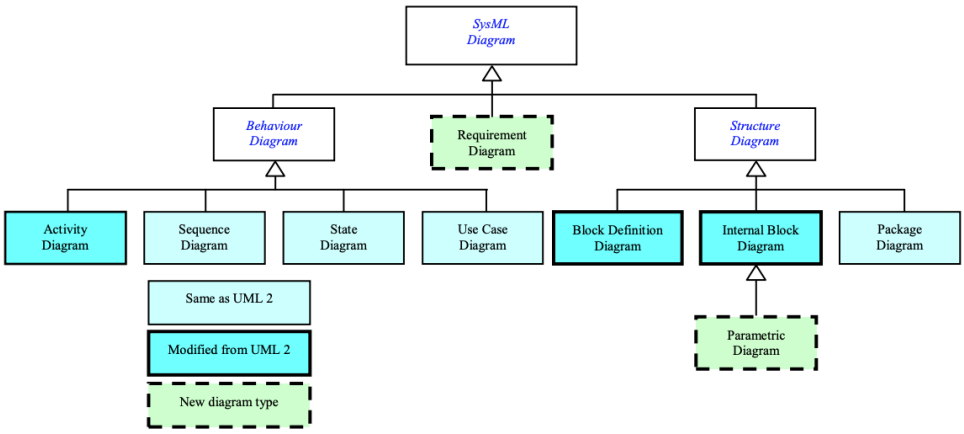

Figure 2.2.: *OMG SysML Diagram Taxonomy [63].*

- *Deployment Diagram*: Shows the physical architecture of hardware and software in a system and describes the actual physical architecture of the system after deployment.

- *Composite Structure Diagram*: Shows the participating elements and their relationships in the context of specific classifiers such as use case, object, collaboration, class or activity. The composite structure diagram can model specific collaboration of elements during runtime which can be different from information modelled in other static diagrams.

2.3.2. SysML

SysML uses a subset of UML and extends it with additional diagrams, reducing the features aimed at software development and shifting the focus towards systems modelling. This is achieved by adding diagrams to model requirements and set constraints on a system, as well as extending existing diagrams to be more suited for modelling purposes and systems engineering applications [63]. Figure 2.2 shows the different diagram types of SysML. The sequence diagram, state machine diagram, use case diagram and package diagrams are directly the same as in UML. The activity diagram, block definition diagram and internal block diagram are similar to UML but modified. Both the parametric diagram and requirement diagram are new to SysML. The following description is based on Hause and the work from Friedenthal, Moore and Steiner [55] provides a comprehensive introduction into SysML.

- *Activity Diagram*: The activity diagram is modified to introduce probabilities for the likelihood that a value will be output, leave the decision node or an object will traverse an edge. Furthermore, allocations are introduced to enable the allocation of one model element by another. These allocations can be used as precursors to more detailed specifications by providing a means for abstraction.

- *Requirement Diagram*: Shows a requirement in the form of a "shall" statement and is related to other modelling artefacts via a set of dependencies that allow for refinement, verification and other dependencies.
- *Block Definition Diagram*: Is similar to the class diagrams in UML which is used to describe relationships between blocks, such as one block being composed of other blocks.
- *Internal Block Diagram*: Is used to describe the internals of a block and provides a detailed view, whereas the block definition diagrams provide a more high-level relationship focused view.
- *Parametric Diagram*: Are used to describe constraints on properties in order to support engineering analysis and is a specialised variant of an internal block diagram. It restricts diagram elements which enables the representation of constaints.

2.3.3. OPM

Object-Process Methodology (OPM) is a holistic approach to systems modeling that integrates both the structure and behavior of systems in a unified framework proposed by Dori [46]. OPMs key difference to UML and SysML is the object-process diagram which depicts both the static structure and dynamic behaviour of a system in one single diagram. In addition, OPM utilizes natural language-like syntax to make it easier to understand for non-modelling experts.

2.3.4. KerML

KerML is a foundational metamodel proposed to the object management group (OMG). The stated purpose of KerML is to serve as a basis for newly developed modelling languages, including SysML v2 in particular. KerML distinguishes between two fundamental classes: *Element* and *Relationship*. In KerML everything is built upon these classes, specializing and extending them. At its core every class in KerML is an Element, which provides means for identification and ownership. Similarly, every structural relationship between these elements is a specialisation of the relationship class, which provides means to connect two or more elements.
KerML provides a number of predefined specializations of elements, which are, among others:

- Package - defines a namespace but offers no functionality on its own.
- Class - owns a relationship, and specializes a general class.
- Feature - describes properties or constraints.

Additionally, KerML provides extensions to its relationship class to provide means to:

- Model ownership hierarchy, making it possible to remove an element and all it's owned elements
- class specialization to model inheritance of features
- binding values and expressions, enabling computation by binding a value to the sum of others for example.

The full specifications of KerML are publicly available and, while highly unlikely to change, not final at time of writing as it is not yet officially standardized [119].

2.3.5. SysML v2

SysML v2 is the first extension to KerML and developed and proposed alongside KerML. It extends KerML to provide a general-purpose modelling language designed to facilitate the MBSE approach and reiterate upon SysML with the lessons learned. SysML v2 provides means to represent requirements, structure and the behaviour of a system. In addition it provides means to model specification and verification cases. The goal of SysML v2 is the support of a multitude of systems engineering methods and practices. While it is a comprehensive, domain-independent language, the OMG expects it to be extended to include more domain and purpose specific extensions, all compatible with each other by using SysML v2 or KerML as the common basis.
Being a comprehensive extension to KerML it provides a multitude of important modeling constructs:

1. *Dependencies* provide means to model dependencies between elements.
2. *Variability* allows for modeling of different variants. A system can be configured by choosing specific variants and provide a means to check for consistency if all choices satisfy the corresponding constraints.
3. *Structure* includes representation for different parts, how they are decomposed, interconnected and classified.
4. *Constraints* specify conditions which are expected or required to satisfy and are further specialised by *Requirements*.
5. *Behaviour* specifies the interaction of parts.
6. *Cases* define steps required to produce a desired result and are especially used in the context of analysis, verification and use cases.
7. *Attributes* which are data type definitions.
8. *Parts* to be able to model decomposition.
9. *Requirements* which bind a constraint to a feature.

Additionally, SysML v2v2 provides specializations of the KerML libraries, as well as further libraries which are discussed by Bajaj, Friedenthal and Seidewitz [13].
Furthermore, SysML v2 provides a multitude of additional classes for features like use case modelling or modelling dynamic behaviours. The full specifications are available publicly in [119].

2.3.6. SysMD

SysMD consists of a modelling language, as well as a tool, called SysMD Notebook. SysMD has been initially designed to be an modelling tool and language providing a low entry barrier to MBSE with domain experts with little to no systems engineering experience [41]. SysMD was designed to extend the SysML v2 standard and be fully inter-operable with KerML. SysMDs main features include::

- A syntax inspired by natural language, following ideas from literate programming.
- The ability to define and check constraints and consistency.
- Being inter-operable with SysML v2.
- Support incomplete models in a textual representation.

The easy to read and write syntax from SysMD follows rules from natural language and is designed to provide a gentle learning curve. The structure of a SysMD statement follows the form:

```
Subject predicate Object "."
```

with the possibility to specify *objects* further.

```
Subject predicate
    Object [":" constraint] = instances
```

Subjects are elements of arbitrary type, *objects* can be either elements or a list of statements which are separated with ",". An *objects* defined as such a list can be considered as a sub-clause. Each object is preceded by an descriptor, which can define it as an attribute or part for example [40]. This is an update to earlier SysMD versions and a concession to the interoperability with SysML v2 [41].
The goal is to provide a syntax readable without prior experience with SysMD or even programming or modelling. This provides a natural language description of a system and can function as documentation and guidance to define a fully formalized model. Without the need for block definitions or following common programming conventions this provides a low entry barrier into MBSE.
The independence of a block structure results in definitions to be made independent of the sequence of statements. This is an important feature, making it possible to integrate SysMD easily into a notebook programming style approach and facilitating iterative design and updates of a model at all stages of the development process.

Chapter 3

Behaviour Influence

One key aspect of inclusive systems engineering (see Chapter 5) is the inclusion of user guidance. In order to provide an A number of influencing techniques have been proposed and implemented both in analog and digital environments. In order to efficiently guide users and avoid adverse effects it is important to review the current theory on human decision making, biases influencing these decisions and techniques making use of those biases to guide users towards desired behavior.

3.1. Human Decision Making

We make decisions all the time, with only a small number of them deliberately thought about. The vast majority of decisions we do automatically, falling back on a plethora of heuristics we have learned. A decision is understood as a cognitive process during which the decision maker has at least two alternative courses of action available to take [9]. Making a decision consists of several phases [60] [42].

1. Collecting data and information
2. Filtering information to classify as relevant or irrelevant
3. Analyzing and processing the information filtered for decision making. Newly acquired information and information of different sources can be merged and linked.
4. Formulating various decision alternatives, identifying different courses of action.
5. Weighing the various alternatives by comparing them
6. Choosing the (subjectively) best course of action

The idea of the so-called *ergonomic man* from the 1950s supposed a decision maker who is completely informed, infinitely sensitive and rational [47]. However, this idea has been refined throughout the years, accepting that human

decision making is often flawed and prone to biased decisions, being recognized as a distinct field of study which began to be recognized as such in the 1970s and 1980s. The work from Kahneman and Tversky [74] from 1979 showed how emotions lead to an irrational decision-making process. In 1992 they expanded their prospect theory towards cumulative prospect theory [124] to consider additional traits such as overconfidence and limited attention.

3.1.1. Dual-Process Theory

The dual-process theory on behaviour was presented by Kahneman and distinguishes between a fast and automatic system, called "System 1" and a slow, reflective system called "System 2" [72]. "System 1" is described as being fast and makes decisions effortless, which guides the vast majority of actions taken, which includes actions such as checking the current time which works subconsciously. "System 2" guides conscious decisions and behaviour. These systems are not fully independent, being able to influence each other. Kahneman provides the example of voluntarily controlling the chewing process in which "System 2" regulates and overrides the involuntary process. Another example is "System 1" being responsible for directly responding to an unexpected sound whereas "System 2" then controls the following actions [72].
The suggested model has been refined since its inception with Evans and Stanovich [50] proposing the dual-process theory of higher cognition in which they distinguish between "type 1" and "type 2" processes. While these correspond to "System 1" and "System 2" respectively, they don't distinguish between two independent systems but between two different types a stimuli can be processed leading to a certain behaviour. In figure 3.1 an overview is given of the defining features of these processes, their typical correlations and the relationship to the previously used systems. The most important distinction Evans and Stanovich provide is the focus on the requirement of working memory.

While human decision making is far from being fully understood the dual-process model provides an important distinction between automatic decision making which is prone to biased responses and the slower, conscious decision making, relying on reflecting and active thought process which can be controlled and is able to willfully overrule automatic decisions.

3.1.2. Behavior Biases

Behavior biases influence decision making, increasing the likelihood of actions. Most of these biases are heuristics affecting type 1 processing, reducing the response time for a decision and are good enough to result in actions with good or at least not bad consequences in the vast majority of the time. Additionally, these biases are processed subconsciously and influence the very first phases of decision making, the collecting, filtering and analyzing of data, taking effect very early in the process.
Giving a detailed description of all biases that can be used to influence decision making would go beyond the scope of this thesis. However, a limited number of

Type 1 process (intuitive)	Type 2 process (reflective)
Defining features	
Does not require working memory	*Requires working memory*
Autonomous	*Cognitive decoupling; mental simulation*
Typical correlates	
Fast	Slow
High capacity	Capacity limited
Parallel	Serial
Nonconscious	Conscious
Biased responses	Normative responses
Contextualized	Abstract
Automatic	Controlled
Associative	Rule-based
Experience-based decision making	Consequential decision making
Independent of cognitive ability	Correlated with cognitive ability
System 1 (old mind)	System 2 (new mind)
Evolved early	Evolved late
Similar to animal cognition	Distinctively human
Implicit knowledge	Explicit knowledge
Basic emotions	Complex emotions

Note. Italicized attributes are the proposed defining characteristics in the current article. Authors proposing two systems include the features attributed to Type 1 and 2 processing but may also include the additional features named.

Figure 3.1.: *Clusters of Attributes Frequently Associated With Dual-Process and Dual-System Theories of Higher Cognition [50]*

biases are discussed briefly, being widely used as a basis for nudging, persuasive systems design and recommender systems.

Anchoring & Adjusting

Effect: If someone make decisions they usually base these decisions on available information. This is called anchoring. While a good mechanism in general, it can result in extremely biased behavior. An anchor could be unreasonably high or low or even arbitrary. It was shown that even randomly generated anchors, for example by a fortune wheel, can influence our behavior[72].
Processing affected: Type 1 processing is affected, as they are not aware that an anchor is used in either anchoring or adjusting from said anchor.
Further Reading: Dolan et al. provide a good overview of some of the effects discussed, including anchoring and adjusting[45].

Framing

Effect: It is very important in which context a decision is made. Two equal statements can evoke very different reactions. This is especially important when considering loss aversion. Both of these effects are often considered in conjunction. But this does not have to be the case. Framing the fuel consumption by fuel per distance instead of distance per fuel can have a high impact [72]. Framing can intensify other effects or subdue their influence by making certain aspects more prominent.
Processing affected: Type 1 processing is mostly affected, especially in conjunction with loss aversion.
Further Reading: See De Martino et al. for an analysis of gambles and framing[44].

Loss Aversion/Endowment Effect

Effect: We usually value a loss higher than a gain[123]. Losing something has a higher emotional effect than acquiring it. A prominent experiment on this effect has been replicated countless times. Half of the students in a lecture are given mugs with the university logo on it. Those that got a mug are asked at what prizes they would sell the mug while those who have not received a mug are asked at what prizes they would buy. The average selling prize is about twice as high as the average buying prize[72]. The loss of a mug is valued higher than the acquiring of a new one.
This effect has also been studied in case of gambles. The idea is to offer a gamble where one can lose or win an amount of money dependent on a coin toss[123]. For people to take the gamble the possible winnings have to be double the possible losses. However, this gamble also depends highly on the number of gambles offered and the framing of the question. Gambles depend heavily on framing, while the mug example shows loss aversion.
Processing affected: This affects type 1 processing and is based upon basic emotions which are especially hard to be regulated by type 2 processing

Further Reading: For a deeper analysis of gambles refer to Rabin and Thaler[104] and to De Martino et al. for an analysis of gambles and framing[44]. Kahneman, Knetch and Thaler go in-depth on loss aversion as well as the endowment effect[73].

Social Norms

Effect: Behavior often follows social norms. These norms are usually perceived as ideal behavior and it is common to think not conforming to such norms will result in isolation and ridicule. Conforming with others is deeply rooted in our behavior and experiments show that even blatantly wrong statements are often accepted in order to conform to the other participants in the experiments [123]. Social norms can take different forms, people are more likely to trust a group or a particular confident speaker, especially if this speaker is also approved by others. Especially social groups and figures hold in high regards are imitated. However, sometimes people comply to norms they do not personally approve off. Challenging a social norm is difficult, making it hard to change social norms, even if most people in a society do not approve the social norm.
Processing affected: Depending on the particular norm both systems can be affected. When the norm is not deeply rooted in everyday behavior type 2 processing overrides the typical behavior to comply with the norm. When the norm has been in effect for a long time type 1 processing has adopted the behaviour change and is affected by the norm too. In this case type 2 processing would be necessary to challenge the behaviour again.
Further Reading: Cialdini and Trost provide a thorough foundation of social influences [32].

Status Quo Bias

Effect: The status quo bias is the effect that people tend to keep whatever is already in place. This ranges from not switching a channel on television to automatic renewal of subscriptions and, depending on the default, being an organ donor or not.[123]. Changing something requires energy and conscious decision making. Even if this change only requires very little energy (like the pressing of a button on a screen) it is often omitted and easier to not think about it. The status quo bias is the most researched of the effects used to nudge and has proven to be the most robust in terms of effectiveness. Even when people are explicitly informed of the effect and that it is used in order to change behavior, it is effective. When the conscious decision to change is done, implementing this change can take a lot of time. This can be further hindered by making the change require additional energy, making a phone call instead of just visiting a website, or even filling out a form for example.
Processing affected: Type 1 processing is affected. Changing options requires reflection upon the decision. This requires working memory and type 2 to make the decision to behave against the status quo.

Further Reading: Kahneman, Knetch and Thaler provide a thorough analysis of the status quo bias [73].

3.2. User Guidance Techniques

3.2.1. Nudging

Thaler and Sunstein propse the possibility to influence behaviour by designing the environment, which they call choice architecture, in order to make a certain decision more likely than others [123] which is called nudging. These nudges are based on biases and basic emotions and target type 2 processing in the vast majority. Nudging has been adapted in the field of human-computer interaction (HCI) with the adoption of nudges in a digital context. The digital choice architecture can be quickly modified and modified for a distinct user with information of previous choices, enabling the personalization of nudges [39] [75].
Jesse and Jannach [71] identified 87 mechanisms and four broad categories of nudges, extending the 23 identified machanisms of nudging from Caraban et al. [28].

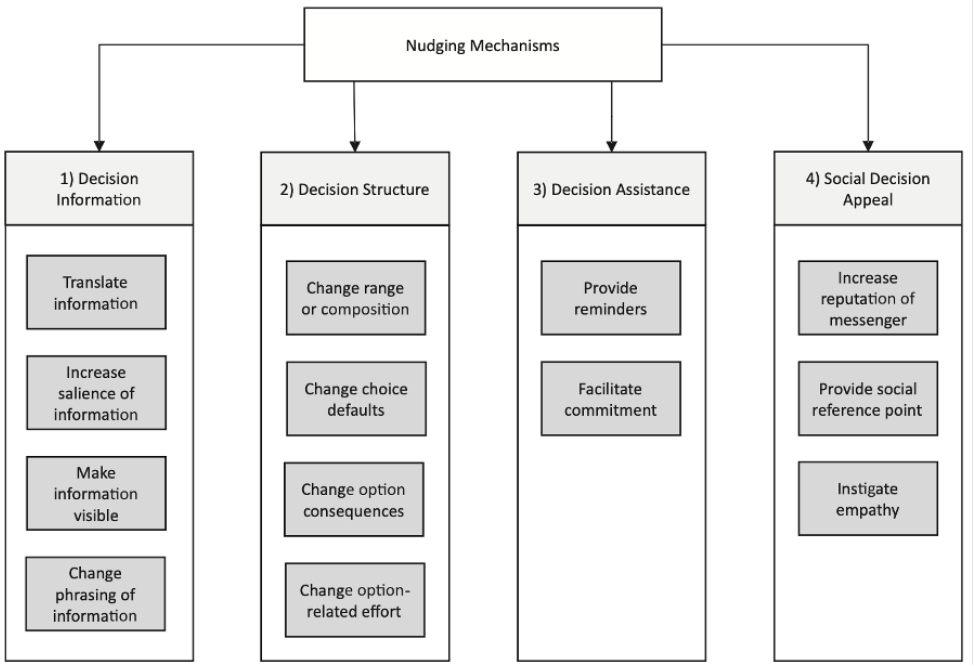

Figure 3.2.: *Taxonomy of nudging mechanisms defined by Jesse and Jannach [71].*

Fig 3.2 shows the taxonomy proposed by Jesse and Jannach. Each category describes an aspect of a decision, and how these aspects can be changed by nudge mechanisms. The 87 identified mechanisms provide examples of these nudge mechanisms with the most commonly used examples being aspects of information simplification, providing a default choice and making use of social comparison.

An important aspect of digital nudging is the focus on single decision points. The impact of one decision on future ones is usually not considered. A long term behaviour change is only pursued implicitly by nudging a recurring decision each time.
The proposition of nudging has sparked a debate on the ethical implications of nudging, already briefly discussed by [123]. Lembecke et al. have provided three core principles important to keep in mind when the use of nudges is considered [82].
Autonomy / Freedom of Choice: A nudge intervention must be easy to avoid as one core principle of nudging is to preserve the autonomy of a user. However, it can be argued that in the modern environment the number of fully autonomous choices are minimal due to the digital environment being mostly created by companies who do not subscribe to preserving the autonomy of their users [118] [117].
Transparency: Without the knowledge of being nudged a user is less likely to be able to identify a nudge and thus avoid it if they choose. What can be considered a sufficient transparency is an ongoing debate, however, in order to classify transparency of nudges the schemata of Hansen and Jespersen can be referred to [61].
Goal-Oriented Justification: Calvien [33] examines strands of argumentation typically used to justify nudges as ethically acceptable. Nudges are supposed to work towards benefits for social goals or the person who is nudged. Selfish goals, such as profit, are not considered as ethical reasons for nudging according to Thaler and Sunstein [123].

3.2.2. Persuasive Systems

The idea to use user interface design to guide user decision making has been pursued by Fogg [52] in the work on persuasive technology. In this work the first framework with five principles was introduced. These principles have since then been expanded towards 28 by Oinas-Kukkonen and Harjumaa [94] and have been included into the work my Murillo-Munoz et al. [92] as well.

Figure 3.3 illustrates examples for the four categories of the last phase of the persuasive system design (PSD) process. These are the specific principles used by persuasive systems design to influence the user behaviour. The primary task support provides influence by simplification and personalization by omitting unimportant information for the user. Social aspects influence all other categories provided by emphasising the social role of the system, appealing to authority and providing social comparison respectively.
The primary task support is highly focused on type 1 processing whereas the other categories take type 1 and type 2 processing into account by provoking reflection on choices and their consequences as well as social implications.
One key difference to nudging is the intend of a lasting behaviour change in PSD by targeting type 2 processing in order to change future behaviour as well as the single decision point. PSD has been applied in multiple healthcare applications and studies, providing evidence for successful implementation [110].

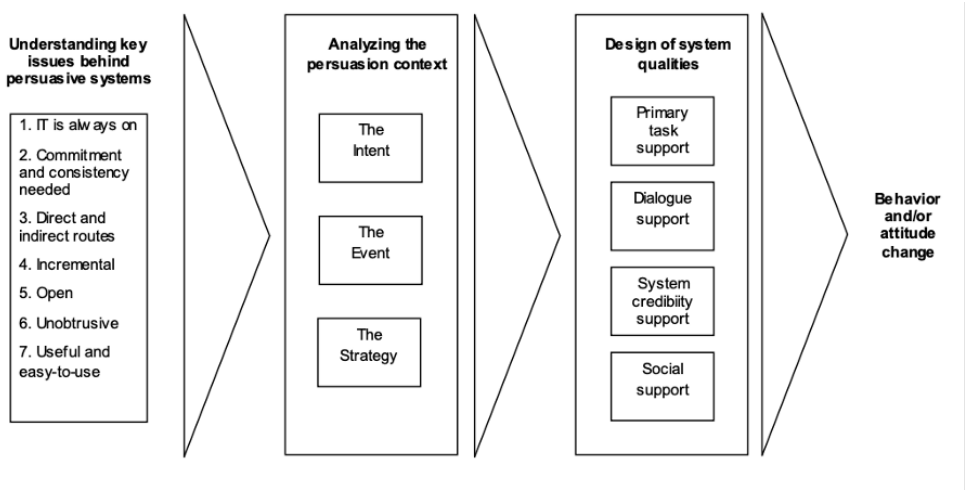

Figure 3.3.: *Oinas-Kukkonen visualisation of persuasive systems [94].*

3.2.3. Recommender Systems

Research has shown that people tend to decide favorably towards recommendations given by their surroundings [107] which has motivated extensive research on so-called recommender systems. These systems gather possible choices by means of finding similarities in content, using ratings given by other users and more [27], to recommend an action or product to a user. Large quantities of data are used to generate these suggestions to provide the perceived best option to the user. The actual quality of such suggestions highly depend on the algorithms used to identify the options and the intended purpose of the recommendation. The pertinence of the suggestion is highly dependent on the considerations of the user. Jesse and Jannach [71] provide evidence of only a small number of nudging techniques having been explored yet in recommender system design. In general, the majority of recommender systems have the goal to reduce the workload of a user in the future by giving suggestions in order to simplify the coming decision. In most cases the recommender system targets type 1 processing unless the active reflection on a decision is prompted to affect future recommendations.

3.2.4. Gamification

In the work of Matallaoi et al. [88] gamification is defined as the integration of game mechanics and elements into non-game environments with the purpose to increase user engagement, enjoyment, as well as loyalty. The goal of gamification is increased motivation, resulting in a change of undesired behaviour into desired behaviour [7]. This is based on the work of Fogg providing an argument for motivation being a strong inherent force, being able to willfully change behaviour [51].
In order to stimulate the user type 1 processing is targeted by utilizing mentally stimulating game aspects. This is done to reduce aversion towards a

task and prompting type 2 processing in future decision making to overrule the intuitive response by reflecting upon the past behaviour and the increased stimulation from the game aspects.

3.2.5. Comparison

A comparison of the above methods is discussed in [37] highlighting similarities between these methods.
All methods utilise biases influencing type 1 and type 2 processing in different ways, with the most prominent biases being framing and simplification. Type 2 processing is usually influenced by providing feedback or prompting social reflection as well as relying on self-monitoring and inherent motivation as a basis [3].

Influence Method	Nudging	Persuasive Systems	Recommender Systems	Gamification
Timing	Decision making	Decision making Reflection	After a decision	Change current decision feedback
Decision affected	Current decision	Current decision Future decisions	Guiding towards future decisions	Future decisions
Processing targeted	Type 1	Type 1 Type 2	Type 1 Type 2	Type 2
Lasting behaviour change?	Repeated nudging	Repeated use social implications Reflection	Following decisions No lasting change	Main intention

Table 3.1.: *Comparison between Nudging, Persuasive Systems, Recommender Systems and Gamification on when the influence is used and to what end [37]*

Table 3.1 provides an overview of the influencing methods, at which time they are used and which decisions are affected. It shows which processing is targeted and if a lasting behaviour change is intended. This provides a broad classification as each of these methods can be implemented in a number of ways.
However, a connection between nudging and PSD is identified, both targeting the decision at hand and predominantly using means to influence type 1 processing. The most important distinction is the intent of behaviour change for future decisions. Nudging only aims to change behaviour of a current decision, requiring repeated nudging to result in lasting behaviour change whereas persuasive systems often aim to achieve a lasting behaviour change by including type 2 processing effects into the design.
Recommender systems and gamification show similarities as well, the main differentiation being intension for lasting behaviour change. Both methods have the goal of changing future decisions, however gamification is directed towards increasing motivation to make recurring decisions more likely when repeating a task, whereas recommender systems focus on similar decisions instead of recurring one.
Referring to figure 3.2 it is possible to categorize recommender systems and gamification as special forms of nudging, usually providing means to make information more visible, change the effort of a decision and providing social reference points.

Chapter 4

Surveys on Systems Engineering Issues and Preferences

4.1. Methodology

To provide a user-centric extension to the current systems engineering process it is necessary to gather insights into the needs and preferences of systems engineers with varying levels of expertise, ranging from no experience in systems engineering towards professional level systems engineers. Several methods to gather this insight were taken into account.
Surveys: Surveys are a widely used method and are especially cost-effective when the exact number of participants is not known previous to conducting the research. This is especially true for online survey, offering the ability to be created and shared over a long period of time, gathering data with minimal additional effort over time [127]. The structured feedback of surveys facilitate comparative analysis between groups of participants or particular features of a product or approach [16]. However, surveys can have difficulties capturing nuances of user behaviour and experience. Additionally, social norms can influence the answers participants provide, as can the survey questions themselves, making it important to take particular care on question design [58].
Interviews: Interviews can provide in-depth and detailed answers of user experiences, attitudes and needs through conversation with individual users. Interviews are best used when user motivation, mental models and emotional response are of particular interest [16]. The flexibility of the format make it possible to adapt questions based on a particular response, making sure both the question and answer are thoroughly understood [8]. Interviews are resource-intensive which is a significant drawback. Interviews require time for preparation, execution and for analysis. Furthermore, they rely heavily on both the participants ability to articulate their experience and the ability of the interviewer to quickly identify the need to adopt questions.
Focus Groups: Bringing together a small number of users to discuss topics of interest with moderation allows for interactive exploration of shared experi-

ences. This method of using a focus group to gather insights is particularly suited to identify common and divergent perspectives and can provide insights not gathered from individual interviews [16]. However, social influence can influence how the members of the group interact, leading to a higher level of agreement.
Observational Methods: The systematical watching and recording of user behavior provides direct evidence of how users interact with a certain tool or feature in a specific environment. This method provides insights on what users do instead of what users report, which can differ due to social norms, over- or underestimations and other factors. However, the presence of observers can potentially influence the behaviour [62]. In addition, this method is similar to interviews when considering resources such as time required.

For this thesis the method of surveys has been chosen. The benefits of providing an asymmetric method to gather data was the most significant advantage of surveys with the added benefit of significantly reducing the financial requirements to gather the desired data. Additionally, the issue of recruiting participants was amplified with the research starting during a time when pandemic prevention measures were enforced.

4.2. Student Survey

In order to identify the needs and preferences of future systems engineers a survey with students from the RPTU Kaiserslautern-Landau was conducted. This survey was intended to get general feedback on visualisation and representation from people with no or only very little systems engineering experience.

4.2.1. Hypothesis Formulation

From chapter 2.2.5 it is save to assume that one key factor contributing to the high entry barrier of systems engineering is the complex syntax which contributes in a long training time. This in turn is valued higher due to loss aversion and leads to status quo bias. The early version of SysMD (see section 2.3.6) was designed to provide intuitive, easy to learn syntax to reduce this entry barrier. In addition, SysML v2 (see section 2.3.5 has a strong focus on providing a textual model representation. Providing both a graphical and textual model representation provides the option to use them for different purposes. Two possible purposes are the creation of new models and the understanding and ability to explain a model to others.
This gives rise to the following hypothesis:

Hypothesis (H1) SysMD is intuitive to use.

Hypothesis (H2) Graphical model representation is preferred to understand and to explain it to others.

Hypothesis (H3) Textual model representation is preferred to create a new model.

Hypothesis (H4) SysMDs simplified syntax reduces the systems engineering entry barrier.

4.2.2. Setup

The first evaluation was setup as an online survey targeted at students of the RPTU Kaiserslautern-Landau. This survey was propagated via university wide newsletter and offered the opportunity to win a 10€ giftcard for participation as an incentive.

The survey was designed to check for intuitive systems engineering design by both asking for the direct perception of intuitiveness and the indirect perception to generate a composite intuitiveness score. The survey consisted of four sections.

Metadata

A small amount of metadata was collected, namely the systems modeling experience, used modelling languages, age and course of study.
Thirty students participated in the study, with only 5 of these with prior systems modelling experience. The age ranged from 21 to 34 years old with backgrounds in computer science, math, cognitive science, electrical engineering, chemistry and mechanical engineering. These students were currently in their bachelor or master semesters and full-time students.
Two of the students with prior systems modelling experience had used SysML before, one of them and two more students had experience with UML while one student ha modelled system behaviour by using petri nets but no experience with UML or SysML.

General Preference

This section asked for the general preference regarding textual or graphical representation of models.

- Would you prefer a textual or graphical representation of a model in order to understand it yourself.
- Would you prefer a textual or graphical representation of a model in order to explain it to others.
- Would you prefer a textual or graphical representation of a model in order to create a model.

SysMD

This section evaluated SysMD as a modelling language. To do so a small example model was presented in the SysMD language. This model was evaluated by the participants on six questions to derive the composite intuitiveness score in addition to the direct intuitiveness question. The questions where

presented as statements to be ranked from *strongly disagree* to *strongly agree*. The composite questions consisted of:

- SysMD is easy to read.
- SysMD is easy to write.
- A description can easily be modeled in SysMD.
- The statements used in SysMD are clear to understand.
- I'm able to understand a SysMD model.
- I would be able to explain a SysMD model to someone else.

SysMD, SysML, SysML v2 comparison

The last section compared SysMD, SysML and SysML v2 by presenting the same model in all three languages. The participants then where asked which language they prefer.

- Which of the three models do you think is the most intuitive?
- Which of the three models is the easiest to understand?
- Which model would you use to explain it to someone else?
- In which modeling language would you most likely create a new model?
- Which of the three models is the most clear in its use of keywords and relations?

4.2.3. Results and Evaluation

In general the students clearly preferred a graphical representation over a textual representation in order to understand a given model themselves. The majority preferred to have access to both a textual or graphical representation while nine participants preferred to have only a graphical representation and one preferred the textual representation only. In order to explain a model to others the use of only a graphical representation increased to 16 of 30 participants preferring to explain a model with only a graphical representation, whereas two participants prefer a textual representation for this task. This preference for graphical representation decreases significantly towards a textual representation preference when tasked to create a model themselves. Seven participants prefer to only use a textual representation to create a model whereas twelve prefer to have access to both and ten prefer to create a model with a graphical representation. Even though the graphical representation is still preferred overall the shift towards textual representation for creation purposes is important to consider and likely to increase with more complex models which are harder to represent graphically.

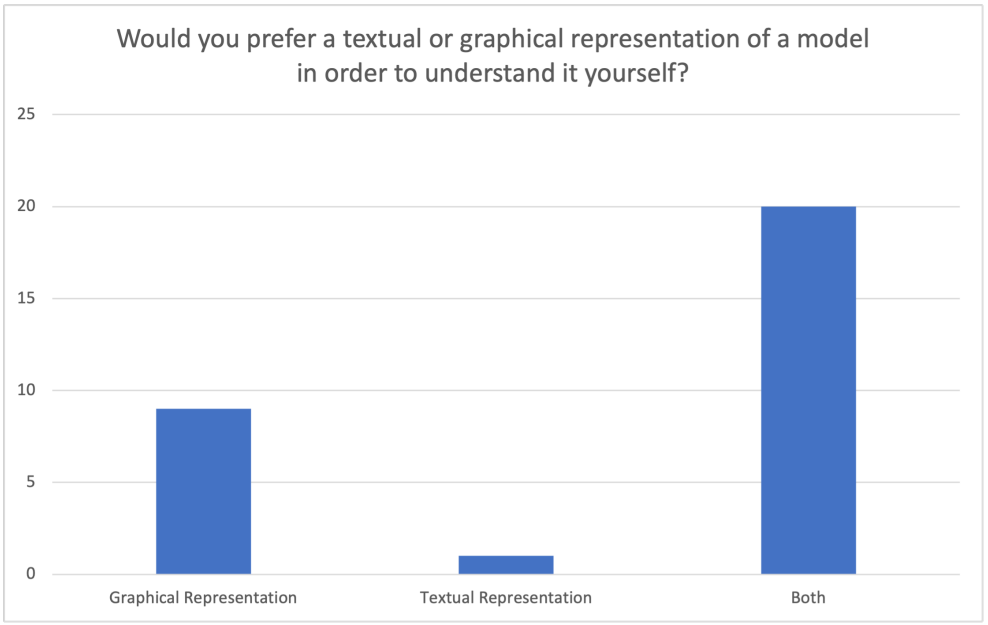

Figure 4.1.: *Preference of graphical vs textual representation to understand a model.*

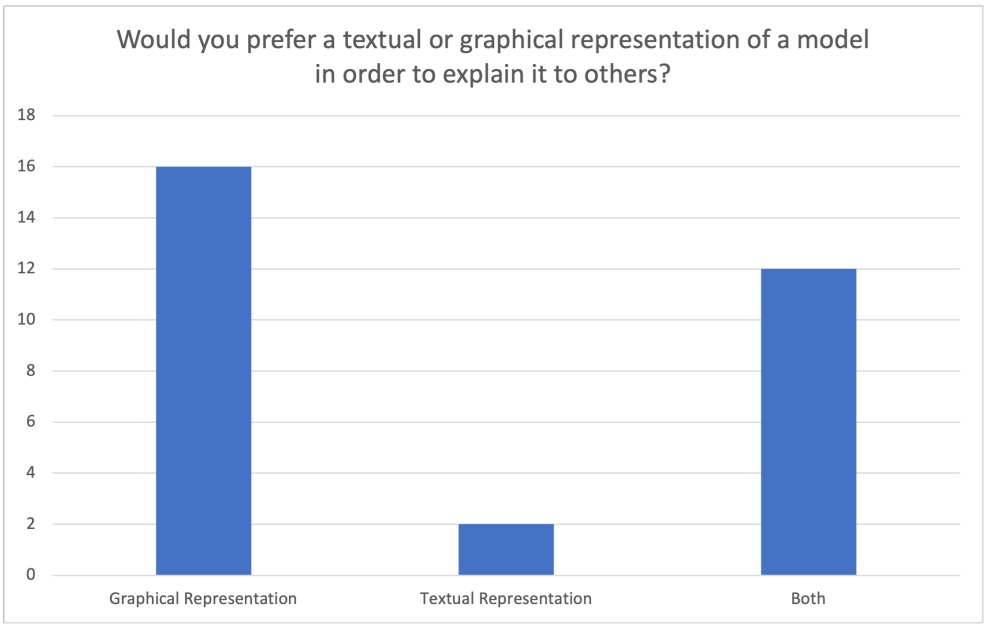

Figure 4.2.: *Preference of graphical vs textual representation to explain a model to others.*

SysMD was generally agreed upon to be intuitive (refer to figure 4.4), easy to read (refer to figure 4.6) and easy to understand (see figure 4.9). Figure 4.5shows that when asked if the participants would be able to explain the given SysMD model the participants were generally unsure, some feeling con-

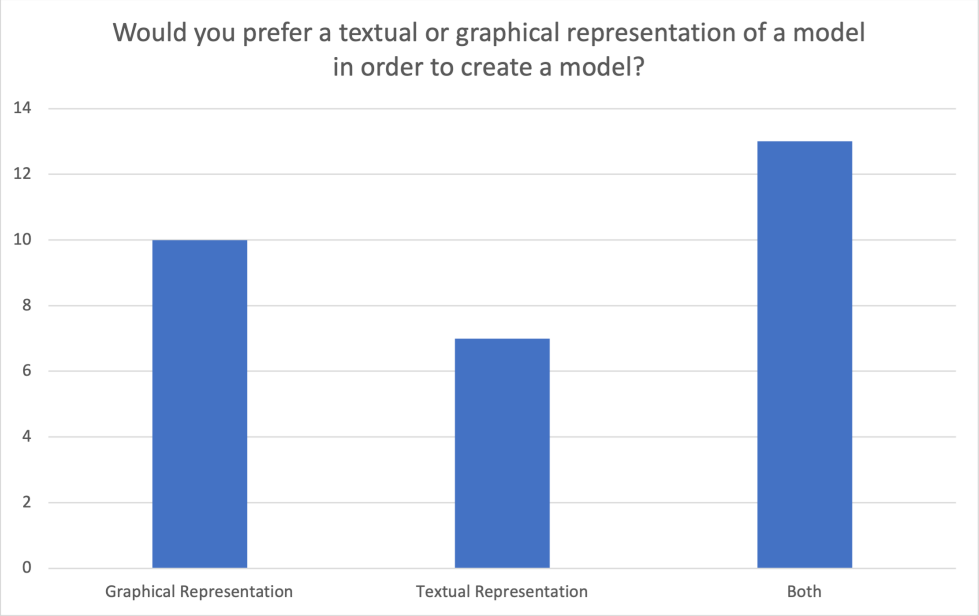

Figure 4.3.: *Preference of graphical vs textual representation to create a model.*

fident to be able to while others were not. The ability to read and understand the SysMD statements does not translate to a confidence being able to create a new model in SysMD as shown in figure 4.8, which is not surprising, given they only had a very small example as a basis and no general model-based systems engineering experience. However, the ability to understand the given model and the statements give support to the idea of translating the complex systems engineering concepts into easy to understand, natural language adjacent, statements to simplify the creation of models and facilitate comprehension.

SysML and SysMD are overall preferred by the participants, when comparing the same model in each language. SysML is considered to be the most intuitive and the most suited to explain it to others (shown in figures 4.14 and 4.13), which is in line with the preference for graphical representation to understand a model and explain it to others. SysMD is considered easier to understand (see figure 4.9) and to have the more concise or clear keywords and relations (see figure 4.11) but lacks a graphical representation to be considered intuitive. Given the preference of graphical representation it is surprising to see that over half of the participants consider SysMD to be easier to understand than SysML, showing the importance of easy to understand syntax, especially for non-systems modeling experts. This is also evident in the preference to create a new model in either SysML or SysMD with only five participants choosing SysML v2 as the preferred means to create a new model, as shown in figure 4.15.

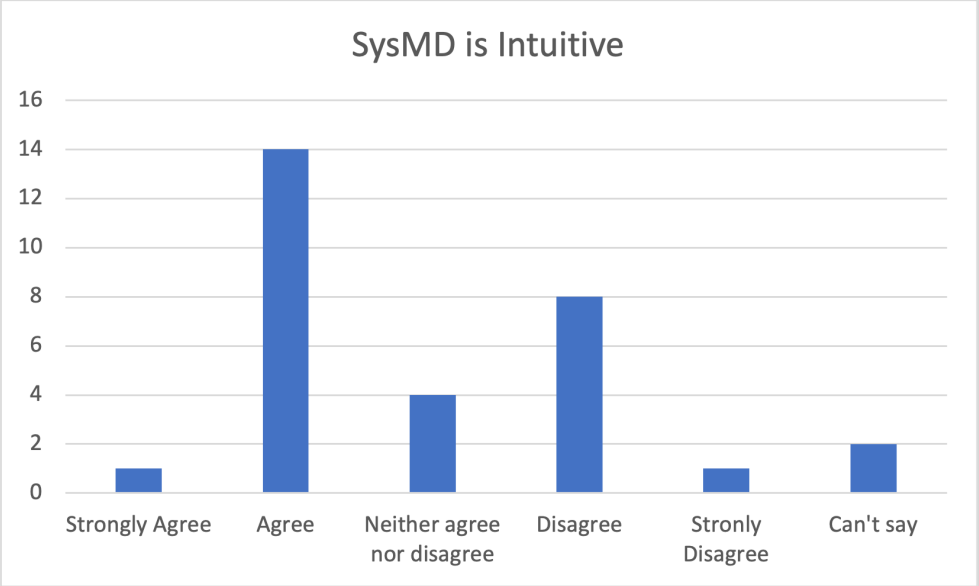

Figure 4.4.: *How much participants agreed with the statement that SysMD is intuitive.*

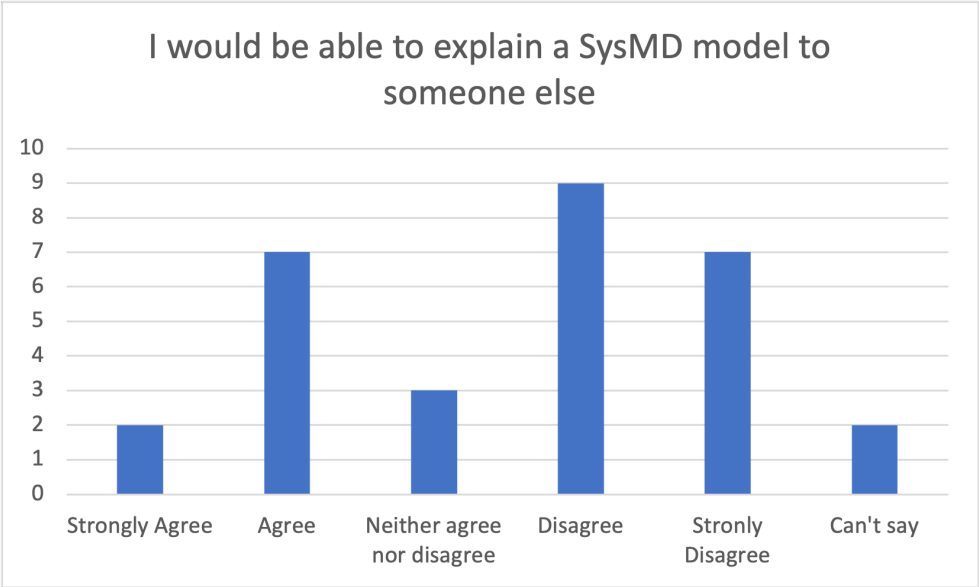

Figure 4.5.: *How much participants agreed with the statement that SysMD is easy to explain.*

4.2.4. Hypothesis Evaluation

- Hypothesis H1 can be considered to be true as seen in the direct response seen in figure 4.4 as well as the implicit response from figures 4.6 and 4.7.
- Hypothesis H2 is supported as well as seen both in figure 4.1 and 4.2.

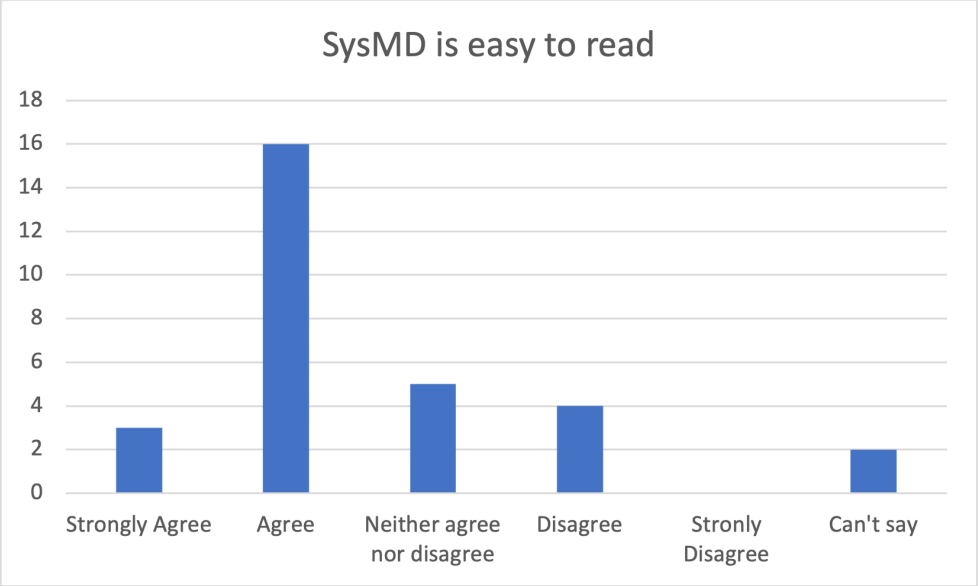

Figure 4.6.: *How much participants agreed with the statement that SysMD is easy to read.*

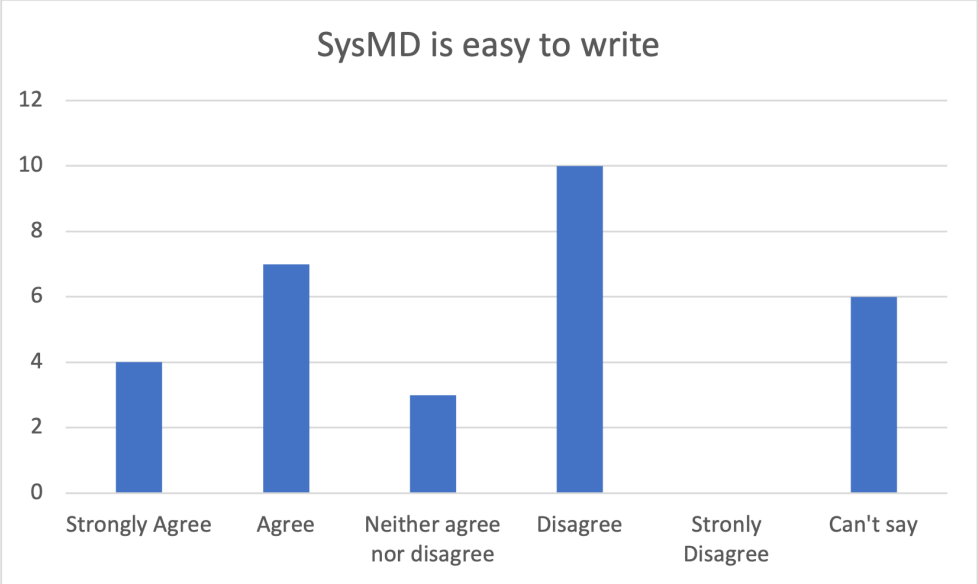

Figure 4.7.: *How much participants agreed with the statement that SysMD is easy to write.*

Both showing a clear preference for graphical representation when unable to access both a graphical and textual representation.

- Hypothesis H3 is not supported as shown in figure 4.3. The preference between textual or graphical representation not clear with a slight preference towards graphical representation.

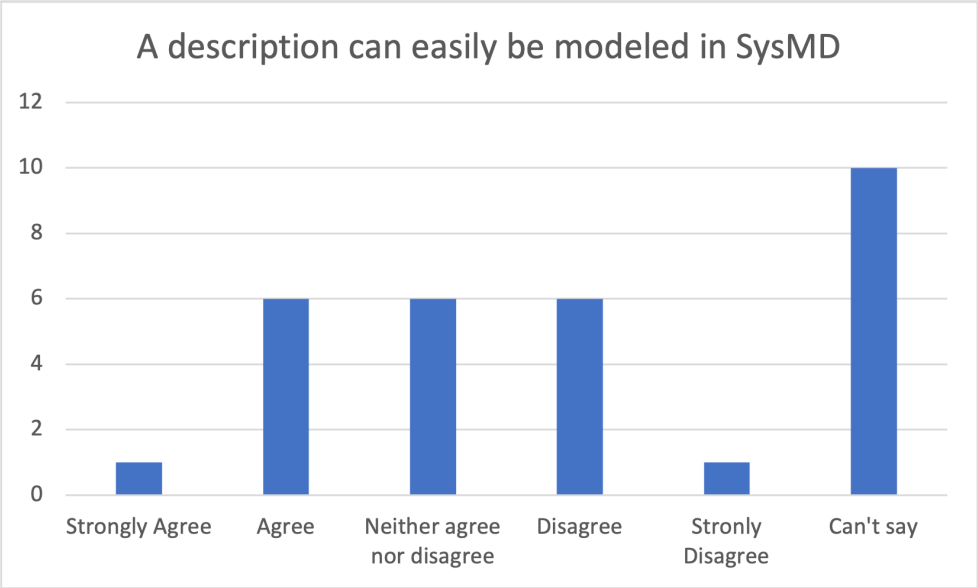

Figure 4.8.: *How much participants agreed with the statement that a new model is easy to be created with SysMD.*

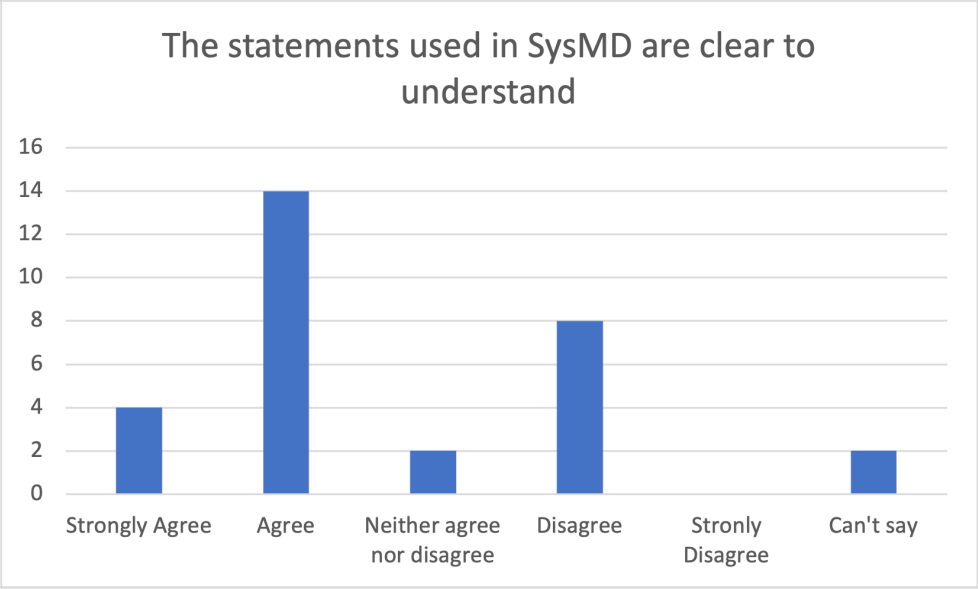

Figure 4.9.: *How much participants agreed with the statement that SysMD is easy to understand.*

- Figures 4.11 up to 4.15 show a general preference towards SysML, which is in accordance to the preference of a graphical representation of models. When comparing the syntax of SysMD and SysML v2 the preference towards the simplified syntax of SysMD is illustrated. This supports hypothesis H4 when considering only textual representations of models.

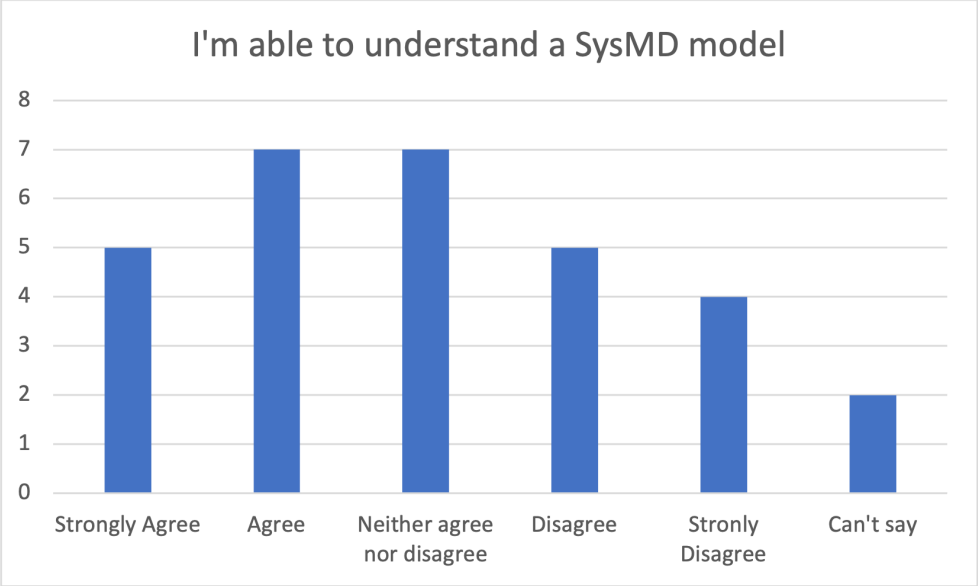

Figure 4.10.: *How much participants agreed with the statement that models in SysMD are easy to understand.*

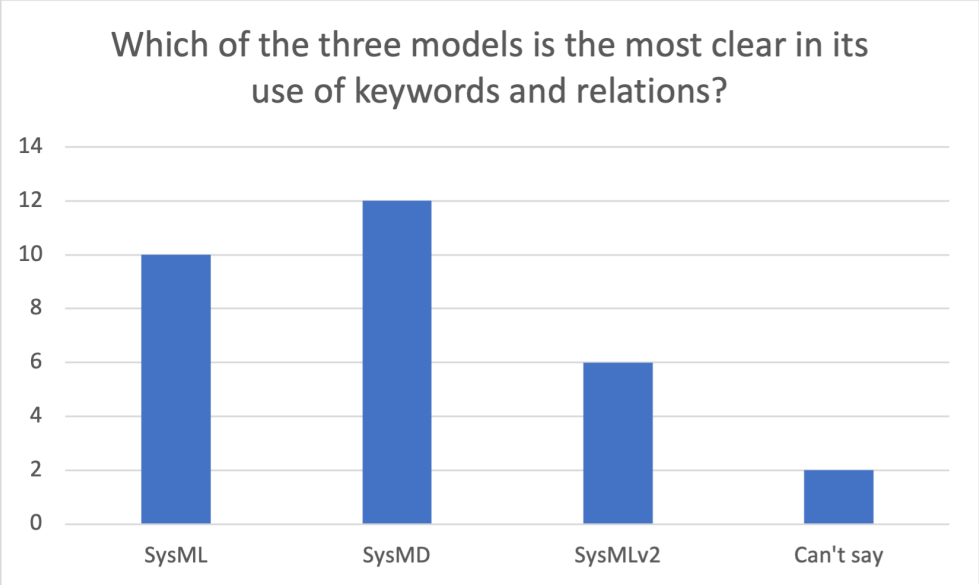

Figure 4.11.: *Rating of which of the three given modelling languages (SysML, SysML v2 and SysMD) has the most clear statements.*

4.2.5. Limitations of the first survey

Due to the limited number of participants in our survey, the results should be interpreted as indicative trends rather than statistically robust findings. The participants in this survey have been students from the RPTU Kaiserslautern-Landau with almost no systems engineering experience. The low number of

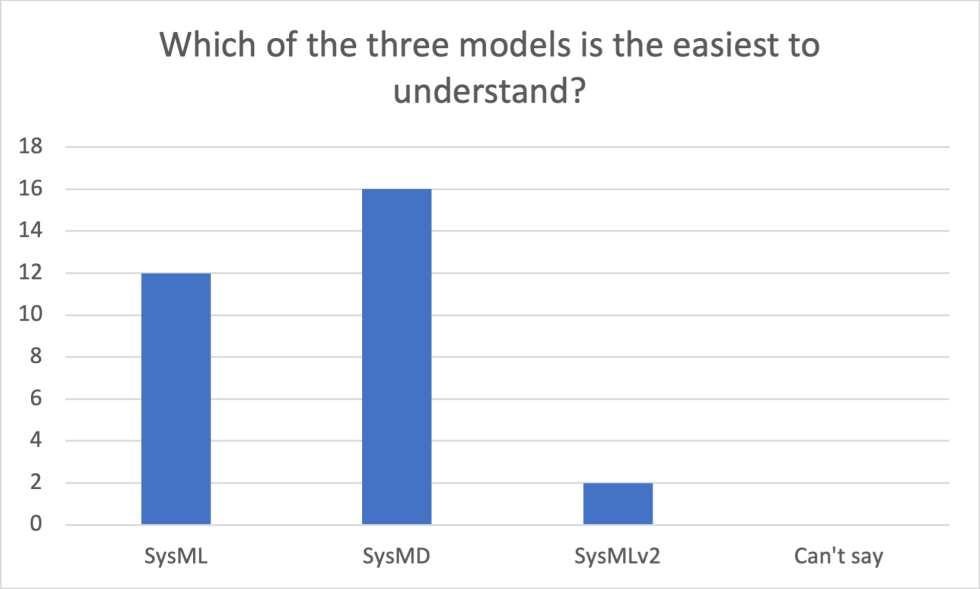

Figure 4.12.: *Rating of which of the three given modelling languages (SysML, SysML v2 and SysMD) is the easiest to understand a model in.*

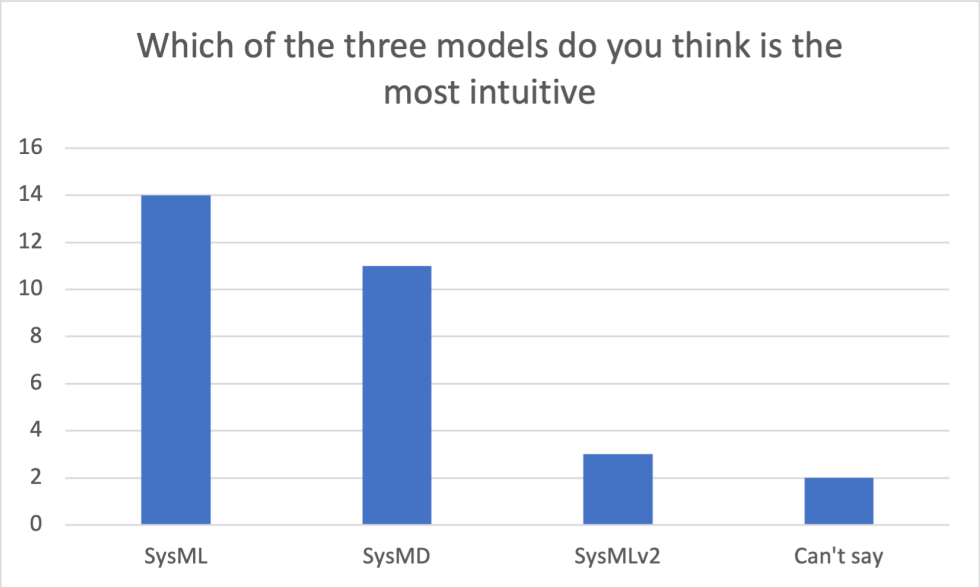

Figure 4.13.: *Rating of which of the three given modelling languages (SysML, SysML v2 and SysMD) is the most intuitive*

participants makes this survey prone to a sampling bias, resulting in indicative findings at best. In addition, all participants were students from the RPTU Kaiserslautern-Landau, thus generally only having access to lectures from the university which can influence preference based upon lecture experience.

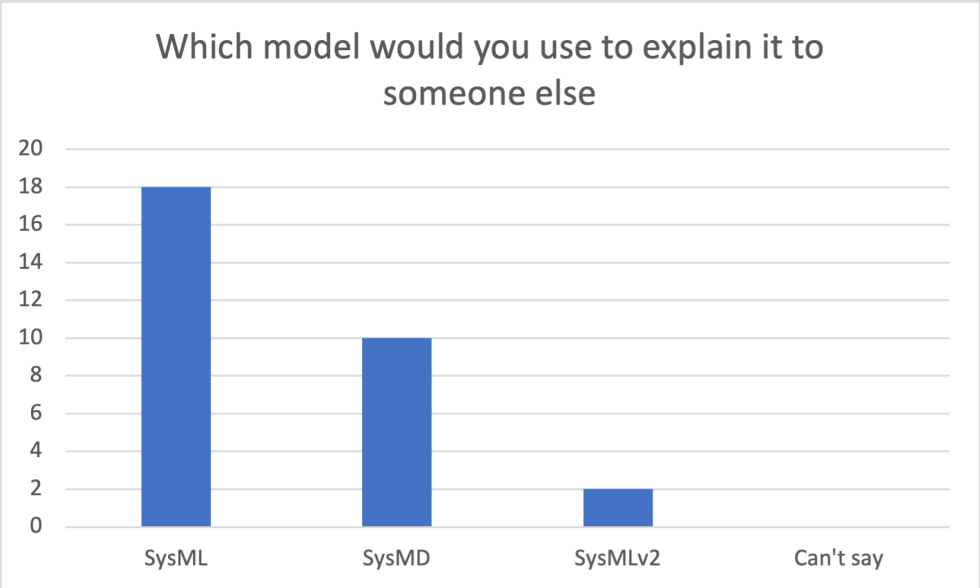

Figure 4.14.: *Rating of which of the three given modelling languages (SysML, SysML v2 and SysMD) provides models best suited to explain to others.*

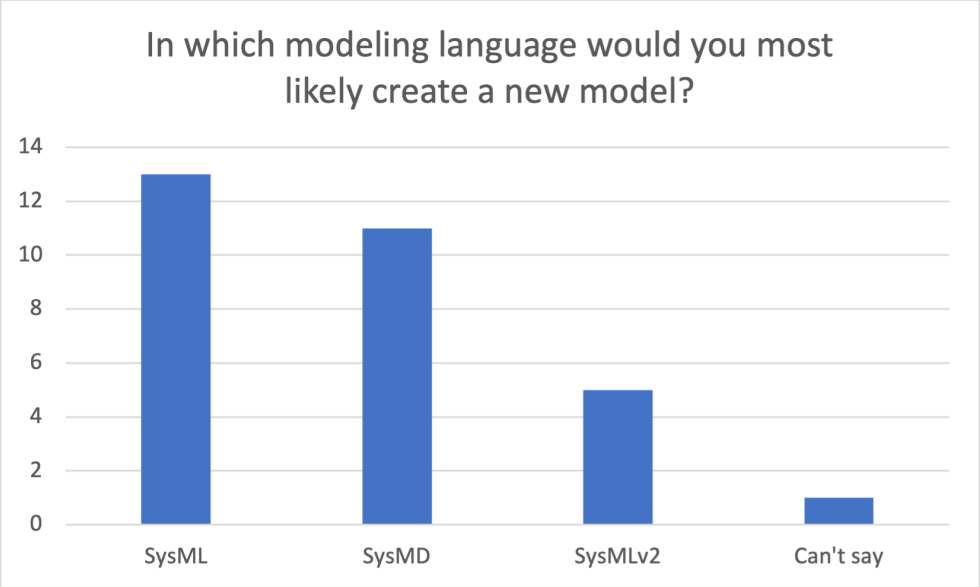

Figure 4.15.: *Rating of which of the three given modelling languages (SysML, SysML v2 and SysMD) would be preferred to create a new model in.*

4.3. INCOSE Survey

4.3.1. Setup

The second study was an online survey as well. This survey was targeted at professionals to receive feedback from their perspective. This survey was

designed to identify the current challenges in systems engineering to evaluate how inclusive systems engineering can tackle these challenges. This study was aimed at people interested in systems engineering or working in a systems engineering field. The goal of this survey was to identify pressing issues in systems engineering and to identify commonly requested features for future systems engineering tools from not only an expert but also beginner perspective.
The survey was shared predominantly with INCOSE and OMG systems engineering groups to collect data from people interested in systems engineering and with varying levels of experience.

4.3.2. Hypothesis Formulation

Similarly to before the focus on textual and graphical representation of SysML v2 suggests the use for different purposes. To validate the results of the student survey this is included in the INCOSE survey as well. With the aim of MBSE to facilitate stakeholder integration it is important to identify stakeholder integration, especially domain expert integration, as an important issue faced by system engineers. It is expected that professionals value the ability to communicate specifications and the existence of features going beyond descriptive modeling, such as verification and validation of models, highly to solve complex tasks whereas beginners struggle with the complex syntax which is already known to professionals.

Hypothesis (H5) Systems engineering has a high entry barrier.

Hypothesis (H6) Domain experts need to be better integrated into the systems engineering process.

Hypothesis (H7) Model validation and verification is valued more with increased systems engineering experience.

Hypothesis (H8) Modelling language syntax simplification is valued higher by participants with less MBSE experience.

Hypothesis (H9) Graphical model representation is generally preferred for visualisation purposes compared to a textual representation.

Hypothesis (H10) Textual model representation is generally preferred for creation purposes compared to a graphical representation.

Hypothesis (H11) Keeping documentation and models consistent throughout the modelling process is an important issue.

Hypothesis (H12) The ability to create models automatically is important.

Hypothesis (H13) The ability to have configurable abstraction levels is important for professionals to communicate the specifications.

Hypothesis (H14) The ability to analyse expressions in a model is valued primarily by experts.

Metadata

In the first section we collected some metadata about the participants. The only metadata which was collected is the age range, expertise in systems engineering and model-based systems engineering in particular, as well as the kind of domain and tool experience possessed.
The survey got 29 responses showing a wide spread of participants in age (see figure 4.16) and systems engineering expertise as shown by figures 4.17 and 4.18.

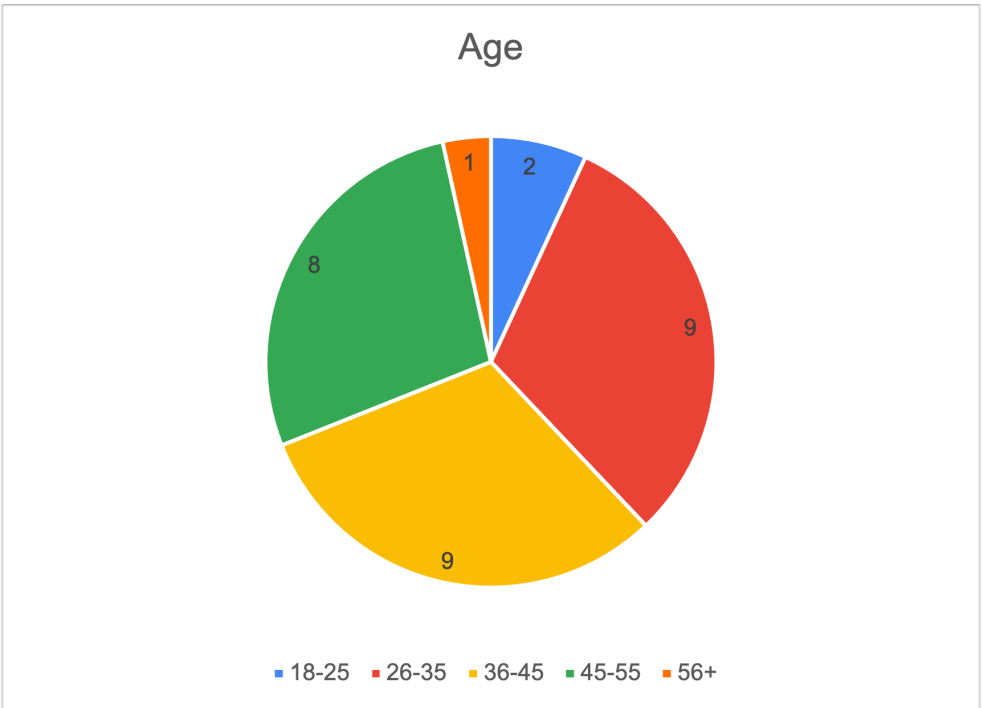

Figure 4.16.: *Age of participants.*

The vast majority of participants in the survey were between 26 and 55 years old, see 4.16 for the exact number of participants per age.

The systems engineering experience was highly spread out, with four participants indicating no systems engineering experience. It was to be expected that most participants at least know the basics of systems engineering and a large part working professionally in systems engineering due to the platform used to share the survey (see 4.17. Experience in model-based systems engineering doubled the number of people with no experience and shows clearly that most people who worked in model-based systems engineering projects do so professionally, with only two participants indicating to have worked in model-based systems engineering projects but not having professional experience (see figure 4.18). This can be tentatively interpreted as supporting the claim of systems engineering moving from document-based towards model-based, adopting the mode-based approach.

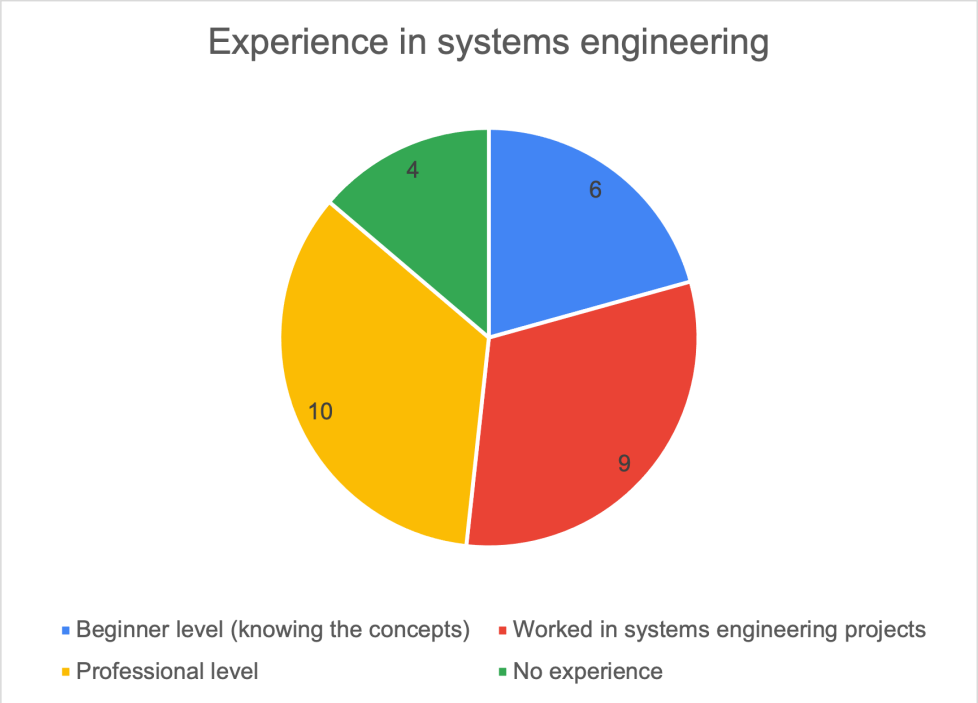

Figure 4.17.: *General systems engineering experience as indicated by the participants.*

Figure 4.19 shows clearly that most participants have software development expertise as well as expertise in electrical engineering and automotive domains. It is important to note that participants were able to state their expertise in a text-field and check multiple answers. However, the spread of domain expertise is shown by two participants from a medical domain.

As expected, three quarter of participants had experience in Excel with the six participants claiming no experience in Excel likely having used the tool before but not in any systems engineering context (see figure 4.20. Dedicated general purpose systems engineering tools such as PREEvision, Enterprise Architect, the Eclipse Modeling Framework and Polarion are surprisingly unknown by the participants of this survey, especially when filtering the data for people knowing both the Eclipse Modeling Framework and Enterprise Architect, which accounts for four of five people knowing the Eclipse Modeling Framework.

Systems Engineering Challenges

The second section starts with a general question as how approachable the participants perceives systems engineering to be for non-experts on a scale of 1 being very hard to get into to 5 corresponding to it being very easy to get into.

Afterwards, 16 challenges are presented which the participant has to rate separately on how important it is to find solutions to the particular challenge.

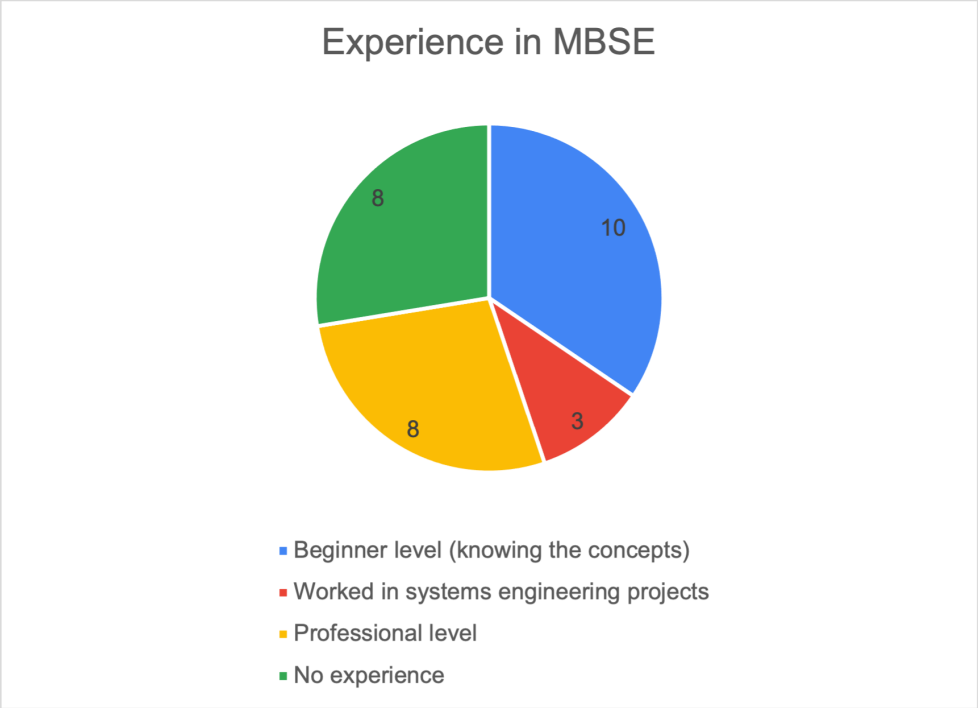

Figure 4.18.: *General model-based systems engineering experience as indicated by the participants.*

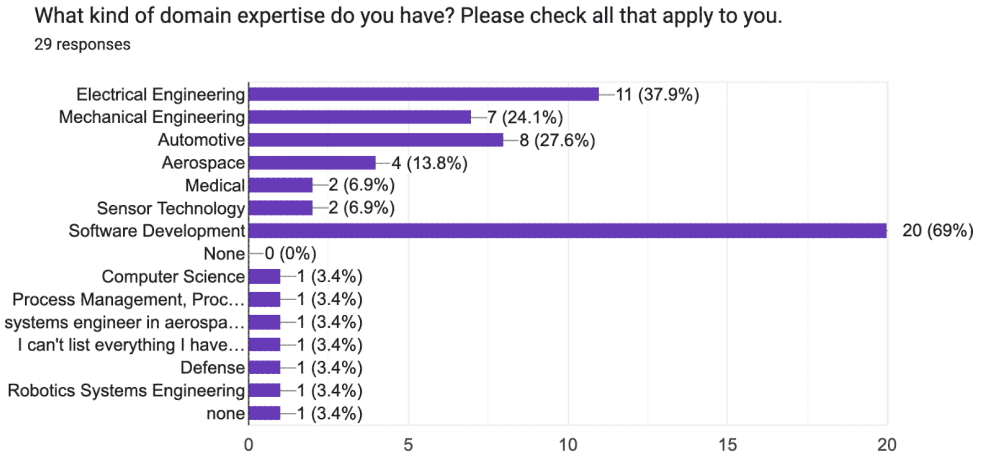

Figure 4.19.: *General domain experience as indicated by the participants.*

These challenges are:

- Late change requests
- Low stakeholder participation

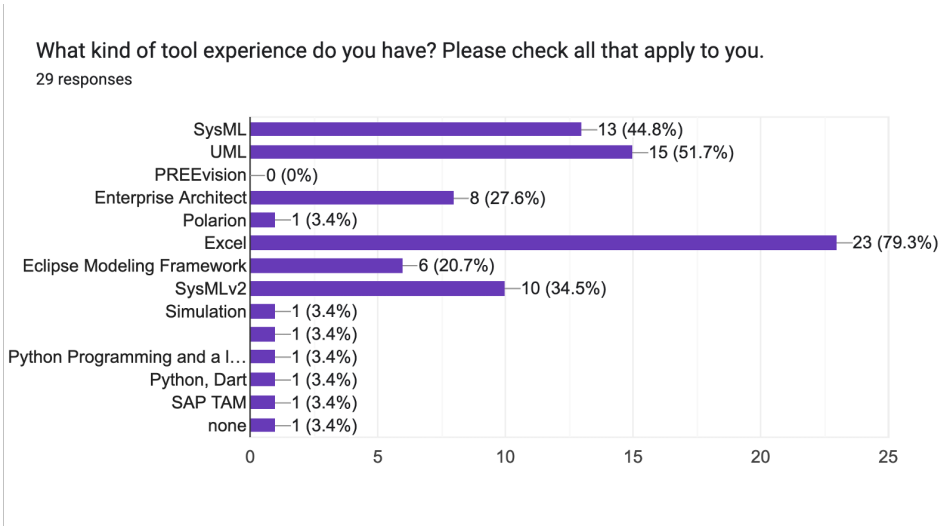

Figure 4.20.: *Tool experience as indicated by the participants.*

- Difficulty with modelling language
- Unintuitive semantics of modelling language
- Outdated documentation
- Misunderstanding between stakeholders
- Missing domain-expert input
- Incomplete or ambiguous requirements
- Missing tool integration
- Outdated/Inconsistent models
- Model validation
- Model verification
- Missing knowledge basis
- Misinterpretation of models
- Missing reference material
- Informal models

All of these are ranked individually on a scale of *very important* to *very unimportant*

Important Features

The third section is similar to the second by design but aims to identify important features for systems engineering. Again, these are ranked from *very important* to *very unimportant* individually. These features are:

- Documentation and model connected (linked/one place)
- Shared knowledge basis
- Version management system
- Simplified syntax for common entities and relationships
- Stakeholder collaboration
- Graphical model representation for creation
- Graphical model representation for visualisation
- Textual model representation for creation
- Textual model representation for visualisation
- Tool independence
- Support incomplete models
- Commonly accepted methodology
- Accessible reference models

Features Beyond Descriptive Modelling

The forth and final section presents four features that go beyond descriptive modelling of a system. These features provide additional functionality and can be realised with extensions to SysML v2 or by means of additional tools. The four features to be ranked are:

- Automated model generation from natural language
- Analysis/Computation of expressions
- Automated consistency checking
- Configurable abstraction levels

4.3.3. Results and Evaluation

To evaluate the survey results I focus on five groups, depending on the indicated systems engineering expertise. The groups are:

- All: All participants are included in this group.
- Professionals: This group only consists of participants who have indicated a professional level expertise in systems engineering. Ten participants are included in this group.
- Intermediate: This group only consists of participants who have worked in systems engineering projects before. Nine participants are included in this group.
- Beginner: This group consists of the participants who have indicated to know the basic concepts of systems engineering without having worked in such projects before. This group consists of six participants.
- No experience: This group consists of the participants who indicated no experience in systems engineering at all. This group only consists of four participants.

To be able to compare the results between the groups a score is created for each group by calculating the average importance score for each question over each group. The answers correspond to a score with:

- *very important* ≡ 5
- *important* ≡ 4
- *neither important nor unimportant* ≡ 3
- *unimportant* ≡ 2
- *very unimportant* ≡ 1
- *can't say* ≡ 0

The option *can't say* introduces missing data into the data set. As this set is very small a listwise deletion, thus removing all participants completely which have chosen a *can't say* response is to be avoided. Thus the feasible options are pairwise deletion, simple imputation or multiple imputation.
Figures 4.21, 4.22 and 4.23 show the difference of simple imputation with the assumption of *can't say* ≡ 0; *can't say* ≡ *neither important nor unimportant* and pairwise deletion respectively. This shows that treating *can't say* as an *neither important nor unimportant* answer is rather similar to the pairwise deletion method. Due to the vast majority of answers being either *important* or *highly important* due to including mostly generally known systems engineering issues the simple imputation method to treat the missing data as the middle value leads to a slightly more conservative result. The option of *can't say*

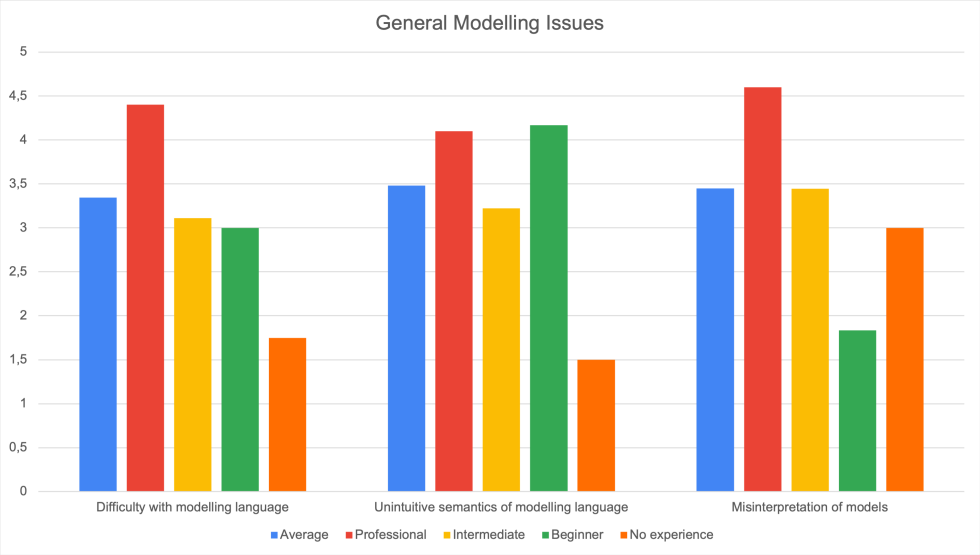

Figure 4.21.: *Perceived importance of general modelling issues on a scale from* very unimportant *to* very important *depending on expertise with* can't say *treated as less than* very unimportant

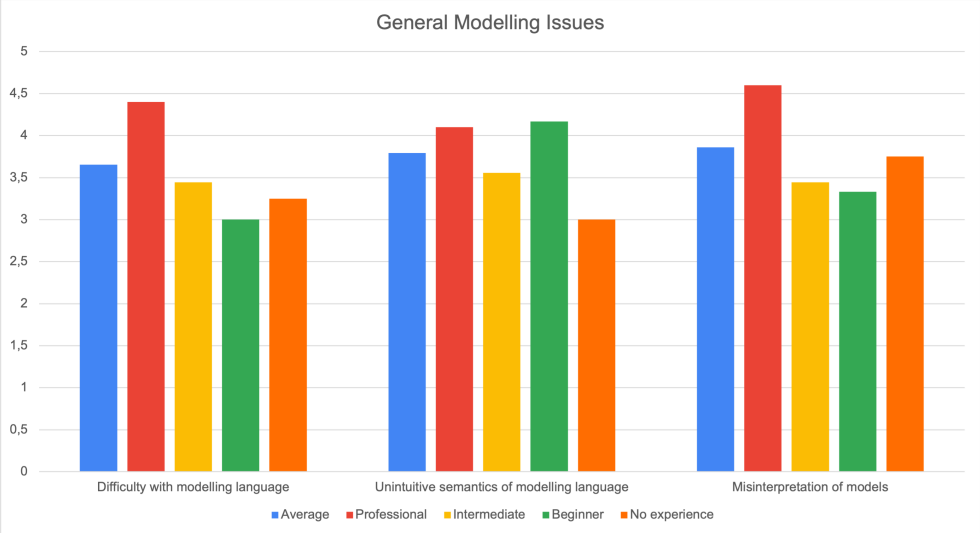

Figure 4.22.: *Perceived importance of general modelling issues on a scale from* very unimportant *to* very important *depending on expertise with* can't say *treated as less equal to* neither important nor unimportant

being treated as less than *very unimportant* provides the most conservative approach, especially considering the respondents with no systems engineering expertise.

For the remainder of the thesis *can't say* has been chosen with a score of zero chosen to keep the number of answers constant for each question to keep the number of participants as high as possible. This provides a highly conservative importance score in order to reduce the impact of individual answers due to

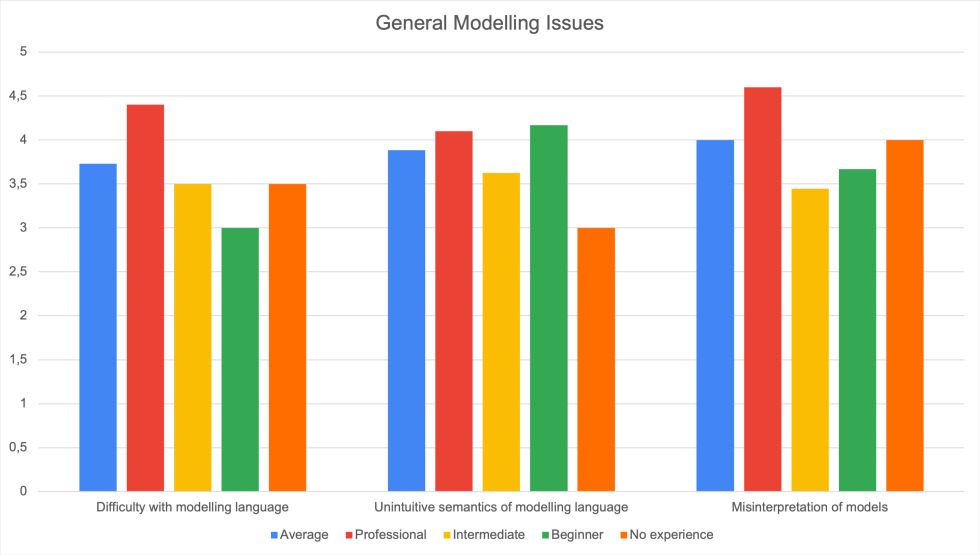

Figure 4.23.: *Perceived importance of general modelling issues on a scale from* very unimportant *to* very important *depending on expertise with* can't say *responses deleted from the response data*

the limited number of participants and to highlight that the evidence provided is only tentative. As most *can't say* responses came from the people with no systems engineering experience this group is only considered very lightly during the rest of the evaluation.

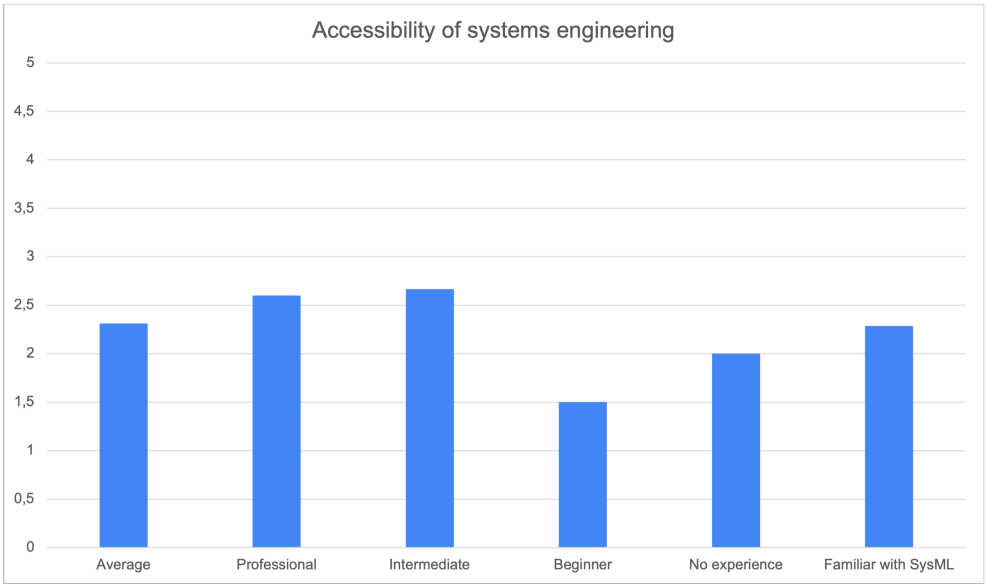

Figure 4.24.: *Perceived accessibility of systems engineering. Scores where possible from 1 ≡ very hard to get into up to 5 ≡ very easy to get into.*

Figure 4.24 illustrates the perceived accessibility of systems engineering in general with a low rating corresponding to systems engineering being very hard

to get into. Participants with a beginner level of systems engineering expertise ranked it as the least accessible, while more professional participants rated it more accessible. Overall, systems engineering is still perceived as rather hard to get into, with professionals still leaning slightly towards it being less accessible approach. Interestingly, experience in SysML does not change this perception.

The most pressing issue identified in this survey, with an average of 4,24 is the existence of *incomplete or ambiguous requirements* as shown in figure 4.25. This issue has been rated as very important or important by all but three participants. This has also been reported multiple times throughout literature, e.g. [101] and [53]. Other general systems engineering issues shown in figure 4.25 are *outdated documentation* and *late change requests*, both known problems in systems engineering and deemed important by the participants. Both of these can be connected to the incomplete requirements, stressing the importance of thorough requirements analysis, preferred with domain-expert participation.

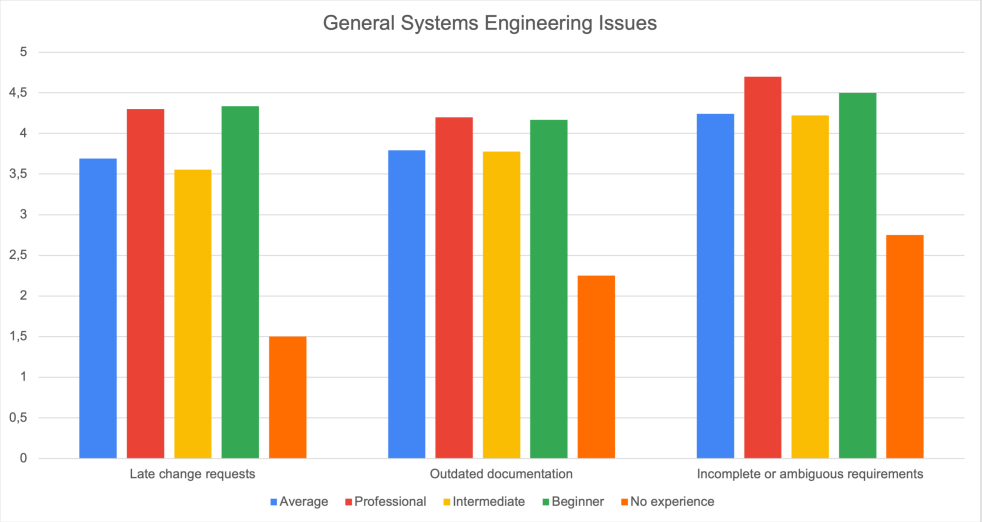

Figure 4.25.: *Importance of issues regarding general systems engineering issues as experienced by participants with an beginner level of expertise.*

Figure 4.26 highlights an interesting result in the perceived mismatch between *misunderstanding between stakeholders* and *low stakeholder participation*. While the potential of misunderstandings is considered to be important overall and by all participants with at least basic systems engineering expertise the issue of low stakeholder participation is not considered as important. This could be due to stakeholder participation already being high, which would be somewhat go against the importance of domain-expert input. This mismatch might be a result of the limited number of participants, however, the fact that this mismatch is almost nonexistent when considering only the professional participants we deem this important to point out as a potential further research topic.

Figures 4.27 and 4.28 highlight model related issues. The issues considered

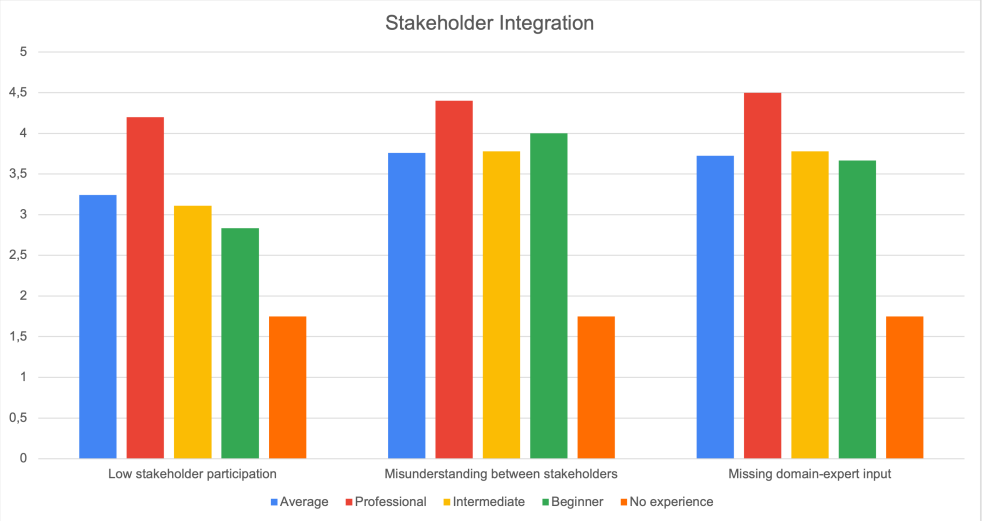

Figure 4.26.: *Importance of issues regarding stakeholder integration as experienced by participants with an beginner level of expertise.*

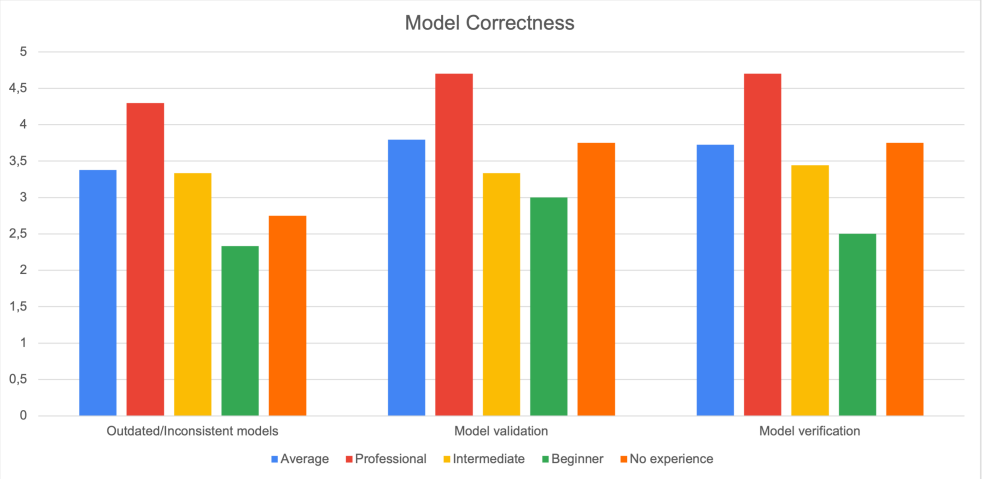

Figure 4.27.: *Importance of model correctness related issues as considered by the participants.*

as the most important in regards to modelling overall are *model validation*, *model verification* and the issue of *unintuitive semantics of modelling languages*. Other issues directly related to models are mostly considered to be somewhat important, but less important compared to general issues or other model related issues. The least important issues overall are the *existence of informal models* and *missing reference models*.

Figure 4.29 shows the preference of participants regarding model representation in either textual or graphical form and in which context each representation is preferred. There is a clear preference for graphical representation for visualisation purposes. This is intuitively true, however, this survey also shows the importance for a textual representation in order to create models. This

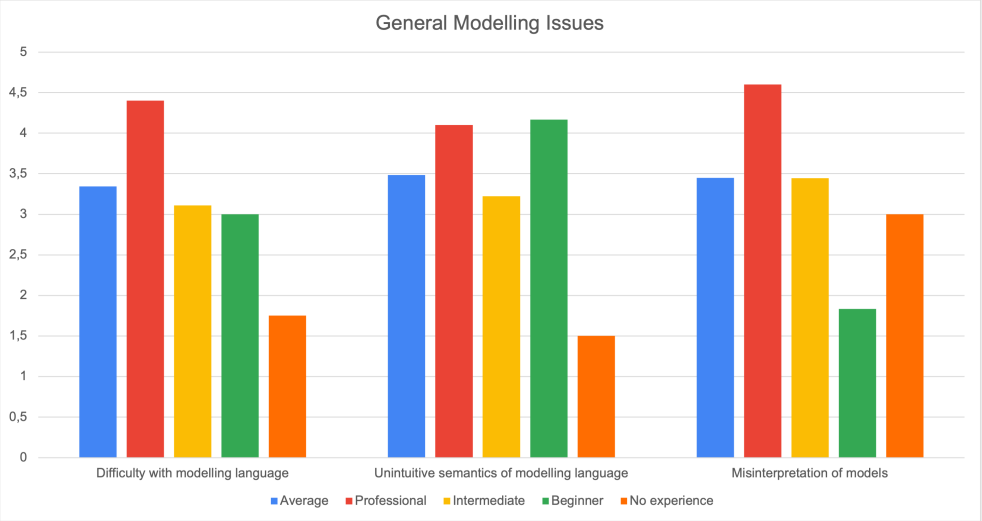

Figure 4.28.: *Importance of general model related issues as considered by the participants.*

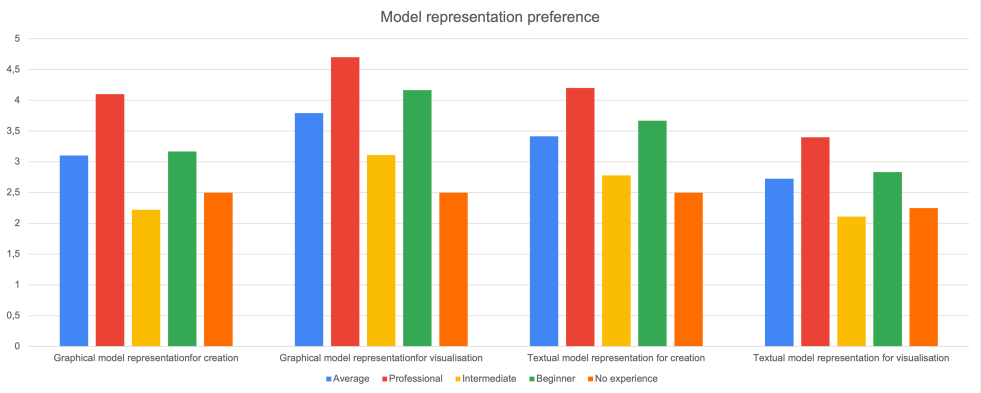

Figure 4.29.: *Importance of model representation as perceived by participation as experienced by professionals.*

indicates an importance for new modelling languages to support both graphical and textual representation for a given model, preferably with automatic translation between both.

When comparing the tool specific features, stakeholder integration features and accessibility features, illustrated in figures 4.30, 4.31 and 4.32 respectively, overall the tool specific features are deemed the most important. This shows the importance of sophisticated tools for systems engineering, offering features such as version management systems and make documentation easy by linking models and documentation. Facilitating stakeholder collaboration, for example by simplification of modelling syntax to support stakeholders create rudimentary models and providing a commonly accepted methodology is deemed more important than providing accessible references or other accessibility related features.

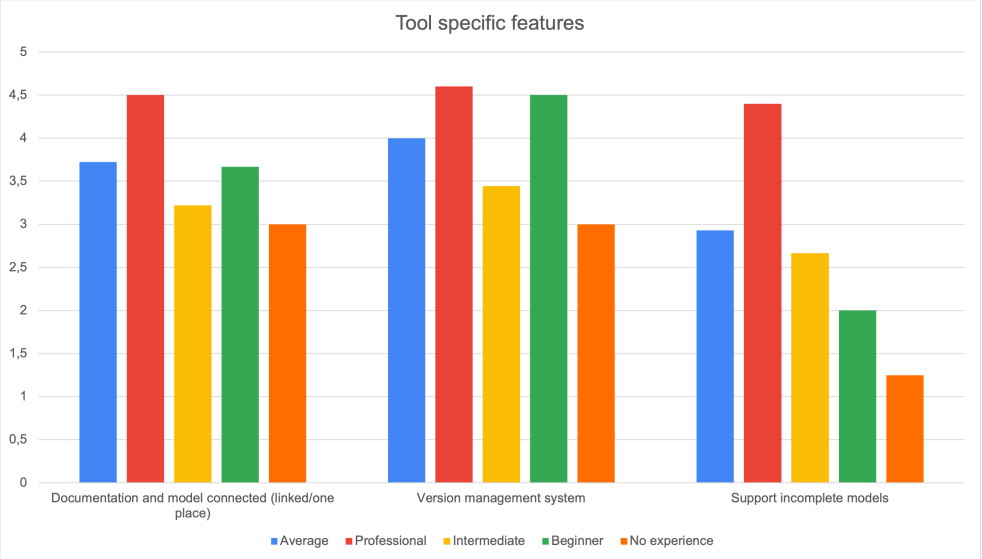

Figure 4.30.: *Importance of features which are tool specific in nature.*

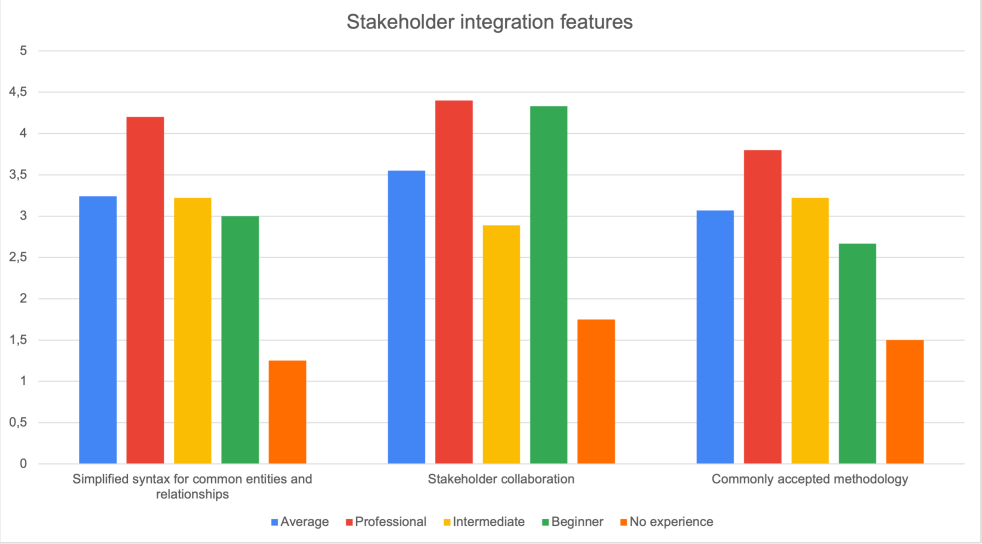

Figure 4.31.: *Importance of issues regarding stakeholder features as experienced by professionals.*

Figure 4.33 illustrates the importance of automated consistency checking in MBSE, being seen as an important feature which goes beyond pure descriptive modelling capabilities.

Professional Level

Participants with a professional level of expertise in systems engineering valued the missing domain-expert input as high as the incomplete or ambiguous requirements with both being rated very with 4,5 and 4,7 as an average score

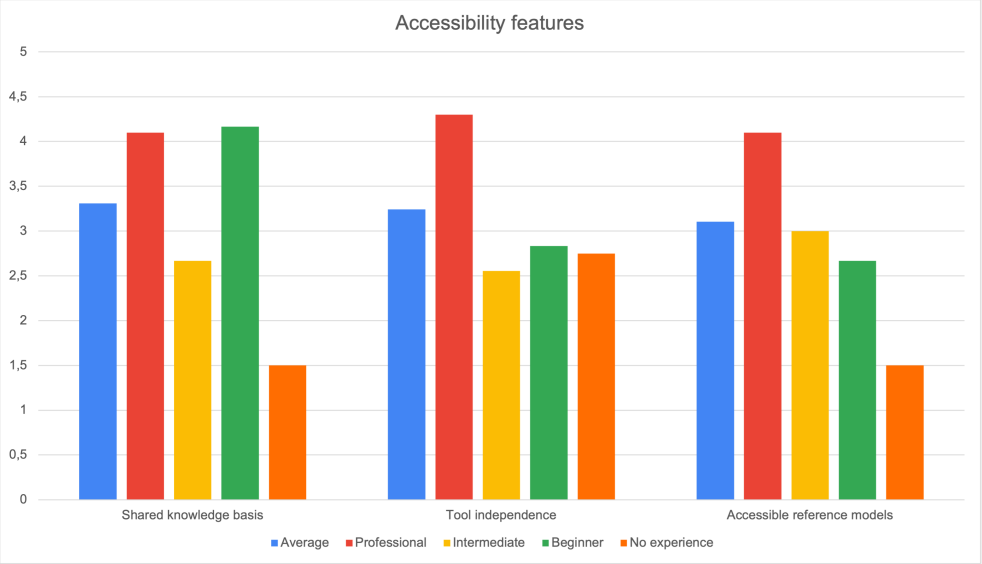

Figure 4.32.: *Importance of features beyond descriptive modelling as experienced by professionals.*

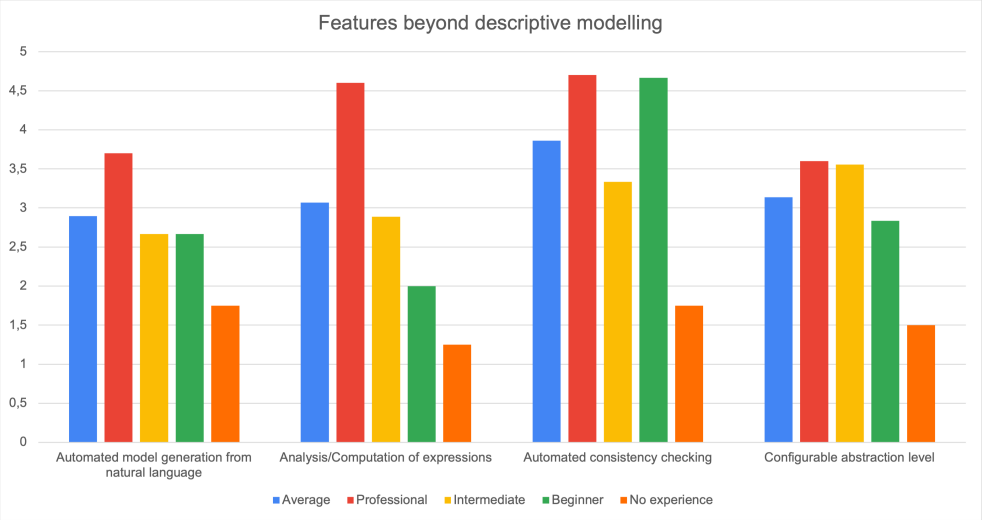

Figure 4.33.: *Importance of features beyond descriptive modelling as experienced by professionals.*

respectively. Both model verification and model validation scored an average of 4,7 making it be rated as important as the issues regarding requirements. Two more rather interesting issues are low stakeholder participation and difficulties with modelling language, which are both rated by professionals as important with an average of 4,1 and 4,3 respectively whereas they are rated as neither important nor unimportant on average including all participants with 3,2 and 3,3 on average.
It is important to point out that the mismatch considering stakeholder participation and domain-expert input is not existent in professional level partic-

ipants. Features going beyond descriptive modeling has two clearly requested features from professionals, being analysis and computation of expressions and automated consistency checking, being rated with 4,6 and 4,7 respectively. The configurable abstraction levels, which scored overall more points than the analysis and computation of expressions, was rated only with average of 3,6. Figures 4.34, 4.35, 4.36, 4.37, 4.38 give a visualisation of the discussed data in a more detailed manner.

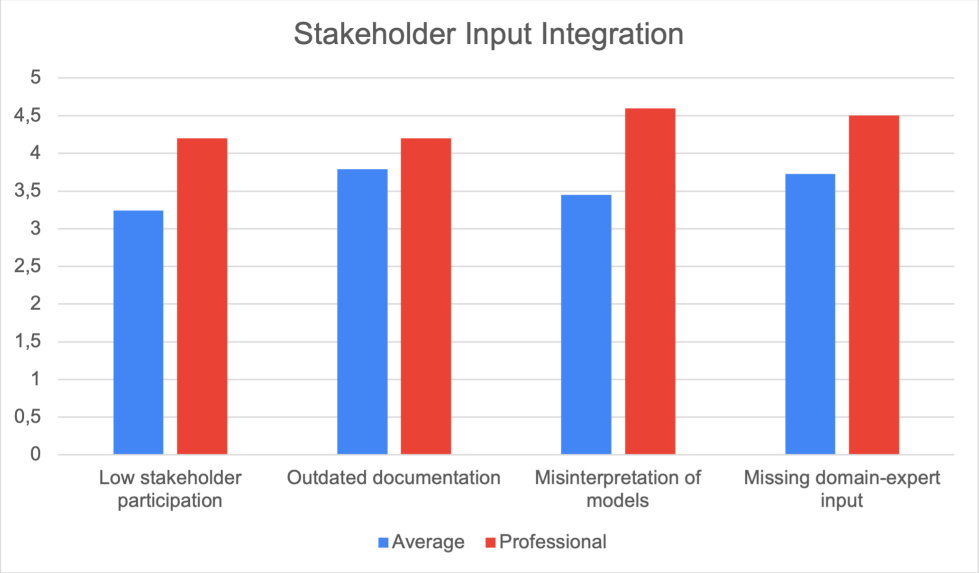

Figure 4.34.: *Importance of issues regarding stakeholder participation as experienced by professionals.*

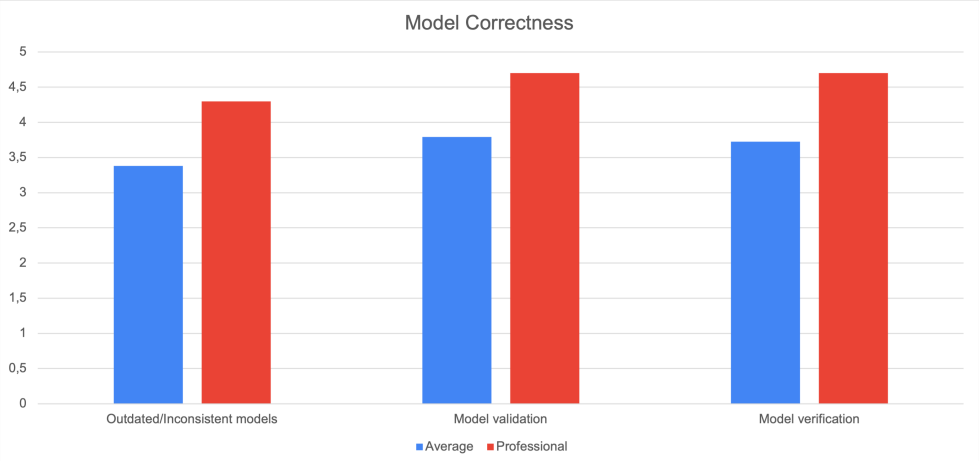

Figure 4.35.: *Importance of issues regarding models as experienced by professionals.*

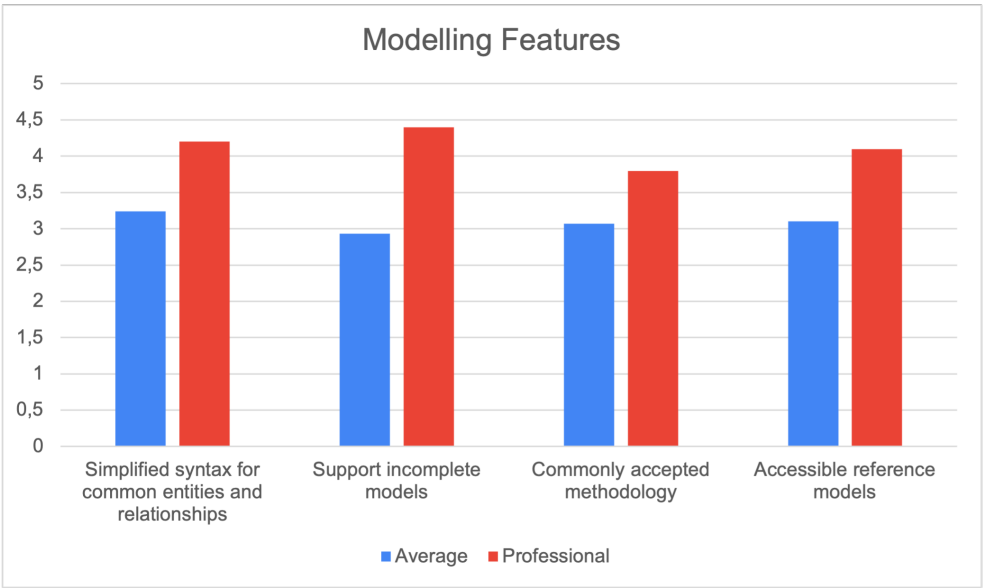

Figure 4.36.: *Importance of features of modelling as experienced by professionals.*

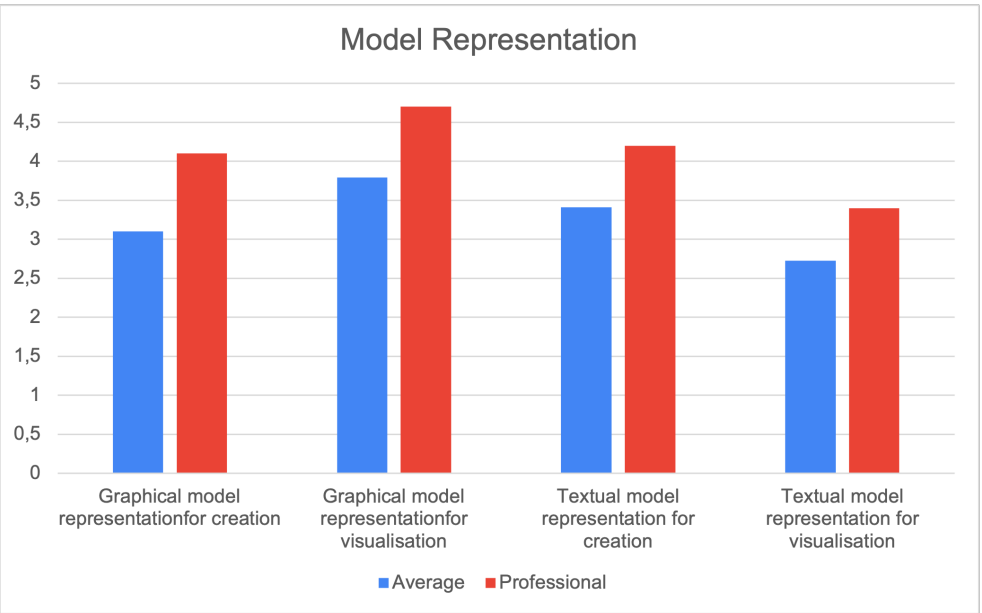

Figure 4.37.: *Importance of model representation as perceived by participation as experienced by professionals.*

Intermediate Level

The group of participants who worked in some systems engineering projects and thus are considered to be of intermediate expertise rated the issues and features overall lower than the average. This is due to the choice of rating *can't say* as zero instead of discounting it all-together. Only the incomplete and ambiguous requirements were rated as important on average with 4,2.

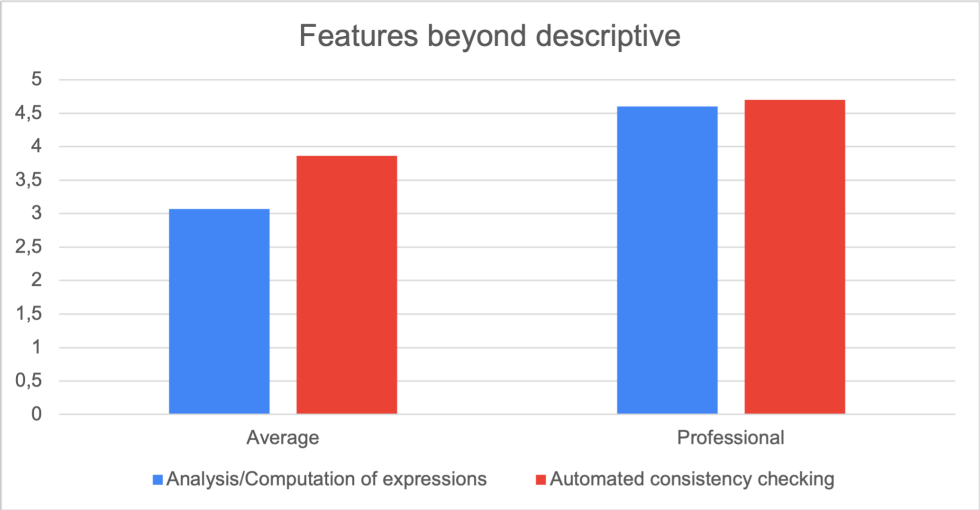

Figure 4.38.: *Importance of features beyond descriptive modelling as experienced by professionals.*

Missing domain-expert input, outdated documentation and misunderstandings between stakeholders were all scored with an average of 3,77. This shows that for intermediate level participants the integration of stakeholder input is considered more important than issues inherent to systems engineering.
Considering requested features the version management scored 3,44 on average with connection of documentation and model, simplified syntax and a commonly accepted methodology scored 3,22 each. This highlights the need for collaboration between stakeholders as well.
Intermediate level participants score the ability to configure abstraction levels higher than each other feature that goes beyond the descriptive modeling. This feature supports collaboration by providing means to create different views of a model to present.

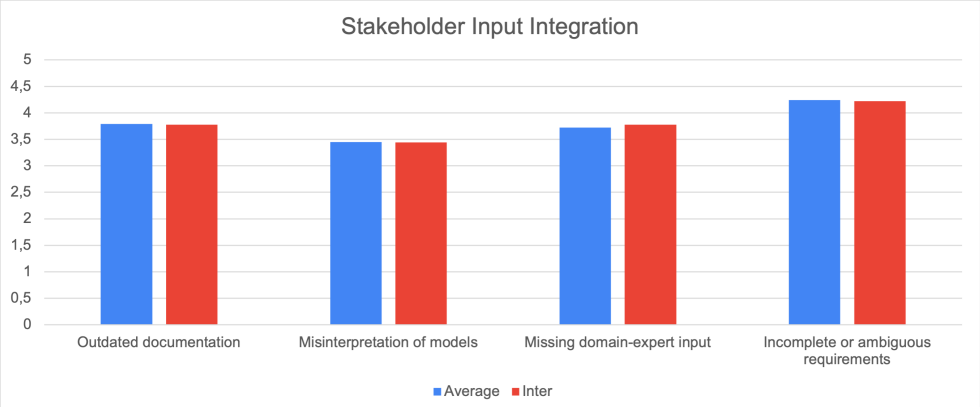

Figure 4.39.: *Importance of issues regarding stakeholder participation as experienced by participants with an intermediate level of expertise.*

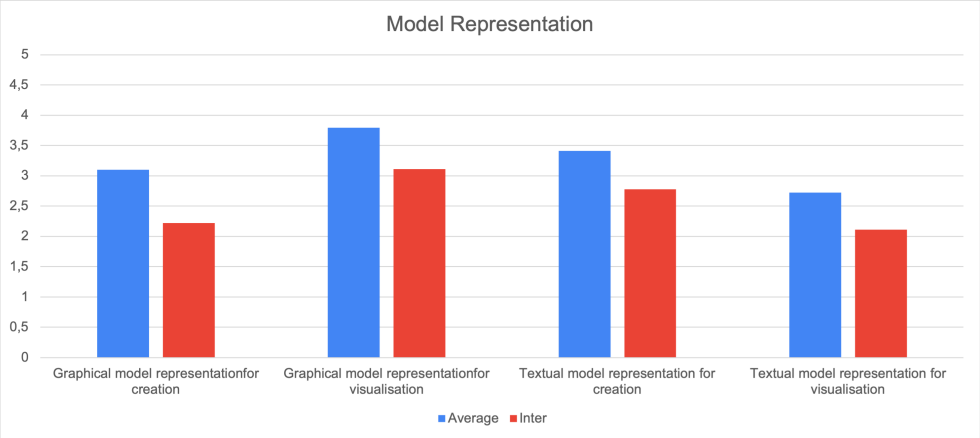

Figure 4.40.: *Importance of issues regarding model representation as experienced participants with an intermediate level of expertise.*

Beginner Level

Six participants are at a beginner level of systems engineering, knowing the concepts but have not yet worked in systems engineering projects. Problems with incomplete requirement analysis are still rated the most pressing issue with a rating of 4.5. Late change requests were rated with 4,3 points and almost as pressing. Interestingly, outdated models and unintuitive syntax and semantics have been rated with 4,16 points on average as more important than missing stakeholder input or misunderstandings between stakeholders. This shows that the syntax poses a high entry bar which can prevent people from starting to get into systems engineering.
For important features a version management system (4,5) and stakeholder collaboration (4,3) are identified as most important. A graphic representation for visualisation is also important with 4,16 and ranked higher than a textual representation for creation with 3,66. However, both are ranked higher than a graphical representation for creation and textual representation for creation, showing a preference of creating something in textual form and being able to transform it into a graphical form for visualisation purposes, see figure 4.41.

No experience

Only four participants had no experience in systems engineering. However, they overall valued model verification and validation as important with an 3,75 average. Additionally they identified misinterpretation of a model as an important issue.
For features they valued linked documentation and version management as important features with 3 points on average each. However, keep in mind that an extremely conservative imputation has been made considering *can't say* responses, which were most common among this group.

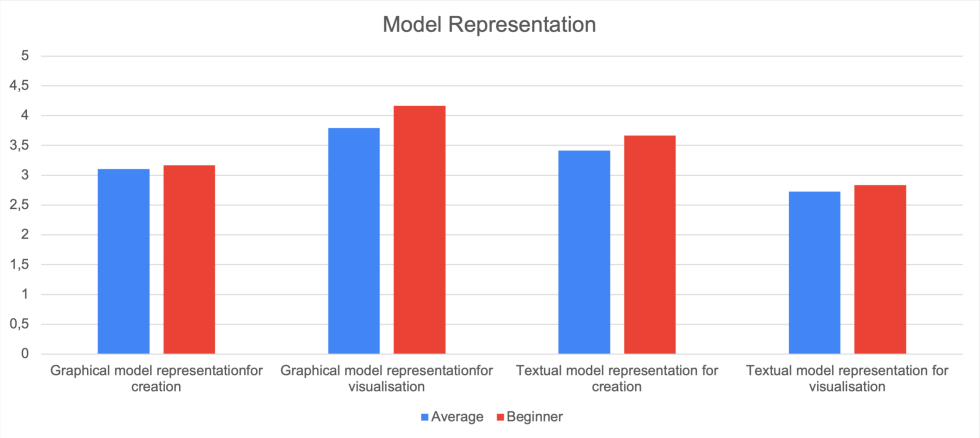

Figure 4.41.: *Importance of issues regarding model representation as experienced participants with an beginner level of expertise.*

Between Level comparison

One interesting result comes from comparing the estimation of accessibility of systems engineering. The participants with professional expertise in systems engineering rate it as easier to get into (on average 2,6) than the ones with only beginner level expertise (on average 1,5) as already shown in figure 4.24. Furthermore, it is clear that professionals value model validation and verification much more highly than intermediate and beginner level participants. Additionally, they also value missing domain expert input and a missing knowledge base much more highly than the other two groups. The issue these three groups agree upon the most is the issue of incomplete and ambiguous requirements. The only issue which beginners rate more important than the other two groups are the unintuitive semantics of modelling languages which is in line with rating systems engineering as harder to get into than professionals.
When considering requested features the version management system is requested highly by all groups. For model creation both beginners and intermediate would prefer a textual representation whereas professionals slightly prefer a graphical representation, which can be explained by being accustomed to it. For model visualisation all groups clearly prefer a graphical representation to a textual one. As to be expected, professionals overall rate features as more important, having more experience to see value in all proposed features. Automated consistency checking is valued highly by all three groups, even though the model validation and verification was only deemed very important by professionals.

4.4. Hypothesis evaluation

- Hypothesis H5 is clearly supported as shown in figure 4.24. This is especially true for participants with only a beginner level systems engineering

experience.

- Hypothesis H6 is supported as illustrated by figure 4.26 showing missing domain expert input being rated especially important by professionals but highly by all groups.
- Hypothesis H7 is supported as illustrated in figure 4.27. Both model validation and verification are rated as highly important by experts. However, both are among the most highly rated issues for participants without systems engineering experience and also rated highly by intermediate level participants. This shows that model validation and verification are deemed as important issues generally, with no clear connection between expertise and importance rating. This is further supported by figure 4.33
- Hypothesis H8 is not supported. Both figures 4.28 and 4.31 show no clear connection between expertise and importance rating for simplification of syntax. Instead it is rated highly among all participants.
- Hypothesis H9 is supported by figure 4.29 showing a clear preference for graphical model representation over textual representation for visualisation purposes.
- Hypothesis H10 is not clearly supported. Figure 4.29 indicates only a minor preference of textual model representation for creation purposes among professionals. The hypothesis is supported when only considering intermediate and beginner level participants.
- Hypothesis H11 is supported. Figure 4.25 and 4.30 both show the general importance of documentation issues in systems engineering.
- Hypothesis H12 is not supported. Figure 4.33 shows the feature of automatic model generation from natural language is not seen as important.
- Hypothesis H13 is not supported generally. Figure 4.33 shows that configurable abstraction levels are valued more highly by intermediate level participants compared to other issues and features when considering only this group of participants. In general, the feature of configurable abstraction levels is not seen as important.
- Hypothesis H14 is not generally supported. However, figure 4.33 shows the ability to compute expressions is valued highly by professionals.

4.5. Overall evaluation

Evaluating both studies one can infer a need to simplify the systems engineering process to deal with ever more complex systems spanning across ever more domains. The surveys show that depending on the systems engineering expertise different features are important to provide. Especially when considering

syntax and user guidance the more proficient participants have less need for it than less proficient ones. However, all participants have shown a preference for a graphical model representation in order for visualisation and a textual representation for creation purposes. This is unsurprising, but highlights the need for future systems engineering languages, such as SysML v2 to keep both a textual and graphical representation in mind, both with distinct and important purpose of their own.
The importance of domain expert and stakeholder input is clearly shown, both directly and indirectly as comprehensive requirements are only possible when all stakeholders work together. Another feature that is generally highly requested are means to keep documentation up to date to represent changes in design, both by linking it with system models and by providing a meaningful version management as an additional way to make changes visible.
When considering features and issues important for professional participants the need of model verification and validation and automated consistency checking stands out.
Professional systems engineers have to deal with ambiguous and incomplete requirements and are missing tools which are able to validate and verify models efficiently. They require sophisticated tools providing features like automated consistency checking and computation of expressions to quickly check the created models.
Intermediate level participants require support in stakeholder collaboration by means of a common methodology, configurable abstraction levels in addition to model validation and verification.
Beginner level participants value a simplification of modelling syntax and general accessibility highly with less emphasis on model verification and validation features.

4.5.1. Limitations of the this survey

Due to the limited number of participants in our survey, the results should be interpreted as indicative trends rather than statistically robust findings. Furthermore, the survey was distributed via opt-in mailing lists from INCOSE and OMG, reaching only participants interested in systems engineering and being members of the OMG and INCOSE which does not represent the general systems engineering population. This is highlighted by figure 4.19 showing a high number of participants with software development and electrical engineering background, whereas the medical, defense and aerospace domains are underrepresented compared to studies such as the one by Akundi and Ankobiah with 75% of participants from the defensive or aerospace domain. These limitations are further increased when analysing a particular focus group or comparing the groups to each other. The preliminary findings of this work are supposed to provide further research topics instead of strong empirical evidence.
The inclusion of *can't say* with a score of zero provides a conservative estimate of the importance scores for each group. Removing *can't say* responses would give a more accurate result by giving more emphasis to each individual

answer. We chose to reduce the impact of each individual answer due to the limited number of participants.
No statistical analysis has been provided due to these limitations.

Chapter 5

Inclusive Systems Engineering

5.1. Objective

From the surveys discussed in chapter 2.2.5 it is save to assume that systems engineering is facing challenges in many different directions as further evidenced by the surveys from chapter 4. In order to provide more comprehensive and valuable requirements, identified as one of the key issues, it is important to involve domain experts and other stakeholders into the requirements analysis and following development phases, especially with today's systems become ever more complex. This gives rise to the need of new development methodologies which can be quickly picked up by non-experts to define system requirements and functions.
For this purpose, this work focuses on on simplification of systems engineering to make it more accessible to domain-experts. To do so the notion of inclusive systems engineering is introduced. Inclusive systems engineering aims to simplify the process of systems engineering in order to make it more approachable to non-experts, especially domain experts. It is an extension to systems engineering, and model-based systems engineering in particular.
The purpose of this simplification is to reduce the hindering factors to adopting MBSE by reducing biases such as loss aversion, status quo biases and foster biases such as social norm appeal as discussed in chapter 3.1.2. To achieve this inclusive systems engineering focuses on three parts:

- *User guidance*: Methods to ease new engineers through the process, fostering MBSE adoption by reducing training times and highlighting benefits

- *Process accessibility*: Focusing on a process that is easy to adopt, especially considering compatibility with domain specific tools and techniques.

- *Tool accessibility*: Features provided by specific systems engineering tools to facilitate easy transition, communication and adoption.

For each part general features are proposed and discussed. Inclusive systems engineering is proposed to facilitate communication between stakeholders and ease of adoption of MBSE, both shown to be highly important issues by the surveys discussed in 2.2.5 and 4.5.

5.2. Definition

In order to follow the systems engineering definition given by INCOSE in chapter 2.2 I have chosen to extend their definition by including the focus on domain experts and an approachable systems engineering approach.
Inclusive *Systems Engineering is a transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological and management methods* ***with the specific focus on domain experts. User guidance techniques, intuitive syntax and interactive approaches are used to make the systems engineering process as approachable as possible for non systems engineering experts.*** [38]

5.2.1. Inclusive Model-Based Systems Engineering definition

The ever increasing complexity of systems in all domains confront modeling experts with increasingly intricate modeling decisions. These decisions need to be made by modeling experts collaborating closely with the domain experts, which is time consuming and can still result in wrong decisions due to misunderstandings. In order to solve this issue and further advance MBSE we consider the inclusive systems engineering approach, aiming to assist domain experts in the systems engineering process.
We call this approach Inclusive Model-Based Systems Engineering, seeing inclusiveness as making the MBSE approach more accessible towards the majority of population, who is not familiar with the inherit computer science terminology and conventions usually presupposed for MBSE. We extend the notion of *MBSE is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.* [96] towards *Inclusive MBSE is MBSE designed to be utilised by domain experts, providing a gentle learning curve, intuitive syntax and methodology, combined with guidance methods.*

Combining the features of inclusive systems engineering and inclusive model-based systems engineering discussed in [40] and [38] respectively shows some features present in both approaches and unique ones depending on the approach, which is shown in 5.1 and split into three sections. The first section consisting of *simplified syntax*, *user guidance* and *integrated documentation* apply to any systems engineering approach. The second section, *tool independence*, *knowledge basis*, *version management*, *extendability* and *code generation* correspond to the broad, abstract, systems engineering concept and

are not necessarily applicable to model-based systems engineering. However, it is preferred to provide these features for MBSE as well. The third section, *graphic & textual representation*, *interactivity*, *incomplete model support*, *user specific views* and *standard compatibility* are particularly important for MBSE. In the following each category and feature are discussed to provide reasoning for inclusion, anticipated benefits and a general suggestion how to support the feature.

	Features	Benefits
INCLUSIVE SE & INCLUSIVE MBSE		
Simplified Syntax	Natural Language Limited # keywords	Reduced learning time Low entry Gradual introduction
User Guidance	Nudging Recommender Systems Feedback	Low entry Comprehensive Systems Domain expert focus
Integrated Documentation	Single File Documentation & Specification	Comprehensive Documentation Knowledge Transfer
INCLUSIVE SE		
Tool Independence	Compatible domain tools	Communication & Cooperation across domains
Knowledge Basis	Domain specific knowledge usable	Knowledge Transfer
Version Management	Version Control & Commit System	Iterative Design
Extendability	Basic solution Domain specific solutions	Common basis Cooperation
Code Generation	Code Extraction from documentation	Comprehensive domain models Complex systems from informal description
INCLUSIVE MBSE		
Graphic & Textual Representation	Automatic Transformation	Complex modeling in text Graphic models to share
Interactivity	Direct Feedback	Bug fixing Constraint Checking Possible Roadmaps
Incomplete Model Support	Abstract missing information	Top-Down & Bottom-Up Agile methods
User Specific Views	Abstraction of Information Domain/Purpose views	Abstract confusing information Task specific views User specific views
Standard Compatibility	Extend standard Multi-Tool support	Introduction to standard Simplified sharing

Table 5.1.: *Overview of all inclusive systems engineering features with exemplary features and expected benefits.*

Table 5.2 provides a revision of the features important for inclusive systems engineering. Inclusive systems engineering builds upon three connected pillars, namely *User Guidance*, *Process Accessibility* and *Tool Accessibility*. These pillars are interconnected but emphasise different aspects of the extension. User guidance focuses on the user throughout the process and tools used. The process accessibility focuses on means to reduce the beginner barrier for systems engineering in general and the tool accessibility focuses on the features a systems engineering or MBSE tool should provide in order to be accessible to

domain experts and beginners. Table 5.2 provides a brief overview of features for each pillar while the coming sections provide a discussion of these features themselves, providing sources and implementation suggestions.

Inclusive Systems Engineering		
User Guidance	Process Accessibility	Tool Accessibility
Helpful Error Messages Nudging Recommender Systems Persuasive Systems Design NLP Abstraction of unnecessary Information	Incomplete Models Graphical & Textual Representation Ontology & Knowledge Base Tool Independence Syntax Simplification Linked Documentation Stakeholder Specific Views	Standard Compatibility Extendability Interactivity Code Generation Version Control System Notebook Programming Integrated Documentation

Table 5.2.: *Three pillars of inclusive systems engineering*

5.3. User Guidance

Guiding users, especially new ones, through a particular systems engineering tool or the general process is important to make systems engineering more approachable. Systems engineering is generally considered to be hard to get into as shown in 4. A number of user guidance techniques can be used to make it easier. Most notably nudging, persuasive system design and recommender systems are suitable to guide a user throughout the systems engineering process when integrated into a tool. The foundation of these methods is discussed in chapter 3.2. User Guidance during the modelling process can make use of nudging, persuasive systems design, recommender systems or automatic code generation, see [37] for a review of possible guidance techniques. For modelling in particular automatic code generation, building a model from a given documentation, or using code to create documentation is especially relevant, however, other techniques are important to include as well. This can guide towards more comprehensive models, removing inconsistencies or facilitate more documentation.

5.3.1. Nudging

Nudging can be used directly during the systems engineering process to provide feedback of incomplete requirements, missing specifications or general common issues. Several different nudge techniques are suitable to be integrated in an systems engineering tool.
As briefly discussed in section 3.2 it is important to consider ethical implications of nudges. Employing nudging in a systems engineering context and influencing domain experts and systems engineers can be considered to be unethical due to selfish goals such as increased profit by the companies which create the systems. However, in general the methods provided in this work aim towards creating better systems with reduced effort and increased domain expert integration. The goal is to achieve a more accessible systems engineering method which provides benefits to those who will use the finished products and to reduce stress on those creating these systems. It is important

to stress that each tool created following inclusive systems engineering has to be examined on its ethical justification.

Nudge	Nudge Mechanism	Nudge Category
Syntax highlighting	Increased salience of attributes Attracting attention	Decision Information
Affected variables highlighting	Increased salience of attributes Attracting attention Mapping of variables	Decision Information
Stakeholder specific views	Customised information Multiple viewpoints Explicit mapping	Decision Information
Scenario specific views	Customised information Multiple viewpoints Explicit mapping	Decision Information
Linked documentation	Change ease of Documentation access	Decision Structure
Drag & Drop features	Provide convenience	Decision Structure
Simplified & Advanced Syntax	Simplification Simplifying active choosing	Decision Information Decision Structure
Simplified & Advanced Error Messages	Simplification Simplifying active choosing Just-in-time support	Decision Information Decision Structure Decision Assistance
Direct Feedback	Just-in-time support Self-control strategies Consequences of actions	Decision Assistence
Syntax completion	Enhancing active choosing Simplifying active choosing	Decision Assistence
AI generated code	Enhancing active choosing Simplifying active choosing Partition of options Structure complex choices	Decision Assistence
Recommender System: Additional information	Customized information Feeling of reciprocity	Decision Information Social Decision Appeal
Recommender System: Model Reuse	Convenience Reduced effort	Decision Assistence
FAQ	Situated suggestions	Decision Assistence
Tutorial	Tutorial Reduced entry barrier	Decision Assistence
Tool suggestion	Anchoring & Default Option	Social Decision Appeal
"Industry standard" Information	Social reference point Following the herd	Social Decision Appeal

Table 5.3.: *List of possible nudges to be utilized in systems engineering with the corresponding nudge mechanism and nudge category.*

Table 5.3 gives a short overview of possible nudges and to which nudge mechanism and category according to Jesse and Jannach [71] they belong. Following is a brief introduction into the nudges and possible implementation suggestions.

Syntax highlighting: Syntax highlighting is a well-known technique in programming to highlight keywords and information on connected syntax to prevent easy mistakes such as misspelling of variables or keywords by highlighting them dynamically. Reijers et al. present a theoretical argument on the benefit of syntax highlighting, present a prototypical implementation and report

on a significant benefit of syntax highlighting for beginners when tasked to understand a model [106]. Syntax highlighting is commonly implemented in programming tools and does not require high effort to support.
Affected variables highlighting: A special case of syntax highlighting is the highlighting of defined variables and of variables affected by running a program. When a component is extended some definitions can have an affect on other components and the variables specified within. Highlighting such effects improves salience of the decision made, providing increased traceability of changes and their effects [22].
Stakeholder specific views: Abstracting unnecessary information can create views suitable for specific stakeholders such as domain experts from a specific domain. These views improve salience of important information by abstracting the rest.
Scenario specific views: Similar to stakeholder specific views the existence of scenario specific views provides more information on specific behaviour or certain specification depth.

Linked documentation: By linking documentation and specifications it is easier to anticipate effects a change will have and more likely the documentation will be updated accordingly. This provides additional structure for a decision to be made.
Drag & Drop features: Drag & Drop features are common in user interfaces, providing easy and intuitive means to change structure or add new information. Collomb and Hascoët [35] introduce methods to adapt Drag & Drop features towards multi-user environments which can further facilitate real-time collaboration of stakeholders.
Simplified & advanced syntax: Providing both a simplified syntax and advanced syntax can lower the entry barrier for beginners. This reduces the overhead for beginners to learn complex syntax but provides it when the need for it arises naturally with more experience.
Simplified & advanced error messages: Following a similar logic to the advanced and simplified syntax the error messages a compiler of a model in textual representation provides can easily overburden a novice user. A simplified version can provide easy suggestions whereas the advanced version can be beneficial for advanced users. This can highlight highly critical warnings and issues and abstracting less relevant ones for simplification [22].
Direct Feedback: Direct feedback can entail a number of different features, however, for MBSE specifically the ability to get validate and verify a model and receive direct feedback has been valued highly by all participant groups in the study discussed before as illustrated in figure 4.28. Additional feedback can consist of information highlighting as mentioned before or on where to find additional information to solve issues. An additional type of feedback is the setting of reminders to review models or specifications [6].
Syntax completion: Providing options for auto-completion of syntax is common practise in programming. It functions both as a tool for increased efficiency by reducing the amount of time spent on writing and provides a

reminder of possible options or required specifications. This can be further enhanced by using AI to suggest following syntax. While over reliance on such assistance is considered to be a potential issue, both the study by Takerngsaksiri et al. [120] and Valov and Buchalcevova [125] show positive effects on error reduction, efficiency and cognitive load in beginners and professionals as well [125].
AI generated code: AI powered tools can be used to either provide a natural language description of a model to support thorough documentation and collaboration as well as generate models and programs from a given documentation. AI enhanced syntax completion or providing well structured feedback tailored to the user offer abundant research opportunities.
Recommender System: additional specifications: Recommender systems can offer a number of different recommendation, such as to what specifications have been added in similar models to highlight potentially missing specifications or requirements by accessing reference models [71].
Recommender System: model reuse: By accessing reference models recommender systems can be used to provide suggestions what models or parts thereof can be reused in a given project to reduce the need of redundant specifications. Especially when dealing with a large number of similar models which differ in only a small number of specifications but require full specifications nonetheless [71].
FAQ: Providing access to a FAQ can provide hints and suggestions for users to follow or to solve issues without the need to use more sophisticated tools such as AI powered suggestions
Tutorial: Providing a meaningful tutorial is especially important for beginners to familiarize themselves with a given tool as well as the general systems engineering process, whether model-based or document-centric. Providing such a tutorial can be important for sophisticated and established tools as well in order to reduce the amount of support needed to be provided by the tool creator.
Tool suggestion: Suggesting a suitable tool is important when domain specific languages and tools are available. This can facilitate the use of specialised tools instead of relying on general solution tools or spending time on finding sub-optimal solutions in a domain specific tool not intended for the given purpose [15].
"Industry standard" information: Presenting a tool, domain specific language or methodology as the "Industry standard" provides a social norm which is likely to be followed by a user. A user will compare themselves to the norm set by the message and try to follow it to behave socially acceptable [15].

5.3.2. Persuasive Systems Design

To guide users along a process the PSD model provides a clear and structured model to follow for implementing user guidance techniques. The ability to elicit more comprehensive documentation has been shown by studies such as the one done by St-Maurice, Burns and Wolting [89]. This study pro-

vides empirical evidence for successfully influencing users to provide earlier data-entry in a clinical environment, which can be tentatively transferred to a more general ability to influence towards more comprehensive and timely documentation. In systems engineering this can facilitate the creation of more comprehensive models to ensure comprehensive requirements and specifications. The findings from Räisänen, Lehto, and Oinas-Kukkonen [105] discuss the PSD model in the context of evaluating software design specifications and provide evidence that the PSD model is suitable to increase the designs level of detail. This is further evidence that the PSD model can be used in a systems engineering context to provide more detailed requirements or component specifications from stakeholders.

5.3.3. Recommender Systems

Recommender systems can use reference material to suggest next steps, providing a specialized form of nudging. These steps can be additional requirements which are common in similar systems, the use of available data or components from earlier systems and more. These recommendations can quickly help to produce comprehensive requirements, suggest solutions to the posed requirements or help during later development processes. By use of both recommender systems and nudging a user, no matter new to systems engineering or an expert, can be guided towards comprehensive specifications and implementation of a new system. This is especially promising when considering domain experts who can determine the usefulness of recommendations thanks to their expertise while being guided through the complex systems engineering process.

5.3.4. Natural Language Processing

Natural language processing (NLP) is a subfield of artificial intelligence. NLP focuses on understanding, interpreting and generating human language by machines. It is supposed to bridge the gab between human communication and machine understanding and employs techniques from linguistics and machine learning. In the context of systems engineering the ability to process requirements and system descriptions from natural language is more important than the generation of natural language phrases.
To discuss NLP the work of Khurana et al. [78] is used as the foundation. First, words are reduced to their basic form and filler words are filtered out, as these do not convey direct information. The words are tagged according to their position in a sentence and then grouped into phrases which form clauses or sentences to analyse the syntactic level. This reveals structural dependency between words. This *parsing* identifies the phrases that convey more meaning when interpreted as a phrase instead of the individual words. This analysis also considers word order, stop and filler words and morphology omitted in the previous analysis. Afterwards the semantic level is analysed to determine the proper meaning of a sentence. As machines cannot rely on non-literate context clues such as inflexion of words or concepts present, multiple possi-

ble meanings of a sentence are processed by its logical structure to identify the most relevant words and the interaction among them. This level includes semantic disambiguation of words with multiple senses [83]. After syntactic and semantic analysis the level of discourse examines meaning derived from more than one sentence. It analysis connections among words and sentences across a document to ensure coherence. This removes possible interpretations identified during the semantic level to remove interpretations leading to an incoherent structure. This connects structures such as entities and pronouns which refer to them and is achieved by a process called co-reference. The last step of analysis is the integration of knowledge or content which comes outside of a document. This context can change the interpretation of a text significantly.
For a systems engineering tool this gives the opportunity to split a given documentation into tokens which are analysed to extract requirements or features.

5.3.5. Abstraction of unnecessary information

Abstraction of unnecessary information is done to simplify complex systems by focusing on essential elements and details. This is critical for managing interdisciplinary complexities and provide stakeholders with the information important for them without overwhelm them with information unknown and unnecessary for their work. What can be considered as unnecessary information is dependent on the scenario which needs to be represented or the stakeholders involved. For scenarios on agent behaviour the scenario description language Q abstracts agents internal mechanisms and only focuses on the behaviour of an agent [69]. For functional level scenarios to cater to textual language end users and for scenarios for simulation or real-world testing Zhang, Khastgir and Jennings propse a two level abstraction approach which is able to be transformed from one scenario to the other by adding or abstracting information [135]. The requirement, functions, logical and physical (RFLP) structure decomposes products into abstraction layers to ensure traceability from high-level requirements through the function and logical level down to the physical level [65]. In order to handle communication between various domains Zhang et al. propose visualization techniques for components, by using a domain-specific visualization, and connectivity, using a model connectivity abstraction method to encapsulate the details [134]. There is no one correct method for abstraction, rather the appropriate abstraction is dependent on the purpose.

5.3.6. Error Messages

When a program is run and issues are detected the compiler will generate an according error message. Usually, these messages are not encountered by domain experts but systems engineers tasked with implementing the system. However, with a notebook approach a model can be created by domain experts and interpreted as a program which can be run. This enables the generation of error messages to present the user and provide important information what

error occurred and possibly how to fix this, depending on the provided information. Barik et al. have shown that reading error messages is comparable to reading source code in terms of time spent and difficulty [14]. This is especially problematic for domain-experts who are not well versed in reading source code, making complex error messages not a suitable tool to provide feedback. To be helpful for domain experts and novice users such an error message needs to be enhanced as many error messages are inadequate [17]. Becker et al. defines enhancement of error messages as an altering of the text to improve the usefulness for a human reader [18]. A number of guidelines have emerged which are considered to provide more helpful error messages when followed. The most applicable of these guidelines in terms of inclusive systems engineering are:
Reduce cognitive load: Hundhausen, Olivares and Carter [66] provide three suggestions how to reduce the cognitive load needed to process an error message. *Placement of information* suggests that it is necessary to provide information needed to complete a task close to the task itself. In a notebook approach this suggests to add error messages directly to the cell and provide all important additional information alongside the message. *Reduction of redundancy* suggests that information should not be processed multiple times. Therefore, each error type should be presented only once, which can contradict the placement in case of the same error made in multiple cells. *Multiple channels*, such as auditory and visual, should be used to present complementary information. *Show solutions or hints*: Kummerfield and Kay [81] provide one example on how showing solutions or giving hints can improve the speed in which novice users are able to fix issues. A similar approach, fuelled by a comprehensive knowledge base consisting of similar models can be conceived. This approach can analyse both the issues leading to the error message and provide one or multiple suggested solutions similar to existing systems which the user can choose from in order to fix the issue.

5.4. Process Accessibility

Process accessibility refers to features making the systems engineering and MBSE process more accessible to domain experts and beginners. These do not include specific user guidance techniques or tool specific solutions but focus on the process and should be implemented alongside user guidance and tool accessibility features.

5.4.1. Tool Independence

Not being constrained by one specific tool provides more flexibility for users to work across different platforms and environments [111] [57]. This is especially important in projects that require high collaboration and domain-expert input. Many domains have highly specialized tools making it important to have a tool independent approach [5]. Being tool independent enhances scaleability and adaptability, not being constraint by the capabilities of a single tool but

being able to use the right tool for the right application. For model-based systems engineering this is especially interesting as tool independence offers the ability of unconstrained variable management, which enables the modelling of numerous configurations and dependencies in fewer models [111]. Tool independence has been rated highly as seen in figure 4.32.

Domain Specific Languages and Tools

Fowler defines a domain specific language (DSL) as: "*... a concise, processable language for describing a specific concern when building a system in a specific domain. The abstractions and notations used are tailored to the stakeholders who specify that particular concern* [54]. Such a DSL needs to be processable to be used with a domain specific tool. A DSL is specific for a certain purpose and not a general purpose language with abstractions relevant to that purpose. Wile [130] provides empirical evidence for the benefits and risks when designing DSLs whereas Karsai et al. provide a list of 26 guidelines which should be followed when designing a DSL [76].
Domain specific tools include:

- Modelica/FMI [98] [21] and MATLAB/Simulink [31] in the electrical engineering domains to provide simulation for electrical systems. FMI is used in any domain in need for co-simulating mechanical, electrical and control subsystems.
- Tools for earbox synthesis tools are used in mechanical engineering [84].
- In automotive systems SysML and AUTOSAR [56] are used to follow the MBSE approach
- The eclipse modeling framework (EMF) [116] is a non-commercial tool for model-driven systems engineering.

Executable documentation

Tegeler et al. [121] discuss executable documentation as a continuation of the trend to align documentation and code. The idea of an executable documentation is to turn the domain specific languages, tools and documentation into fully-fledged modelling and programming languages. This results into purpose-specific languages in which artefacts are modelled and can be created simply by providing the requirements of the artefact.

5.4.2. Simplified Syntax

Complex syntax can lead to several issues during systems engineering, depending on the context. During requirement analysis complex syntax can lead to ambiguous requirements, providing insufficient information or information prone to misinterpretation [91]. Ambiguity in requirements specification and

documentation, both in case of document-based or model-based systems engineering, can lead to misalignment between stakeholders. Complex syntax results in a higher cognitive load which reduces productivity. In addition, the requirement to learn complex syntax can be a hindrance to educate oneself in a new methodology, such as transitioning from document-based systems engineering towards model-based systems engineering.
A simplification of syntax, especially for frequently used statements reduces the cognitive load, making it easier to learn and adopt new systems engineering methodologies while simultaneously facilitating the creation of clear requirements and the use of modern tools to further improve requirements specification [95]. Tools such as natural language processing using large language models are promising to automatically correct requirements if faults are detected. However, the training of such models requires significant resources. Using natural language adjacent statements reduces the cognitive load and reduces complexity while keeping an underlying formalism to reduce ambiguity. Such syntax can be easier understood by domain-experts and lowers the entry bar while also providing a formal basis for large language models to be trained with as an intermediate step towards full natural language processing. The importance of simplified syntax is also highlighted by the study with members of INCOSE and the OMG as illustrated in figures 4.31 and 4.28.

Literate Programming

Knuth introduced the notion of literate programming by stating that better documentation can come from considering programs as *works of literature* and emphasising the importance to not instruct a computer that to do, but to explaining to a human what the computer is supposed to do [79]. In this work he introduced the WEB language and claims to need the same time to create a literate program as a structured program but with more documentation. Literate programming was considered to be a poor fit for software engineering, however Mathematica [132] is still a widely used tool, given that math and other sciences can benefit from the idea to integrate documentation and equations directly into a program . Pieterse, Kourie and Boake highlight literate programming essentials in their work [100]. These essentials are:

- *Literate quality*: The program is supposed to explain a human reader what it is supposed to do.

- *Psychological order*: The program is structured with a human reader in mind, with a logic structure enhancing human understanding which can be significantly different from a logic structure required by the compiler [23]. This also differentiates a structured program with comprehensive documentation from a literate program.

- *Integrated Documentation*: Documentation is an integral part of the program, not an afterthought or list of requirements the program needs to fulfill. Instead of the program being instructions for a computer with

some comments for a human reader the program should be an explanation to a human which includes comments for the compiler [131]

- *Table of contents, index and cross references*: An essential part of the information is the inclusion of a table of contents, an index and cross references between the related modules. This information enhances user understanding and conveys the global concept of the program. In modern literate programs it is assumed that all cross references are hyperlinks which can be created automatically [100].

- *Pretty printing*: Pretty printing refers to automatic application of indentation, font syles and text colors and other means to improve readability and ease of understanding such as syntax highlighting.

- *Verisimilitude*: The term versimilitude refers to the requirement that generation of executable code and a human readable literate version of a program is the same task and requires only one document and should not need two separate documents [126]. This results in all additional documents created for understanding need to be always included and updated during the same time when a program or model is updated.

Literate programming fits the idea of inclusive systems engineering well, emphasising the idea of creating systems and documentation at the same time with an accessible and human readable syntax. By following the principles of literate programming and providing an intuitive, easy to use syntax the entry bar for domain experts to participate in systems engineering is lowered. This is especially true with ever more readily accessible natural language processing tools.

5.4.3. Knowledge Base

A knowledge base is a structured repository of information which is designed to store, manage and retrieve knowledge in a machine-readable format. By storing systems and parts thereof the creation of a knowledge base is enabled. By referring to such a knowledge base it is possible to enhance communication between domain-experts and stakeholders. By reusing parts available in a knowledge base it is possible to create larger, more comprehensive systems in a shorter amount of time by focusing on the new components and functionality of a system instead of spending time to recreate systems that have been modelled many times before. The use of a knowledge base can also facilitate user guidance by providing more sophisticated recommendations or provide documentation for existing systems to enhance understanding of unfamiliar system components. Depending on the size and complexity of a system one can choose a suitable knowledge base.
Ontologies provide the formal structure and semantics to organise the information in a knowledge base. For large and complex systems, which are the main focus of MBSE the web ontology language (OWL) is suitable as it is designed for complex ontologies. OWL supports reasoning tasks such as checking

for class consistency and inferring relationships, which is important to infer relationships between parts of a model to support user guidance techniques such as recommender systems and AI powered inferences [60] [129]. More lightweight alternatives like the resource description framework (RDF) [86] and its extension RDF Schema (RDFS) [90] are more simple and can be useful for lightweight ontologies to provide reasoning on abstracted high-level models and function as a transition before making use of OWL and its capabilities.

5.4.4. Graphical & Textual Representation

Graphic and Textual representation both have benefits and disadvantages. A textual representation usually suits systems engineers making it easier and faster to develop large and complex systems, whereas a graphical notation enables easier communication and explanation between stakeholders. A clear graphical representation is intuitively understood, even by people not familiar with the intricacies of the model. However, creating a graphic representation of a system is usually slow, requiring the correct choice of graphical notation, well-arranged use of connections and well thought-out use of spacing, often being extremely hard to change or extend afterwards. Textual representations are suitable to create detailed relationships quickly, not being constrained by space or layout considerations. Textual representations can be easily extended and reused. However, textual representations are not well suitable for visualisation.
The preference of graphical and textual representations is supported by the data shown in figure 4.29.

Textual Representation

A textual representation uses formal language or textual syntax to describe a system in detail. A textual representation offers very precise definitions, making it very suitable for complex and very large systems. Text-based models can take advantage of version control systems and easily be shared, facilitating efficient collaboration and conflict resolution.
However, textual representations have a steeper learning curve and require domain-specific knowledge and technical expertise to interpret effectively. Graphical representations offer higher accessibility to non-specialists compared to textual formats.

Graphical Representation

A graphical representation uses visual elements such as diagrams, graphs or flowcharts to depict the relationships, processes and structures of a system. A graphical representation offers intuitive understanding of systems and are easy to interpret for users without technical expertise. They simplify the complex relationships by providing easy to understand visual representations. By providing a shared visual language graphical models are easy to share among stakeholders with different domain expertise.

Graphical representations of models are hard to scale for large or complex systems. The representation can become cluttered and hard to manage easily. In order to prevent cluttered models intricate details can be omitted, making the model less detailed in the process which can abstract important information unintentionally.

Blended approaches

An inclusive model-based systems engineering tool preferably enables both the use of graphic and textual representations. Offering the generation of a textual representation given a graphical representation and vice versa is beneficial, giving engineers the possibility to understand or explain a given model with the graphical representation, while extending a model using the textual representation.
As shown in figures 4.12, 4.1, 4.2 as well as figure 4.29 a graphical representation is preferred for visualisation purposes, whereas for creation a textual representation is preferred. This shows the importance of a blended approach, where a model can be represented depending on the purpose. KerML and SysML v2 provide means for both graphical and textual representation of a model. This enables tools to be created that can convert one representation into the other.

5.4.5. Incomplete Models

The complexity of large models makes it hard to correctly build a large model from scratch [85]. However, missing parts of a model will usually result in syntax errors in a textual representation. This makes incomplete models, given with in textual representations, unsuitable to use in some cases. An inclusive model-based systems engineering tool needs to support the use of incomplete models by abstracting undefined parts of a model. For modeling purposes undefined relationships or missing objects can be represented by substitutions. Such substitutions can be refined and changed at a later stage, but reduce the need for fully defined systems in the beginning.
The use of incomplete models has a number of benefits, the most prominent being able to both work from a top-down, as well as a bottom-up, perspective, being able to focus on the important parts of the intended scale. Additionally this enables an agile engineering process wherein different groups can focus on smaller parts of a large model, not needing the whole model to exist already and connecting the different parts in a later overarching stage. Incomplete models enable easier fixing, working on specific problems or omitting parts to be completed and extended at a later time. This results in models being easier to reiterate upon through the development process, even if major changes are required. So far in MBSE the use of incomplete models is only possible with graphical representations. The support of incomplete textual models can support interactivity, as well as, the validity of models to use as roadmaps, making it possible to test possibilities depending on possible future products, without reducing the current use. Additionally, incomplete models enable methods for

consistency checking or constraint checking for parts of a model, reducing the risk of inconsistent models by testing smaller parts. Additionally, this supports easier cooperation, being able to model parts and extend them easily in the textual representation to merge them at a later point in time.
In MBSE incomplete models play a significant role by enabling iterative refinement, early validation of parts and efficient collaboration. Incomplete models can use the so-called 3-valued scoped partial modeling to approximate system behaviour and constraints proposed by Marussy et al. [87]. In this approach predicate and counter abstraction is combined to evaluate well-formedness rules and multiplicity requirements on the partial models. This allows for identification of design flaws early in the process.
The use and support of incomplete models enables early-stage models to focus on high-level architecture and requirements. These models are intentionally incomplete to be refined later during the development and provide easy to communicate high-level architecture for stakeholders.

5.4.6. Stakeholder Specific Views

Providing the option to render different views of a model is important for using a model as a communication tool depending on the audience [112]. A coarse model offers a good grasp for stakeholders, being a useful tool in the early design process or updating different teams, whereas comprehensive and detailed models are more useful for in-team discussions between domain experts, who have no use for a large-scale view. Such views can also abstract unnecessary information when extending a model in a certain direction, giving access to basic functionality without overwhelming the engineer. The identification of different stakeholders and what kind of information is most suitable for them to create views accordingly is highly dependent on the individuals. Thus a flexible method to configure an abstraction level directly would be preferable over a set definition of views and the abstractions used.
User specific views are especially important for systems engineers which value stakeholder collaboration and the ability to present system specifications as shown in figure 4.33.
The notion of views explored in the UML toolkit by Erikson [48] specifies a use-case view, a logical view, an implementation view, process view and development view. These view concepts can be extended towards views depending on the user, such as organisational views providing information important to specific stakeholders. Which information is important for stakeholders highly depend on their role. These viewpoints have been reviewed by works from Darke and Shanks [43] and Brugha and Varvasovszky [25].

5.5. Tool Accesibility

The tool accessibility features focus on providing mechanism to make systems engineering tools more approachable. User guidance techniques can and should be used to make tools more accessible, however these techniques are discussed

in their own section. The features discussed in this section are specific to tools and can exclusive to one another depending on the tool or process specifications. Such examples include different means of version control or providing extendability which can be in conflict with the use of a hyper specific DSL.

5.5.1. Integrated Documentation

The benefits of comprehensive documentation have been shown multiple times. However, documentation is often done insufficient, especially neglecting changes of a system and it's requirements over time [53] [101]. Documentation is traditionally done in it's own separate documents stemming from document-based systems engineering, which requires additional effort to link these documents and to keep them up to date. This is especially true with systems becoming more complex and interconnected.
One way to solve this issue is to integrate the documentation directly into the system specification. This is possible when using tools which allow for specification and documentation to be done in the same document, for example by using markdown language which can identify specification sections and documentation sections in one file.
MBSE provides integrated documentation by providing a central model from which additional documentation is created. By utilising AI techniques such as machine learning and large language models documentation can be created automatically. This approach follows the approach discussed in 5.5.5 with the alteration of creating documentation and natural language instead of code from the processed specifications.
Such integrated documentation facilitates the communication of system parts to stakeholders both easily readable as documentation and formal syntax. In addition, this integrated documentation can be used to fuel recommender systems or AI tools to provide automatic creation of system specification from informal documentation in one document. A linked or integrated documentation was rated as highly important by the survey discussed in the conducted survey as illustrated in figure 4.30.

5.5.2. Notebook programming

One option to integrate documentation and code, such as textual model representations and system features, is the use of notebook programming. A notebook is a sequence of cells. Each cell has a type which is either markdown for documentation or code. Cells which include markdown text can be rendered by the notebook while code sections are run and produce an output. Typically, notebooks are a linear sequence of cells, however other options such as a spreadsheet-style do exist [114]. Notebooks become ever more popular, shown by the exponential growth of notebook files stored in the GitHub centralized repository [97]. Kery et al. [77] argues for future notebook programming to include diverse GUI capabilities, such as drag and drop functionalities between cells and the inclusion of editable spreadsheets and other GUI fea-

tures.
In its current version SysMD includes a notebook which provides the functionality to interpret markdown, SysMD and SysML v2 code. This tool is currently exploring the implementation of such features.

5.5.3. Version Management

Providing a version management system, also known as version control system, is important to facilitate an iterative approach and to enhance collaboration between stakeholders. Version control offers a number of benefits such as:

- Working parallel, from a shared starting point, on distinct parts of a system to be integrated to be connected at a later point in time.
- Enhanced traceability of changes and their impact on a system, for example by simulation of different versions.
- Enable working on distinct configurations of a system without impacting other stakeholders.
- Enable to use of incomplete models to be communicated between stakeholders.

Version management has been consistently rated as very important as shown in figure 4.30.
Version control systems are usually either a centralized version control system, such as concurrent version system (CVS) [29] or Subversion (SVN) [34] or a distributed version control system, such as Git [30] or Mercurial [93].

centralized version control system

A central version control system stores all versions of files in a single, central repository. A developer can check out the files from the central server before making changes and committing them back. This is simple to set up and manage and provides a clear history of the changes made to a file. However, such a system offers only limited offline capabilities and is vulnerable to a server failure. For a comparison between CVS and SVN I refer to the comparison provided by Abukari [2].

decentralized version control system

In a decentralized version control system each developer has a complete copy of the repository, including all files and its history. Changes are made locally and committed locally to be pushed to a shared repository at a later point in time. This offers high redundancy, supports offline work and facilitates fast operations for local tasks or many local iterations of a file before committing it to the central repository. However, the workflows of a decentralized version control system are more complex, especially if multiple instances of a file are created from users in parallel. Such files need to be merged and issues in

merging can be highly complex to solve, usually outside. the capabilities of the average user.
For distributed systems engineering where specification changes come from multiple sources and have to be communicated frequently a decentralized version control system is preferred to provide offline capabilities. In addition this offers the option for companies to use existing version control systems internally and coordinate with a decentralized system.

5.5.4. Extendability

Extendability facilitates domain-expert collaboration by providing a basic solution which can be expanded differently depending on different domains. This provides a common basis among stakeholders to improve communication and ensure compatibility between different domain specific solutions. One such basic solution is KerML as described in 2.3.4 which provides a basis for languages such as SysML v2 and is designed to be extended upon to provide different domain specific solutions which are all compatible thanks to the common basis of KerML. This provides more flexibility and supports tool independence described in 5.4.1.

5.5.5. Code Generation

Code generation refers to the general process of automatically creating program code from specifications, inputs or models. This improves efficiency by reducing manual coding efforts and enables developers to focus on higher-level problem-solving. With current systems engineering being mostly based on the graphical representations done in SysML code generation has been used for the implementation phases of systems. With the introduction of SysML v2 and its textual representation it is possible to generate models automatically from specifications, design and requirements.
The ability to use SysML diagrams to generate corresponding software has been shown by Jamro by creating code for Program Organisation Units from the given diagrams [70]. The feature of automatic model generation from informal description was not valued important according to figure 4.33. However, with the fast improving abilities of AI the integration is worthwhile, as shown by studies such as the one from Crabb and Jones [36].

GUI processing

Various machine learning techniques have been used to generate code from GUI images [115] without a clear indication which techniques are the most well suited for the task. Aşıroğlu et al. [11] present a technique allowing for the creation of HTML code from hand-drawn mock-up images with a high accuracy. Techniques like these can be used to convert SysML diagrams from various sources into textual representations of the given graphical model. This enables the transition from SysML to SysML v2 even for systems in which

SysML diagrams are only existing in formats not directly supported by future SysML v2 tools.

Generative AI using Large Language Models

The advances made in large language models (LLM) and generative AI provide easily accessible means for code generation, using existent systems as a basis to provide system specifications from informal description. By using code generation techniques the documentation can be used to create system specification, such as models with minimal effort. These models can be expanded upon and verified quickly, without the need to repeatedly creating similar system specification over and over. The existence of a comprehensive knowledge base can be used for training purposes. However, without such an knowledge base LLMs can already be used beneficially. Crabb and Jones [36] have shown that LLMs can be used to create high-level models and, more importantly in the context of inclusive systems engineering, answer detailed questions on domain-specific topics. Bader, Vereno and Neureiter [12] highlight the ability of AI to improve user-centric MBSE by providing means to overcome beginner hurdles and assist in standardizing the use of modeling languages. Pontes Miranda et al. [102] focus on the use of AI capabilities to overcome the use of different modelling languages for various system's aspects by connecting them through views with abstracted information and supporting engineers to create these model views with the help of AI.

5.5.6. Interactivity

Interactive use has a multitude of benefits [109]. Feedback is an important tool to make sure work is done correctly, efficiently and early feedback reduces frustration, as less work has to be redone in case of problems.
Interaction with the user can have many different forms, for example providing real-time feedback to enable iterative refinement, such as providing feedback of a model by analysing consistency. This improves the modelling process, being able to identify inconsistencies or necessary refinements. Allowing users to modify parameters or create custom configurations of a tool is a common type of interactivity as well. The use of user guidance techniques such as recommender systems, comprehensive error messages and others as discussed in 5.3 also provide various forms of interactivity. The use of interactive tools has been shown to provide a multitude of benefits, ranging from feedback to suggested next steps in the process [109].

5.5.7. Standard Compatibility

Being compatible with a standard is important to reduce the number of incompatible tools. The goal of ISE is to be accessible for domain experts, being incompatible with existing standards or focusing on single domains would contradict this purpose.
Being compatible with a widespread standard improves systems engineering

accessibility and offers access to the vast resources, tools and features provided for the standard. Furthermore, standard compatible systems can easily be communicated and adapted to domain specific purposes. Being compatible with an existing standard provides a number of benefits:

- Access to a vast amount of resources, both for developers as well as users.
- Increased compatibility with other tools.
- Extensions to the standard can easier be implemented to provide additional features.
- New features provided can be distinguished, making the contribution more clear.

SysMLv2 is being standardized by the OMG group, based on their KerML design. KerML is designed with the professed goal of providing a basis to be extended upon with new features and tools.

Chapter 6

Conclusions

The issue of ever more complex systems which require ever more domain-expert input is a pressing matter in systems engineering. One option to facilitate domain-expert input and comprehensive requirement analysis is to simplify the systems engineering process, to facilitate communication between systems engineers and domain experts, by mitigating misunderstandings based on unfamiliarity with the process. To identify important issues and requested features in systems engineering from future domain experts a survey was conducted with students from the RPTU Kaiserslautern with little to no prior systems engineering experience. This study identified the need to simplify systems engineering syntax, especially considering model-based systems engineering. This was achieved by presenting an early version of SysMD which provided simplified syntax, close to natural language, as an alternative to the SysML v2 syntax for very common definitions and relationships. This survey identifies the preference for the simplified, natural language adjacent, syntax of SysMD and a preference for graphical representation to understand a given model and a textual representation to create new models in.
A second survey collected feedback from people interested in systems engineering or actively working in this domain by sharing the study with OMG and INCOSE members. This survey did not consider language specifics from SysML, SysML v2 or SysMD but rather focused on general issues and requested features. This survey identified four groups of participants, those with professional level systems engineering expertise, an intermediate level, beginner level and no experience at all. The data from this survey indicates the importance of issues on average, as well as for the distinct subgroups. Issues such as simplification of syntax and providing supporting features such as version management and a documentation directly linked with the system model were considered important for all participants, whereas issues on model verification and validation and the ability to compute expressions was considered to be more important by professional systems engineers than. Overall, systems engineering is considered to be hard to get into, providing indicative evidence for a need of simplification.
The method of surveys has been chosen due to its asynchronous nature, pro-

viding a means to gather participants over a period of time without the need to spend resources on conducting interviews or overseeing systems engineers or domain experts in their usual work environment. This makes the studies prone to sampling biases which is discussed as limitations to the studies.
From the insights gathered in the surveys and by reviewing studies from the literature the concept of inclusive systems engineering has been developed. Inclusive systems engineering is defined as an extension systems engineering with the focus on increased accessibility to facilitate domain expert integration. User guidance has been proven to increase motivation and reduce adverse affects on adopting new technology or changing behaviour. In order to identify features to achieve a simplification and lowering the entry bar for domain experts and new systems engineers alike this thesis explores user guidance techniques and gives an introduction into human decision making.
Inclusive systems engineering is based on including features enhancing tool accessibility, accessibility of the process as well as user guidance techniques. For each of these categories features and their provided advantages have been discussed. Additionally, it highlights the need to not only improve the capabilities of new tools and languages but also keep in mind the users and their needs.

6.1. Future Works

The studies only provide tentative evidence for the importance of issues and requested features being prone to sampling bias and the under representation of experts from domains such as medical, defense or aerospace.
A more comprehensive survey combined with interviews conducted with domain experts from multiple domains can provide more robust insight on the current state of systems engineering and domain expert integration.
The implementation of inclusive systems engineering features in an existing tool can provide an example implementation to be reviewed and tested in order to identify how the proposed features can be integrated and how effective these features are in lowering the entry bar for beginners. To test this an example implementation can be integrated into a systems engineering lecture for students to evaluate the tool and identify benefits and challenges.
With the expansion of systems engineering into ever more domains it is important to evaluate the inclusive systems engineering approach for different domains. Especially the ability to provide textual, as well as graphical representations of a model needs further research. A tools able to switch from one representation to the other with minimal effort to support visualisation and creation would be ideal to test if the preference reported in the surveys is only reported.
Further research on the mismatch between perceived stakeholder collaboration and missing domain-expert input importance is needed. One reason for this mismatch could be the high integration of stakeholders who are not domain experts and a low integration of domain experts. Further investigation on this mismatch by guided interviews can identify the cause of this effect.

The development of SysMD realises some of the proposed features for inclusive MBSE, however with the ongoing development it has not yet been able to implement all proposed features and to use SysMD as a means to evaluate inclusive systems engineering as a whole. Both SysMD and SysML v2 development need to be examined for their support of inclusive systems engineering and the effect this has on MBSE adoption.

Bibliography

[1] Kenan Abacı and Ihsan Tolga Medeni. "Efficiency of electronic document management systems: a case study". In: *Sci. Educ. Innov. Context Mod. Probl* 5 (2022), pp. 75–86.

[2] Arnold Mashud Abukari. "Configuration Management: A comparative Analysis of CVS and SVN". In: *International Journal of ICT and Management* 1.1 (2013), pp. 157–162.

[3] Alexander T Adams, Jean Costa, Malte F Jung, and Tanzeem Choudhury. "Mindless computing: designing technologies to subtly influence behavior". In: *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*. 2015, pp. 719–730.

[4] Morayo Adedjouma, Thibaud Thomas, Chokri Mraidha, Sebastien Gerard, and Guillaume Zeller. "From document-based to model-based system and software engineering". In: *Proceedings of the OSS4MDE* (2016).

[5] Alexander Ahlbrecht, Jasper Sprockhoff, and Umut Durak. "A system-theoretic assurance framework for safety-driven systems engineering". In: *Software and Systems Modeling* (2024), pp. 1–18.

[6] Bushra Alkadhi, Robert Hendley, and Rami Bahsoon. "Nudge Me and I Change: The Effect of Digital Nudging on Software Engineers' Motivation and Behaviour Towards Data Quality". In: *37th International BCS Human-Computer Interaction Conference*. BCS Learning & Development. 2024, pp. 193–204.

[7] Alaa AlMarshedi, Vanissa Wanick, Gary B Wills, and Ashok Ranchhod. "Gamification and behaviour". In: *Gamification: Using game elements in serious contexts*. Springer, 2016, pp. 19–29.

[8] Hamza Alshenqeeti. "Interviewing as a data collection method: A critical review". In: *English linguistics research* 3.1 (2014), pp. 39–45.

[9] Federico de Andreis. "A theoretical approach to the effective decision-making process". In: *Open Journal of Applied Sciences* 10.6 (2020), pp. 287–304.

[10] Jorge Aranda, Daniela Damian, and Arber Borici. "Transition to model-driven engineering: What is revolutionary, what remains the same?" In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2012, pp. 692–708.

[11] Batuhan Aşıroğlu, Büşta Rümeysa Mete, Eyyüp Yıldız, Yağız Nalçakan, Alper Sezen, Mustafa Dağtekin, and Tolga Ensari. "Automatic HTML code generation from mock-up images using machine learning techniques". In: *2019 Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science (EBBT)*. Ieee. 2019, pp. 1–4.

[12] Elias Bader, Dominik Vereno, and Christian Neureiter. "Facilitating User-Centric Model-Based Systems Engineering Using Generative AI." In: *MODELSWARD*. 2024, pp. 371–377.

[13] Manas Bajaj, Sanford Friedenthal, and Ed Seidewitz. "Systems modeling language (SysML v2) support for digital engineering". In: *Insight* 25.1 (2022), pp. 19–24.

[14] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. "Do developers read compiler error messages?" In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE. 2017, pp. 575–585.

[15] Mona Batora, Roman Ognevoj, Moritz Schöck, Katharina Ritzer, Nikola Bursac, et al. "Engineering Better Decisions: Exploring Influencing Factors of Behavioral Economics on SGE-System Generation Engineering". In: *DS 130: Proceedings of NordDesign 2024, Reykjavik, Iceland, 12th-14th August 2024* (2024), pp. 888–897.

[16] Kathy Baxter, Catherine Courage, and Kelly Caine. *Understanding your users: a practical guide to user research methods*. Morgan Kaufmann, 2015.

[17] Brett A Becker. "An effective approach to enhancing compiler error messages". In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 2016, pp. 126–131.

[18] Brett A Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, et al. "Compiler error messages considered unhelpful: The landscape of text-based programming error message research". In: *Proceedings of the working group reports on innovation and technology in computer science education* (2019), pp. 177–210.

[19] Bo-Christer Björk. "Electronic document management in construction–research issue and results". In: (2003).

[20] Benjamin S Blanchard. *System engineering management*. John Wiley & Sons, 2004.

[21] Torsten Blochwitz, Martin Otter, Martin Arnold, Constanze Bausch, Christoph Clauß, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, Manuel Monteiro, Thomas Neidhold, et al. "The functional mockup interface for tool independent exchange of simulation models". In: *Proceedings of the 8th international Modelica conference*. Linköping University Press. 2011, pp. 105–114.

[22] Chris Brown. "Digital nudges for encouraging developer actions". In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE. 2019, pp. 202–205.

[23] Marcus Edward Brown. *An interactive environment for literate programming*. Texas A&M University, 1988.

[24] Manfred Broy. "Challenges in automotive software engineering". In: *Proceedings of the 28th international conference on Software engineering*. 2006, pp. 33–42.

[25] Ruairí Brugha and Zsuzsa Varvasovszky. "Stakeholder analysis: a review". In: *Health policy and planning* 15.3 (2000), pp. 239–246.

[26] Antonio Bucchiarone, Jordi Cabot, Richard F Paige, and Alfonso Pierantonio. "Grand challenges in model-driven engineering: an analysis of the state of the research". In: *Software and Systems Modeling* 19 (2020), pp. 5–13.

[27] Erion Çano and Maurizio Morisio. "Hybrid recommender systems: A systematic literature review". In: *Intelligent data analysis* 21.6 (2017), pp. 1487–1524.

[28] Ana Caraban, Evangelos Karapanos, Daniel Gonçalves, and Pedro Campos. "23 ways to nudge: A review of technology-mediated nudging in human-computer interaction". In: *Proceedings of the 2019 CHI conference on human factors in computing systems*. 2019, pp. 1–15.

[29] Per Cederqvist, Roland Pesch, et al. *Version management with CVS*. 1992.

[30] Scott Chacon and Ben Straub. *Pro git*. Springer Nature, 2014.

[31] Devendra K Chaturvedi. *Modeling and simulation of systems using MATLAB and Simulink*. CRC press, 2017.

[32] Robert B Cialdini and Melanie R Trost. "Social influence: Social norms, conformity and compliance." In: (1998).

[33] Christine Clavien. "Ethics of nudges: A general framework with a focus on shared preference justifications". In: *Journal of Moral Education* 47.3 (2018), pp. 366–382.

[34] Ben Collins-Sussman. "The subversion project: buiding a better cvs". In: *Linux Journal* 2002.94 (2002), p. 3.

[35] Maxime Collomb and Mountaz Hascoët. "Extending drag-and-drop to new interactive environments: A multi-display, multi-instrument and multi-user approach". In: *Interacting with Computers* 20.6 (2008), pp. 562–573.

[36] Erin Smith Crabb and Matthew T Jones. "Accelerating model-based systems engineering by harnessing generative ai". In: *2024 19th Annual System of Systems Engineering Conference (SoSE)*. IEEE. 2024, pp. 110–115.

[37] Sandor Dalecke. “A Review of User Guidance Techniques to enable “inclusive” Systems Engineering for Domain Experts”. In: *Application of Emerging Technologies* 115.115 (2023).

[38] Sandor Dalecke. “Human Factors in Software and Systems Engineering, Vol. 140, 2024, 59–66 AHFE”. In: *Human Factors in Software and Systems Engineering* (2024), p. 59.

[39] Sandor Dalecke and Randi Karlsen. “Designing dynamic and personalized nudges”. In: *Proceedings of the 10th international conference on web intelligence, mining and semantics*. 2020, pp. 139–148.

[40] Šandor Dalecke and Christoph Grimm. “SysMD: An Inclusive Modelling Tool”. In: *2024 19th Annual System of Systems Engineering Conference (SoSE)*. IEEE. 2024, pp. 178–183.

[41] Šandor Dalecke, Khushnood Adil Rafique, Axel Ratzke, Christoph Grimm, and Johannes Koch. “SysMD: Towards “Inclusive” Systems Engineering”. In: *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE. 2022, pp. 1–6.

[42] Roy Darioshi and Eyal Lahav. “The impact of technology on the human decision-making process”. In: *Human Behavior and Emerging Technologies* 3.3 (2021), pp. 391–400.

[43] Peta Darke and Graeme Shanks. “Stakeholder viewpoints in requirements definition: A framework for understanding viewpoint development approaches”. In: *Requirements engineering* 1 (1996), pp. 88–105.

[44] Benedetto De Martino, Dharshan Kumaran, Ben Seymour, and Raymond J Dolan. “Frames, biases, and rational decision-making in the human brain”. In: *Science* 313.5787 (2006), pp. 684–687.

[45] Paul Dolan, Michael Hallsworth, David Halpern, Dominic King, Robert Metcalfe, and Ivo Vlaev. “Influencing behaviour: The mindspace way”. In: *Journal of Economic Psychology* 33.1 (2012), pp. 264–277.

[46] Dov Dori. *Object-Process Methodology: A Holistic Systems Paradigm; with CD-ROM*. Springer Science & Business Media, 2002.

[47] Ward Edwards. “The theory of decision making.” In: *Psychological bulletin* 51.4 (1954), p. 380.

[48] Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado. *UML 2 toolkit*. John Wiley & Sons, 2003.

[49] Jeff A Estefan et al. “Survey of model-based systems engineering (MBSE) methodologies”. In: *Incose MBSE Focus Group* 25.8 (2007), pp. 1–12.

[50] Jonathan St BT Evans and Keith E Stanovich. “Dual-process theories of higher cognition: Advancing the debate”. In: *Perspectives on psychological science* 8.3 (2013), pp. 223–241.

[51] Brian J Fogg. “A behavior model for persuasive design”. In: *Proceedings of the 4th international Conference on Persuasive Technology*. 2009, pp. 1–7.

[52] Brian J Fogg. "Persuasive technology: using computers to change what we think and do". In: *Ubiquity* 2002.December (2002), p. 2.

[53] Andrew Forward and Timothy C Lethbridge. "The relevance of software documentation, tools and technologies: a survey". In: *Proceedings of the 2002 ACM symposium on Document engineering*. 2002, pp. 26–33.

[54] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

[55] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[56] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkelin, Kenji Nishikawa, and Klaus Lange. "AUTOSAR–A Worldwide Standard is on the Road". In: *14th international VDI congress electronic systems for vehicles, Baden-Baden*. Vol. 62. 5. Citeseer. 2009.

[57] J.Z. Gao, C. Chen, Y. Toyoshima, and D.K. Leung. "Developing an integrated testing environment using the World Wide Web technology". In: *Proceedings Twenty-First Annual International Computer Software and Applications Conference (COMPSAC'97)*. 1997, pp. 594–601. DOI: 10.1109/CMPSAC.1997.625078.

[58] Lynda T Goodfellow. "An overview of survey research". In: *Respiratory Care* 68.9 (2023), pp. 1309–1313.

[59] Object Management Group, ed. *Kernel Modeling Language™ (KerML™) Version 1.0 Beta*. English. Feb. 2024. URL: https://www.omg.org/spec/KerML.

[60] Kristina L Guo. "DECIDE: a decision-making model for more effective decision making by health care managers". In: *The health care manager* 39.3 (2020), pp. 133–141.

[61] Pelle Guldborg Hansen and Andreas Maaløe Jespersen. "Nudge and the manipulation of choice: A framework for the responsible use of the nudge approach to behaviour change in public policy". In: *European journal of risk regulation* 4.1 (2013), pp. 3–28.

[62] Steven A Harvey. "Observe before you leap: Why observation provides critical insights for formative research and intervention design that you'll never get from focus groups, interviews, or KAP surveys". In: *Global Health: Science and Practice* 6.2 (2018), pp. 299–316.

[63] Matthew Hause et al. "The SysML modelling language". In: *Fifteenth European systems engineering conference*. Vol. 9. 2006, pp. 1–12.

[64] Kaitlin Henderson and Alejandro Salado. "The effects of organizational structure on MBSE adoption in industry: Insights from practitioners". In: *Engineering Management Journal* 36.1 (2024), pp. 117–143.

[65] Laszlo Horvath and Imre J Rudas. "Systems engineering in product definition". In: *2015 IEEE 13th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE. 2015, pp. 181–186.

[66] C. D. Hundhausen, D. M. Olivares, and A. S. Carter. "IDE-Based Learning Analytics for Computing Education: A Process Model, Critical Review, and Research Agenda". In: *ACM Trans. Comput. Educ.* 17.3 (Aug. 2017). DOI: 10.1145/3105759. URL: https://doi.org/10.1145/3105759.

[67] INCOSE. *INCOSE systems engineering handbook*. John Wiley & Sons, 2023.

[68] INCOSE. *International Council on Systems Engineering (INCOSE) Systems Engineering Vision 2035*. 2021. URL: https://www.incose.org/docs/default-source/se-vision/incose-se-vision-2035.pdf?sfvrsn=e32063c7_10.

[69] Toru Ishida. "Q: A scenario description language for interactive agents". In: *Computer* 35.11 (2002), pp. 42–47.

[70] Marcin Jamro. "Automatic generation of implementation in SysML-based model-driven development for IEC 61131-3 control software". In: *2014 19th international conference on methods and models in automation and robotics (mmar)*. IEEE. 2014, pp. 468–473.

[71] Mathias Jesse and Dietmar Jannach. "Digital nudging with recommender systems: Survey and future directions". In: *Computers in Human Behavior Reports* 3 (2021), p. 100052.

[72] Daniel Kahneman and Patrick Egan. *Thinking, fast and slow*. Vol. 1. Farrar, Straus and Giroux New York, 2011.

[73] Daniel Kahneman, Jack L Knetsch, and Richard H Thaler. "Anomalies: The endowment effect, loss aversion, and status quo bias". In: *Journal of Economic perspectives* 5.1 (1991), pp. 193–206.

[74] Daniel Kahneman and Amos Tversky. "Prospect theory: An analysis of decision under risk". In: *Handbook of the fundamentals of financial decision making: Part I*. World Scientific, 2013, pp. 99–127.

[75] Randi Karlsen and Anders Andersen. "Recommendations with a nudge". In: *Technologies* 7.2 (2019), p. 45.

[76] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. "Design guidelines for domain specific languages". In: *arXiv preprint arXiv:1409.2378* (2014).

[77] Mary Beth Kery, Donghao Ren, Kanit Wongsuphasawat, Fred Hohman, and Kayur Patel. "The future of notebook programming is fluid". In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–8.

[78] Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. "Natural language processing: state of the art, current trends and challenges". In: *Multimedia tools and applications* 82.3 (2023), pp. 3713–3744.

[79] Donald Ervin Knuth. "Literate programming". In: *The computer journal* 27.2 (1984), pp. 97–111.

[80] Adrian Kuhn, Gail C Murphy, and C Albert Thompson. "An exploratory study of forces and frictions affecting large-scale model-driven development". In: *Model Driven Engineering Languages and Systems: 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30–October 5, 2012. Proceedings 15*. Springer. 2012, pp. 352–367.

[81] Sarah K. Kummerfeld and Judy Kay. "The neglected battle fields of syntax errors". In: *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20*. ACE '03. Adelaide, Australia: Australian Computer Society, Inc., 2003, pp. 105–111. ISBN: 0909925984.

[82] Tim-Benjamin Lembcke, Nils Engelbrecht, Alfred Benedikt Brendel, and Lutz M Kolbe. "To Nudge or not to Nudge: Ethical Considerations of Digital nudging based on its Behavioral Economics roots." In: *ECIS*. 2019.

[83] Elizabeth D Liddy. "Natural language processing". In: (2001).

[84] Yi-shih Lin, Kristina Shea, Aylmer Johnson, John Coultate, and Jamie Pears. "A method and software tool for automated gearbox synthesis". In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 49026. 2009, pp. 111–121.

[85] Azad M Madni and Michael Sievers. "Model-based systems engineering: Motivation, current status, and research opportunities". In: *Systems Engineering* 21.3 (2018), pp. 172–190.

[86] Frank Manola, Eric Miller, Brian McBride, et al. "RDF primer". In: *W3C recommendation* 10.1-107 (2004), p. 6.

[87] Kristóf Marussy, Oszkár Semeráth, and Dániel Varró. "Automated generation of consistent graph models with multiplicity reasoning". In: *IEEE Transactions on Software Engineering* 48.5 (2020), pp. 1610–1629.

[88] Amir Matallaoui, Nicolai Hanner, and Rüdiger Zarnekow. "Introduction to gamification: Foundation and underlying theories". In: *Gamification: Using game elements in serious contexts* (2017), pp. 3–18.

[89] Justin St-Maurice, Catherine Burns, Justin Wolting, et al. "Applying persuasive design techniques to influence data-entry behaviors in primary care: repeated measures evaluation using statistical process control". In: *JMIR Human Factors* 5.4 (2018), e9029.

[90] Brian McBride. "The resource description framework (RDF) and its vocabulary description language RDFS". In: *Handbook on ontologies*. Springer, 2004, pp. 51–65.

[91] Ana Melo, Roberta Fagundes, Valentina Lenarduzzi, and Wylliams Barbosa Santos. "Identification and measurement of Requirements Technical Debt in software development: A systematic literature review". In: *Journal of Systems and Software* 194 (2022), p. 111483.

[92] MF Murillo-Munoz, Mabel Vazquez-Briseno, Christian Xavier Navarro Cota, and Juan I Nieto-Hipólito. "A framework for design and development of persuasive mobile systems". In: *2018 International Conference on Electronics, Communications and Computers (CONIELECOMP)*. IEEE. 2018, pp. 59–66.

[93] Bryan O'Sullivan. *Mercurial: The Definitive Guide: The Definitive Guide*. " O'Reilly Media, Inc.", 2009.

[94] Harri Oinas-Kukkonen and Marja Harjumaa. "Persuasive systems design: key issues, process model and system features 1". In: *Routledge handbook of policy design*. Routledge, 2018, pp. 87–105.

[95] Arthur HM de Oliveira, Pedro Almeida Reis, Fernando Sarracini Júnior, Mairon Sena Cavalcante, Jonathan VC de Lima, Luis FC Soares, and Lucas Henrique Marchiori. "Impact Analysis of using Natural Language Processing and Large Language Model on Automated Correction of Systems Engineering Requirements". In: *INCOSE International Symposium*. Vol. 34. 1. Wiley Online Library. 2024, pp. 992–1007.

[96] OMG. *Systems Modeling Language (SysML) v2 RFP*. Dec. 2002. URL: https://www.omg.org/cgi-bin/doc.cgi?ad%2F2017-12-2.

[97] Peter Parente. *Estimate of public Jupyter notebooks on GitHub*. 2014.

[98] Matthias Pazold, Sebastian Burhenne, Jan Radon, Sebastian Herkel, and Florian Antretter. "Integration of Modelica models into an existing simulation software using FMI for Co-Simulation". In: *9th International Modelica Conference*. 2012, pp. 949–954.

[99] Hang Thu Pho and Torben Tambo. "Integrated management systems and workflow-based electronic document management: An empirical study". In: *Journal of Industrial Engineering and Management (JIEM)* 7.1 (2014), pp. 194–217.

[100] Vreda Pieterse, Derrick G Kourie, and Andrew Boake. "A case for contemporary literate programming". In: *ACM International Conference Proceeding Series*. Vol. 75. 2004, pp. 2–9.

[101] Reinhold Plösch, Andreas Dautovic, and Matthias Saft. "The value of software documentation quality". In: *2014 14th International Conference on Quality Software*. IEEE. 2014, pp. 333–342.

[102] James William Pontes Miranda, Hugo Bruneliere, Massimo Tisi, and Gerson Sunyé. "Towards an In-Context LLM-Based Approach for Automating the Definition of Model Views". In: *Proceedings of the 17th ACM SIGPLAN International Conference on Software Language Engineering*. 2024, pp. 29–42.

[103] Marshall Pratt and Matthew Dabkowski. "Analysing the integration of MBSE approaches within the aerospace industry according to UTAUT". In: *Industrial and Systems Engineering Review* 10.2 (2022), pp. 127–134.

[104] Matthew Rabin and Richard H Thaler. "Anomalies: risk aversion". In: *Journal of Economic perspectives* 15.1 (2001), pp. 219–232.

[105] Teppo Räisänen, Tuomas Lehto, and Harri Oinas-Kukkonen. "Practical findings from applying the PSD model for evaluating software design specifications". In: *Persuasive Technology: 5th International Conference, PERSUASIVE 2010, Copenhagen, Denmark, June 7-10, 2010. Proceedings 5*. Springer. 2010, pp. 185–192.

[106] Hajo A Reijers, Thomas Freytag, Jan Mendling, and Andreas Eckleder. "Syntax highlighting in business process models". In: *Decision Support Systems* 51.3 (2011), pp. 339–349.

[107] Francesco Ricci, Lior Rokach, and Bracha Shapira. "Introduction to recommender systems handbook". In: *Recommender systems handbook*. Springer, 2010, pp. 1–35.

[108] Chris Rupp, Stefan Queins, et al. *UML 2 glasklar: Praxiswissen für die UML-Modellierung*. Carl Hanser Verlag GmbH Co KG, 2012.

[109] Samar Al-Saqqa, Samer Sawalha, and Hiba AbdelNabi. "Agile software development: Methodologies and trends." In: *International Journal of Interactive Mobile Technologies* 14.11 (2020).

[110] Hanna Schneider, Kilian Moser, Andreas Butz, and Florian Alt. "Understanding the mechanics of persuasive system design: a mixed-method theory-driven analysis of freeletics". In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 2016, pp. 309–320.

[111] Felix Schwägerl, Thomas Buchmann, Sabrina Uhrig, and Bernhard Westfechtel. "Towards the integration of model-driven engineering, software product line engineering, and software configuration management". In: *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. IEEE. 2015, pp. 1–14.

[112] Aditya A Shah, Aleksandr A Kerzhner, Dirk Schaefer, and Christiaan JJ Paredis. "Multi-view modeling to support embedded systems engineering in SysML". In: *Graph transformations and model-driven engineering: essays dedicated to Manfred Nagl on the occasion of his 65th birthday* (2010), pp. 580–601.

[113] Iqtiar Siddique. "MBSE Implementation in Small Satellite Systems: Rationale for Adoption over Traditional Document-Based Systems Engineering". In: *Available at SSRN 5149862* (2024).

[114] Jeremy Singer. "Notes on notebooks: Is Jupyter the bringer of jollity?" In: *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 2020, pp. 180–186.

[115] Daniel de Souza Baulé, Christiane Gresse von Wangenheim, Aldo von Wangenheim, and Jean CR Hauck. "Recent Progress in Automated Code Generation from GUI Images Using Machine Learning Techniques." In: *J. Univers. Comput. Sci.* 26.9 (2020), pp. 1095–1127.

[116] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[117] Cass R Sunstein. "Do people like nudges". In: *Admin. L. Rev.* 68 (2016), p. 177.

[118] Cass R Sunstein. "The ethics of nudging". In: *Yale J. on Reg.* 32 (2015), p. 413.

[119] Systems-Modeling. *Systems-modeling/sysml-V2-release: The latest incremental release of SysML V2. start here.* URL: `https://github.com/Systems-Modeling/SysML-v2-Release`.

[120] Wannita Takerngsaksiri, Cleshan Warusavitarne, Christian Yaacoub, Matthew Hee Keng Hou, and Chakkrit Tantithamthavorn. "Students' Perspectives on AI Code Completion: Benefits and Challenges". In: *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE. 2024, pp. 1606–1611.

[121] Tim Tegeler, Steve Boßelmann, Jonas Schürmann, Steven Smyth, Sebastian Teumert, and Bernhard Steffen. "Executable documentation: From documentation languages to purpose-specific languages". In: *International Symposium on Leveraging Applications of Formal Methods*. Springer. 2022, pp. 174–192.

[122] Thomas Teper, Kelly X Campo, Casey E Eaton, Garima Bhatia, and Bryan Mesmer. "Considerations for Implementation of Model-Based Systems Engineering in Different Sectors and System Types Based on Academic Literature". In: *Systems Engineering* (2025).

[123] Richard Thaler and C Sunstein. *NUDGE: Improving Decisions About Health, Wealth, and Happiness*. Vol. 47. June 2009. ISBN: 9780141040011.

[124] Amos Tversky and Daniel Kahneman. "Advances in prospect theory: Cumulative representation of uncertainty". In: *Journal of Risk and uncertainty* 5 (1992), pp. 297–323.

[125] Marcel Valový and Alena Buchalcevova. "The psychological effects of AI-assisted programming on students and professionals". In: *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2023, pp. 385–390.

[126] Christopher J Van Wyk. "Literate programming: an assessment." In: *Communications of the ACM* 33.3 (1990), pp. 361–363.

[127] Vasja Vehovar and Katja Lozar Manfreda. "Overview: online surveys". In: *The SAGE handbook of online research methods* (2017), pp. 143–161.

[128] Andreas Vogelsang, Tiago Amorim, Florian Pudlitz, Peter Gersing, and Jan Philipps. "Should I stay or should I go? On forces that drive and prevent MBSE adoption in the embedded systems industry". In: *Product-Focused Software Process Improvement: 18th International Conference, PROFES 2017, Innsbruck, Austria, November 29–December 1, 2017, Proceedings 18*. Springer. 2017, pp. 182–198.

[129] Frank Wawrzik, Khushnood Adil Rafique, Farin Rahman, and Christoph Grimm. "Ontology learning applications of knowledge base construction for microelectronic systems information". In: *Information* 14.3 (2023), p. 176.

[130] David Wile. "Lessons learned from real DSL experiments". In: *Science of Computer Programming* 51.3 (2004), pp. 265–290.

[131] Ross N Williams. "FunnelWeb user's manual". In: *V1. 0 for FunnelWeb* 3 (1992).

[132] Stephen Wolfram. *The mathematica book*. Wolfram Research, Inc., 2003.

[133] A Wayne Wymore. *Model-based systems engineering*. CRC press, 2018.

[134] Peng Zhang, Zsolt Lattmann, James Klingler, Sandeep Neema, and Ted Bapty. "Visualization techniques in collaborative domain-specific modeling environment". In: *SoutheastCon 2015*. IEEE. 2015, pp. 1–6.

[135] Xizhe Zhang, Siddartha Khastgir, and Paul Jennings. "Scenario description language for automated driving systems: A two level abstraction approach". In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2020, pp. 973–980.

Appendix A

Student Survey Data

ID	Timestamp	Do you have System Modeling experience?	Yes, How much? (in years)	Yes, which language did you use?	Other
1	2022/07/18 2:12:24 pm EET	No			
2	2022/07/18 2:14:40 pm EET	No			
3	2022/07/18 2:15:01 pm EET	No			
4	2022/07/18 2:27:01 pm EET	No			
5	2022/07/18 2:33:03 pm EET	No			
6	2022/07/18 2:34:58 pm EET	No			
7	2022/07/18 2:35:31 pm EET	No			
8	2022/07/18 2:37:20 pm EET	No			
9	2022/07/18 2:39:31 pm EET	No			
10	2022/07/18 2:40:05 pm EET	No			
11	2022/07/18 2:57:37 pm EET	Yes	2	Other	PN
12	2022/07/18 2:59:36 pm EET	No			
13	2022/07/18 3:02:28 pm EET	No			
14	2022/07/18 3:11:43 pm EET	No			
15	2022/07/18 3:14:24 pm EET	No			
16	2022/07/18 4:13:02 pm EET	No			
17	2022/07/18 5:13:10 pm EET	No			
18	2022/07/18 6:37:57 pm EET	No			
19	2022/07/18 9:11:54 pm EET	No			
20	2022/07/18 9:32:31 pm EET	No			
21	2022/07/19 1:34:10 am EET	No			
22	2022/07/19 9:04:14 am EET	Yes	2	UML	
23	2022/07/19 11:52:03 am EET	No			
24	2022/07/19 3:47:46 pm EET	No			
25	2022/07/19 5:34:06 pm EET	No			
26	2022/07/19 5:51:49 pm EET	Yes	2	UML	
27	2022/07/20 2:25:13 pm EET	No			
28	2022/07/21 9:33:16 am EET	No			
29	2022/07/21 11:55:53 am EET	Yes	0.8	SysML	
30	2022/07/22 5:12:09 pm EET	Yes	1	SysML	UML

Figure A.1.: *Unrefined data from the student survey 1*

ID	Would you prefer a textual or graphical representation of a model in order to understand it yourself?	Would you prefer a textual or graphical representation of a model in order to explain it to others?	Would you prefer a textual or graphical representation of a model in order to create a model?
1	Graphical representation	Graphical representation	Both
2	Graphical representation	Both	Textual representation
3	Both	Both	Both
4	Both	Both	Textual representation
5	Both	Both	Graphical representation
6	Graphical representation	Both	Textual representation
7	Both	Graphical representation	Both
8	Graphical representation	Graphical representation	Graphical representation
9	Graphical representation	Both	Both
10	Both	Graphical representation	Graphical representation
11	Both	Graphical representation	Both
12	Both	Both	Both
13	Both	Textual representation	Graphical representation
14	Both	Graphical representation	Graphical representation
15	Both	Both	Both
16	Both	Graphical representation	Textual representation
17	Both	Graphical representation	Textual representation
18	Graphical representation	Graphical representation	Graphical representation
19	Both	Graphical representation	Both
20	Both	Both	Both
21	Graphical representation	Both	Both
22	Both	Graphical representation	Textual representation
23	Both	Both	Both
24	Both	Graphical representation	Graphical representation
25	Both	Both	Graphical representation
26	Graphical representation	Graphical representation	Graphical representation
27	Textual representation	Textual representation	Textual representation
28	Graphical representation	Graphical representation	Graphical representation
29	Both	Graphical representation	Both
30	Both	Graphical representation	Both

Figure A.2.: *Unrefined data from the student survey 2*

ID	SysMD is Intuitive	SysMD is easy to write	SysMD is easy to read	A description can easily be modeled in SysMD
1	Agree	Agree	Agree	Agree
2	Agree	Disagree	Neither agree nor disagree	Disagree
3	Neither agree nor disagree	Disagree	Neither agree nor disagree	Neither agree nor disagree
4	Agree	Disagree	Agree	Can't say
5	Agree	Can't say	Strongly agree	Can't say
6	Agree	Disagree	Agree	Neither agree nor disagree
7	Agree	Strongly agree	Agree	Neither agree nor disagree
8	Agree	Agree	Neither agree nor disagree	Neither agree nor disagree
9	Agree	Agree	Agree	Neither agree nor disagree
10	Disagree	Disagree	Agree	Neither agree nor disagree
11	Can't say	Can't say	Can't say	Can't say
12	Agree	Agree	Agree	Can't say
13	Neither agree nor disagree	Can't say	Neither agree nor disagree	Can't say
14	Disagree	Agree	Agree	Disagree
15	Can't say	Can't say	Can't say	Can't say
16	Agree	Neither agree nor disagree	Agree	Strongly agree
17	Agree	Disagree	Agree	Disagree
18	Neither agree nor disagree	Strongly agree	Strongly agree	Agree
19	Disagree	Strongly agree	Agree	Can't say
20	Disagree	Disagree	Disagree	Disagree
21	Strongly disagree	Can't say	Disagree	Can't say
22	Disagree	Disagree	Disagree	Disagree
23	Disagree	Neither agree nor disagree	Neither agree nor disagree	Strongly disagree
24	Disagree	Can't say	Disagree	Can't say
25	Agree	Disagree	Agree	Agree
26	Disagree	Neither agree nor disagree	Agree	Disagree
27	Agree	Disagree	Agree	Agree
28	Neither agree nor disagree	Agree	Agree	Agree
29	Strongly agree	Strongly agree	Strongly agree	Agree
30	Agree	Agree	Agree	Can't say

Figure A.3.: *Unrefined data from the student survey 3*

ID	The statements used in SysMD are clear to understand	I'm able to understand a SysMD model	I would be able to explain a SysMD model to someone else
1	Agree	Agree	Agree
2	Agree	Can't say	can't say
3	Agree	Neither agree nor disagree	Disagree
4	Disagree	Disagree	Disagree
5	Agree	Agree	Strongly disagree
6	Agree	Agree	Neither agree nor disagree
7	Disagree	Disagree	Disagree
8	Agree	Agree	Agree
9	Strongly agree	Strongly agree	Strongly agree
10	Agree	Neither agree nor disagree	Disagree
11	Neither agree nor disagree	Neither agree nor disagree	Disagree
12	Neither agree nor disagree	Neither agree nor disagree	Disagree
13	Agree	Disagree	Neither agree nor disagree
14	Agree	Neither agree nor disagree	Disagree
15	Can't say	Can't say	can't say
16	Strongly agree	Strongly agree	Agree
17	Agree	Neither agree nor disagree	Strongly disagree
18	Agree	Agree	Agree
19	Can't say	Strongly disagree	Strongly disagree
20	Disagree	Disagree	Disagree
21	Disagree	Strongly disagree	Strongly disagree
22	Disagree	Disagree	Strongly disagree
23	Disagree	Strongly disagree	Strongly disagree
24	Disagree	Strongly disagree	Strongly disagree
25	Disagree	Neither agree nor disagree	Disagree
26	Agree	Agree	Agree
27	Agree	Agree	Neither agree nor disagree
28	Agree	Strongly agree	Strongly agree
29	Strongly agree	Strongly agree	Agree
30	Strongly agree	Strongly agree	Agree

Figure A.4.: *Unrefined data from the student survey 4*

ID	Which of the three models do you think is the most intuitive	Which of the three models is the easiest to understand	Which model would you use to explain it to someone else?
1	SysMD	SysMD	SysMD
2	SysML	SysML	SysML
3	Can't say	SysML	SysML
4	SysML	SysML	SysML
5	SysMD	SysMD	SysMD
6	SysML	SysMD	SysMD
7	SysMD	SysMD	SysMD
8	SysMLv2	SysMLv2	SysML
9	SysMD	SysMD	SysMD
10	SysMD	SysMD	SysMD
11	SysMD	SysMD	SysML
12	SysML	SysML	SysML
13	SysML	SysML	SysML
14	SysML	SysML	SysML
15	SysML	SysML	SysML
16	SysMD	SysMD	SysMD
17	SysMD	SysMD	SysMD
18	Can't say	SysML	SysML
19	SysMD	SysMD	SysML
20	SysMLv2	SysMD	SysMLv2
21	SysML	SysMD	SysMD
22	SysMD	SysMD	SysMD
23	SysML	SysML	SysML
24	SysML	SysML	SysML
25	SysML	SysMD	SysML
26	SysML	SysML	SysML
27	SysMLv2	SysMLv2	SysMLv2
28	SysML	SysML	SysML
29	SysMD	SysMD	SysML
30	SysML	SysMD	SysML

Figure A.5.: *Unrefined data from the student survey 5*

ID	In which modeling language would you most likely create a new model?	Which of the three models is the most clear in its use of keywords and relations?	Are you a student from the TU Kaiserslautern?	What do you study?
1	SysMD	SysMD	Yes	Informatik
2	SysML	SysML	Yes	Fahrzeugtechnik
3	SysMLv2	SysML	Yes	Cognitive Science
4	SysML	SysMLv2	Yes	Sozioinformatik
5	SysMD	SysMD	Yes	Informatik
6	SysML	SysMD	Yes	Computer Science
7	SysMD	SysMD	Yes	Master in computer science
8	SysMLv2	SysMLv2	Yes	Computer Science(B.Sc.)
9	SysMD	SysMD	Yes	Masters in Computer Science
10	SysMD	SysMD	Yes	Informatik
11	SysMD	SysMD	Yes	EIT/AUT
12	SysML	SysML	Yes	Computer Science
13	SysMD	SysML	Yes	Mathematics
14	SysML	SysML	Yes	Informatik
15	SysML	SysML	Yes	Commercial vehicle technology
16	SysMD	SysMD	Yes	Electrical Engineering Bachelor
17	SysMD	SysMD	Yes	chemistry
18	SysML	SysML	Yes	Cognitive Science
19	SysML	Can't say	Yes	Mathematics
20	SysMLv2	SysMLv2	Yes	Masters in Computer Science
21	SysML	SysMLv2	Yes	Computer Science
22	SysMD	SysMD	Yes	Mathematics
23	SysMLv2	SysMLv2	Yes	Mathematik
24	SysML	SysML	Yes	physics
25	SysMD	SysMD	Yes	Chemistry
26	SysML	SysML	Yes	Computer Science
27	SysMLv2	SysMLv2	Yes	Embedded systems
28	SysML	SysMD	Yes	Informatik
29	Can't say	Can't say	Yes	Commercial Vehicle Technology
30	SysML	SysML	Yes	Maschinenbau

Figure A.6.: *Unrefined data from the student survey 6*

Appendix B

INCOSE Survey Data

ID	How would you rate systems engineering in regards to being approachable to non-experts?	Late change requests	Low stakeholder participation	Difficulty with modelling language
1	1	Very important	Very important	Neither important nor unimportant
2	1	Unimportant	Very important	Very unimportant
3	2	Very important	Unimportant	Unimportant
4	1	Can't say	Can't say	Can't say
5	2	Neither important nor unimportant	Important	Important
6	3	Can't say	Can't say	Can't say
7	3	Neither important nor unimportant	Important	Neither important nor unimportant
8	2	Very important	Important	Unimportant
9	2	Important	Neither important nor unimportant	Very important
10	2	Neither important nor unimportant	Neither important nor unimportant	Neither important nor unimportant
11	2	Important	Can't say	Very important
12	4	Neither important nor unimportant	Neither important nor unimportant	Important
13	2	Important	Neither important nor unimportant	Important
14	1	Very important	Very unimportant	Very important
15	2	Neither important nor unimportant	Unimportant	Important
16	2	Important	Important	Very important
17	4	Important	Neither important nor unimportant	Can't say
18	2	Neither important nor unimportant	Very important	Very important
19	2	Very important	Very important	Very important
20	2	Very important	Important	Very important
21	3	Important	Very important	Important
22	3	Very important	Very important	Important
23	3	Unimportant	Unimportant	Very unimportant
24	3	Important	Very important	Important
25	2	Important	Important	Neither important nor unimportant
26	2	Important	Very unimportant	Important
27	4	Very important	Important	Important
28	3	Important	Very important	Very important
29	2	Very important	Neither important nor unimportant	Neither important nor unimportant

Figure B.1.: *Unrefined data from the INCOSE survey 1*

ID	Unintuitive semantics of modelling language	Outdated documentation	Misunderstanding between stakeholders	Difficulty with modelling language
1	Important	Important	Very important	Neither important nor unimportant
2	Important	Very important	Very unimportant	Very unimportant
3	Very important	Unimportant	Important	Unimportant
4	Can't say	Can't say	Can't say	Can't say
5	Neither important nor unimportant	Very important	Neither important nor unimportant	Important
6	Can't say	Can't say	Can't say	Can't say
7	Unimportant	Unimportant	Very important	Neither important nor unimportant
8	Unimportant	Important	Very important	Unimportant
9	Very important	Very important	Neither important nor unimportant	Very important
10	Neither important nor unimportant	Important	Important	Neither important nor unimportant
11	Very important	Very important	Important	Very important
12	Very important	Very important	Important	Important
13	Very important	Important	Neither important nor unimportant	Important
14	Very important	Very important	Very important	Very important
15	Very important	Neither important nor unimportant	Neither important nor unimportant	Important
16	Very important	Very important	Important	Very important
17	Can't say	Unimportant	Neither important nor unimportant	Can't say
18	Important	Neither important nor unimportant	Important	Very important
19	Very important	Very important	Very important	Very important
20	Important	Very important	Very important	Very important
21	Important	Important	Very important	Important
22	Important	Important	Important	Important
23	Very unimportant	Neither important nor unimportant	Neither important nor unimportant	Very unimportant
24	Neither important nor unimportant	Important	Important	Important
25	Neither important nor unimportant	Important	Important	Neither important nor unimportant
26	Neither important nor unimportant	Very important	Important	Important
27	Important	Very important	Very important	Important
28	Very important	Neither important nor unimportant	Very important	Very important
29	Neither important nor unimportant	Very important	Very important	Neither important nor unimportant

Figure B.2.: *Unrefined data from the INCOSE survey 2*

ID	Missing domain-expert input	Incomplete or ambiguous requirements	Missing tool integration	Outdated/Inconsistent models
1	Neither important nor unimportant	Important	Very important	Can't say
2	Unimportant	Important	Important	Unimportant
3	Very important	Very important	Neither important nor unimportant	Important
4	Can't say	Important	Can't say	Important
5	Important	Neither important nor unimportant	Important	Neither important nor unimportant
6	Can't say	Can't say	Can't say	Can't say
7	Important	Important	Unimportant	Neither important nor unimportant
8	Important	Very important	Unimportant	Can't say
9	Important	Very important	Unimportant	Neither important nor unimportant
10	Neither important nor unimportant	Important	Neither important nor unimportant	Important
11	Important	Very important	Important	Important
12	Important	Important	Unimportant	Neither important nor unimportant
13	Neither important nor unimportant	Important	Important	Important
14	Important	Important	Important	Important
15	Important	Important	Important	Neither important nor unimportant
16	Neither important nor unimportant	Neither important nor unimportant	Neither important nor unimportant	Very important
17	Neither important nor unimportant	Important	Important	Important
18	Important	Very important	Important	Neither important nor unimportant
19	Very important	Very important	Very important	Very important
20	Very important	Very important	Very important	Very important
21	Very important	Very important	Very important	Very important
22	Very important	Very important	Neither important nor unimportant	Important
23	Important	Important	Neither important nor unimportant	Very unimportant
24	Very important	Important	Neither important nor unimportant	Important
25	Important	Very important	Important	Neither important nor unimportant
26	Important	Very important	Very important	Very important
27	Very important	Very important	Very important	Very important
28	Important	Important	Very important	Neither important nor unimportant
29	Important	Very important	Important	Very important

Figure B.3.: *Unrefined data from the INCOSE survey 3*

ID	Model validation	Model verification	Missing knowledge basis	Misinterpretation of models
1	Can't say	Can't say	Important	Can't say
2	Very important	Unimportant	Important	Very unimportant
3	Important	Important	Important	Very important
4	Very important	Very important	Neither important nor unimportant	Very important
5	Very important	Very important	Neither important nor unimportant	Neither important nor unimportant
6	Can't say	Can't say	Can't say	Can't say
7	Unimportant	Unimportant	Very unimportant	Important
8	Can't say	Can't say	Can't say	Can't say
9	Unimportant	Unimportant	Very unimportant	Important
10	Very important	Very important	Very important	Important
11	Important	Important	Can't say	Can't say
12	Important	Very important	Neither important nor unimportant	Important
13	Very important	Very important	Important	Important
14	Very important	Very important	Important	Very important
15	Very important	Very important	Important	Important
16	Important	Important	Neither important nor unimportant	Unimportant
17	Important	Important	Important	Important
18	Important	Important	Important	Important
19	Very important	Very important	Very important	Very important
20	Very important	Very important	Very important	Very important
21	Very important	Very important	Very important	Very important
22	Important	Important	Neither important nor unimportant	Very important
23	Very unimportant	Very unimportant	Important	Very unimportant
24	Important	Important	Important	Important
25	Neither important nor unimportant	Neither important nor unimportant	Very important	Important
26	Very important	Very important	Important	Important
27	Very important	Very important	Very important	Very important
28	Very important	Very important	Important	Very important
29	Very important	Very important	Neither important nor unimportant	Important

Figure B.4.: *Unrefined data from the INCOSE survey 4*

ID	Missing reference material	Informal models	Documentation and model connected (linked/one place)	Shared knowledge basis
1	Can't say	Can't say	Important	Important
2	Important	Unimportant	Important	Important
3	Unimportant	Neither important nor unimportant	Important	Important
4	Important	Can't say	Important	Can't say
5	Neither important nor unimportant	Neither important nor unimportant	Very important	Neither important nor unimportant
6	Can't say	Can't say	Can't say	Can't say
7	Unimportant	Unimportant	Neither important nor unimportant	Neither important nor unimportant
8	Can't say	Can't say	Can't say	Important
9	Very unimportant	Unimportant	Important	Very unimportant
10	Can't say	Can't say	Neither important nor unimportant	Neither important nor unimportant
11	Neither important nor unimportant	Can't say	Very important	Important
12	Neither important nor unimportant	Can't say	Can't say	Neither important nor unimportant
13	Neither important nor unimportant	Neither important nor unimportant	Very important	Important
14	Very important	Important	Very important	Very important
15	Important	Neither important nor unimportant	Important	Neither important nor unimportant
16	Neither important nor unimportant	Unimportant	Very important	Neither important nor unimportant
17	Important	Important	Important	Important
18	Neither important nor unimportant	Neither important nor unimportant	Neither important nor unimportant	Neither important nor unimportant
19	Very important	Very important	Very important	Important
20	Very important	Very important	Very important	Important
21	Important	Important	Very important	Very important
22	Neither important nor unimportant	Important	Very important	Neither important nor unimportant
23	Important	Very unimportant	Can't say	Can't say
24	Neither important nor unimportant	Neither important nor unimportant	Important	Important
25	Neither important nor unimportant	Important	Important	Neither important nor unimportant
26	Important	Neither important nor unimportant	Very important	Very important
27	Important	Important	Very important	Very important
28	Neither important nor unimportant	Unimportant	Neither important nor unimportant	Important
29	Neither important nor unimportant	Neither important nor unimportant	Very important	Important

Figure B.5.: *Unrefined data from the INCOSE survey 5*

ID	Version management system	Simplified syntax for common entities and relationships	Stakeholder collaboration	Graphical model representation for creation
1	Very important	Very important	Very important	Important
2	Very important	Unimportant	Very important	Unimportant
3	Neither important nor unimportant	Neither important nor unimportant	Important	Very unimportant
4	Important	Can't say	Can't say	Very important
5	Important	Neither important nor unimportant	Important	Neither important nor unimportant
6	Can't say	Can't say	Can't say	Can't say
7	Important	Neither important nor unimportant	Neither important nor unimportant	Important
8	Very important	Very unimportant	Important	Important
9	Neither important nor unimportant	Important	Neither important nor unimportant	Very unimportant
10	Important	Unimportant	Neither important nor unimportant	Unimportant
11	Important	Unimportant	Neither important nor unimportant	Neither important nor unimportant
12	Very important	Important	Very important	Can't say
13	Important	Important	Neither important nor unimportant	Very important
14	Very important	Very important	Very important	Very important
15	Important	Neither important nor unimportant	Neither important nor unimportant	Unimportant
16	Important	Important	Neither important nor unimportant	Unimportant
17	Important	Important	Neither important nor unimportant	Important
18	Important	Neither important nor unimportant	Important	Important
19	Very important	Very important	Very important	Very important
20	Very important	Very important	Very important	Important
21	Very important	Important	Important	Very important
22	Important	Very important	Very important	Important
23	Can't say	Can't say	Can't say	Can't say
24	Important	Important	Very important	Unimportant
25	Neither important nor unimportant	Neither important nor unimportant	Neither important nor unimportant	Important
26	Very important	Important	Very important	Very important
27	Very important	Important	Very important	Important
28	Very important	Important	Neither important nor unimportant	Neither important nor unimportant
29	Important	Important	Neither important nor unimportant	Neither important nor unimportant

Figure B.6.: *Unrefined data from the INCOSE survey 6*

ID	Graphical model representation for visualisation	Textual model representation for creation	Textual model representation for visualisation	Tool independence
1	Important	Neither important nor unimportant	Neither important nor unimportant	Can't say
2	Important	Important	Very unimportant	Important
3	Very important	Important	Unimportant	Neither important nor unimportant
4	Important	Important	Important	Important
5	Important	Important	Neither important nor unimportant	Important
6	Can't say	Can't say	Can't say	Can't say
7	Very important	Neither important nor unimportant	Unimportant	Unimportant
8	Important	Neither important nor unimportant	Neither important nor unimportant	Neither important nor unimportant
9	Very important	Important	Unimportant	Neither important nor unimportant
10	Unimportant	Unimportant	Unimportant	Neither important nor unimportant
11	Neither important nor unimportant	Neither important nor unimportant	Neither important nor unimportant	Important
12	Unimportant	Very unimportant	Very unimportant	Can't say
13	Very important	Important	Unimportant	Very important
14	Very important	Very important	Very important	Neither important nor unimportant
15	Neither important nor unimportant	Unimportant	Unimportant	Important
16	Unimportant	Important	Neither important nor unimportant	Neither important nor unimportant
17	Important	Important	Important	Important
18	Important	Unimportant	Unimportant	Unimportant
19	Very important	Very important	Very important	Very important
20	Very important	Very important	Very important	Very important
21	Very important	Important	Important	Important
22	Important	Important	Important	Very important
23	Can't say	Can't say	Can't say	Can't say
24	Important	Important	Unimportant	Neither important nor unimportant
25	Important	Neither important nor unimportant	Neither important nor unimportant	Neither important nor unimportant
26	Very important	Important	Important	Important
27	Very important	Very important	Important	Very important
28	Very important	Very important	Unimportant	Very important
29	Neither important nor unimportant	Important	Unimportant	Important

Figure B.7.: *Unrefined data from the INCOSE survey 7*

ID	Support incomplete models	Commonly accepted methodology	Accessible reference models	Automated model generation from natural language
1	Can't say	Unimportant	Important	Very important
2	Very unimportant	Important	Very important	Unimportant
3	Neither important nor unimportant	Neither important nor unimportant	Very important	Neider important nor unimportant
4	Can't say	Can't say	Can't say	Neider important nor unimportant
5	Neither important nor unimportant	Neither important nor unimportant	Neither important nor unimportant	Unimportant
6	Can't say	Can't say	Can't say	Can't say
7	Important	Neither important nor unimportant	Neither important nor unimportant	Unimportant
8	Can't say	Can't say	Can't say	Unimportant
9	Important	Important	Unimportant	Important
10	Unimportant	Neither important nor unimportant	Neither important nor unimportant	Unimportant
11	Important	Important	Can't say	Can't say
12	Can't say	Important	Important	Neider important nor unimportant
13	Important	Neither important nor unimportant	Important	Important
14	Important	Neither important nor unimportant	Unimportant	Important
15	Neither important nor unimportant	Neither important nor unimportant	Important	Neider important nor unimportant
16	Unimportant	Important	Neither important nor unimportant	Neider important nor unimportant
17	Important	Important	Important	Neider important nor unimportant
18	Neither important nor unimportant	Important	Neither important nor unimportant	Important
19	Very important	Very important	Very important	Very important
20	Very important	Important	Important	Very important
21	Very important	Important	Important	Important
22	Important	Important	Important	Very important
23	Can't say	Can't say	Can't say	Can't say
24	Important	Important	Neither important nor unimportant	Important
25	Neither important nor unimportant	Neither important nor unimportant	Neither important nor unimportant	Important
26	Very important	Important	Very important	Can't say
27	Very important	Important	Important	Very important
28	Important	Unimportant	Very important	Very unimportant
29	Important	Important	Important	Unimportant

Figure B.8.: *Unrefined data from the INCOSE survey 8*

ID	Analysis/Computation of expressions	Automated consistency checking	Configurable abstraction level
1	Can't say	Very important	Can't say
2	Important	Very important	Unimportant
3	Important	Important	Neider important nor unimportant
4	Can't say	Can't say	Can't say
5	Neider important nor unimportant	Important	Neider important nor unimportant
6	Can't say	Can't say	Can't say
7	Neider important nor unimportant	Neider important nor unimportant	Important
8	Can't say	Very important	Important
9	Unimportant	Unimportant	Important
10	Unimportant	Neider important nor unimportant	Neider important nor unimportant
11	Can't say	Important	Important
12	Neider important nor unimportant	Important	Important
13	Very important	Very important	Neider important nor unimportant
14	Important	Very important	Important
15	Important	Important	Important
16	Important	Very important	Important
17	Neider important nor unimportant	Important	Important
18	Neider important nor unimportant	Important	Neider important nor unimportant
19	Very important	Very important	Very important
20	Very important	Important	Very important
21	Very important	Very important	Important
22	Very important	Very important	Important
23	Can't say	Can't say	Can't say
24	Important	Important	Important
25	Neider important nor unimportant	Neider important nor unimportant	Neider important nor unimportant
26	Very important	Very important	Can't say
27	Very important	Very important	Very important
28	Important	Very important	Neider important nor unimportant
29	Important	Very important	Very important

Figure B.9.: *Unrefined data from the INCOSE survey 9*

Appendix C

Curriculum Vitae

Personal Details

E-Mail dalecke@cs.uni-kl.de

Work and Teaching experience

2019–2025 **Wissenschaftlicher Mitarbeiter** RPTU Kaiserslautern-Landau
Computer Science Department, Design of Cyber-Physical Systems research group

2016–2019 **Wissenschaftliche Hilfskraft** RPTU Kaiserslautern-Landau
Computer Science Department, Tutor "Rechnersysteeme 1+2"

2015–2018 **Wissenschaftliche Hilfskraft** RPTU Kaiserslautern-Landau
Math Department, Tutor "Mathevorkurs"

Studying abroad

2019.01–2019.05 **UIT The Arctic University of Norway**
research stay resulting in the master thesis under supervision of Prof. Randi Karlsen

Academic education

2017–2019 **Master of Science in Informatik Ø2,0** TU Kaiserslautern
Masterarbeit: *Designing Dynamic and Personalized Nudges - A Model*

2011–2017 **Bachelor of Science Informatik Ø2,8** TU Kaiserslautern
Bachelorthesis: *An Analysis of Wikimedia´s Caching Infrastructure*

Administrative experience

2017.05–2018.05 **RPTU Kaiserslautern-Landau**
Student Representative Member of the Student Parliament

2012.05–2016.05 **RPTU Kaiserslautern-Landau**
Student Representative in the Computer Science Department

Civil Service

2009.09–2010.05 **Civil Service**
Seniorenzentrum St. Kilian- Taking care of the elderly

Education

1999-2009 **Abitur Ø2,2** Integrierte Gesamtschule Mannheim Herzogenried
Advanced courses: Math, English, German, Chemistry
Voluntary: Psychology

Science Communication

2020.03 **Winner of contest: "Wer schreibt die beste Wissenschaftsreportage"**
Topic: Nudging - ein Stups in die richtige Richtung

2022.03 **Participant of contest: "Wer schreibt die beste Wissenschaftsreportage"**
, Topic: Digitale Zwillinge - Die Zukunft der Industrie?

2023.03 **Participant of contest: "Wer schreibt die beste Wissenschaftsreportage"**
Topic: Haben wir nicht alle ein bisschen ADHS?