Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Bachelorarbeit

Bachelor Thesis

Entwicklung eines Frameworks für die automatisierte Architekturvalidierung

Development of a framework for automated architecture validation

Rushanth Rasaratnam
Matrikelnummer: 367117

Aachen, 13. Juli 2025

Gutachter:
Prof. Dr.-Ing. Stefan Kowalewski
Prof. Dr. rer. nat. Bernhard Rumpe

Betreuer:
David Klüner M.Sc.

Diese Arbeit wurde vorgelegt am
Lehrstuhl Informatik 11 – Embedded Software

This is to be done

Zusammenfassung

An abstract is basically a very short summary of the thesis topic. It describes short and precise the content of the thesis and what has been achieved. An abstract should not contain any discussion, quotation or reference to figures, chapters and so on. Please note that if you write your thesis in german, there should be a german and english version of the abstract. Although an abstract should generally not contain examples we provide here an example abstract for this document:

There exists a large variety of latex templates such as guidelines explaining or demonstrating the common structuring, dos and donts of bachelor and master theses. This document aims at providing an example for a typical structure of such theses containing useful knowledge about scientific writing and providing students with a correspondent latex template for bachelor or master theses at i11. Moreover, it gives some advice on very basic latex functionality for citation and figure handling.

Eidesstattliche Versicherung

Rasaratnam, Rushanth	367117
Name, Vorname	Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Bachelorarbeit mit dem Titel

Entwicklung eines Frameworks für die automatisierte Architekturvalidierung

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum	Unterschrift

1. Einleitung

An introduction motivates the topic of the thesis and explains the origin of the task. It starts often with a very general statement followed by explanations which difficulties (of course with relevance to the thesis topic) arise. Finally, the difficulties are narrowed down to the topic of the thesis. At this point the reader should understand the relevance of the problem being addressed by the thesis. An example for an introduction could look like the text following below.

Motivate the topic, what is this thesis about and why is that important?

At the end of their bachelor or master studies young students have to write a final thesis. For many of them this is their first written academic document of mentionable relevance. Thus, those students have no experience with academic writing and therefore require often some basic guidelines. Guidelines are usually provided directly by the thesis advisers up front and/or during thesis review. Providing such guidelines in a written and structured form to the students results hopefully in less review work for advisor and student, more consistent and clear theses and hence theses of better quality.

Dies ist eine Anmerkung was noch zu machen ist.

1.1. Aufgabenstellung

This document aims at providing a basic template for students writing their bachelor or master thesis at Lehrstuhl Informatik 11 (I11). It shall provide a basic overview on how such writings are structured and provide some useful hints.

1.2. Gliederung

The outline of a thesis should point out how the different chapters are related to each other and that the thesis is well structured. It should show that the different chapters form a single coherent document. An example how such an outline could look like for this document is given below.

Chapter 2 gives an introduction and overview on latex citations and figures to provide some basic knowledge how to handle those latex-constructs. The proximate Chap. 3 presents an overview of work related to this document. Chapter 4 introduces subsequently how citations have to be used in order to avoid wrong citation styles

or even plagiarism. It therefore requires information introduced before in chapter 2. Additionally, this chapter provides some hints dealing with figures in latex and their quality. The document is concluded by Chap. 6 which gives also an outlook how this document could be extended in the future.

2. Grundlagen

Dieses Kapitel führt in die für das Framework relevanten Konzepte und Technologien ein. Zunächst wird der Begriff der E/E-Architektur samt ihrer Evolution und ihres Aufbaus erläutert. Anschließend wird die Softwarearchitektur behandelt: Nach der Definition grundlegender Begriffe werden die Prinzipien der Serviceorientierten Architektur vorgestellt und darauf aufbauend die Rolle von Middleware in der Fahrzeugtechnik beleuchtet. Ein weiterer Abschnitt erläutert die Motivation, Ziele und Methoden zur Validierung von Architekturen. Den Abschluss bildet die Vorstellung des für die Implementierung verwendeten Webentwicklungs-Stacks.

2.1. Automobile E/E-Architektur

Für alle softwaregesteuerten Funktionen innerhalb eines Fahrzeuges bildet die elektrische/elektronische Architektur (E/E-Architektur) das physikalische und logische Fundament. Sie definiert die grundlegende Organisation von Hardwarekomponenten wie Electronic Control Unit (ECU), Sensoren, Energieversorgungssystemen und Kabelbäumen und steuert deren Kommunikation und Interaktion, um die erwarteten Funktionen zu realisieren [Jia19]. Diese Architektur hat im Laufe der Zeit mehrere Varianten durchlaufen und wird stetig weiterentwickelt. Im Folgenden werden die evolutionären Stufen der E/E-Architektur genauer betrachtet.

2.1.1. Evolution der Architekturentwicklung

Ursprünglich waren Fahrzeuge hauptsächlich mechanische Produkte, was sich jedoch mit der Einführung von Elektronik, wie Motorsteuergeräte und Airbags, änderte und die Ära der Elektrifizierung innerhalb der Automobilindustrie begann. Zunächst waren, aufgrund der geringen Anzahl, die Verbindungen zwischen den elektronischen Komponenten meist Punkt-zu-Punkt, sprich Komponenten wurden direkt miteinander verbunden. Dies führte jedoch durch die steigende Anzahl an ECUs, zu einem Anstieg der Komplexität und des Kabelbaums. Um die Kommunikationseffizienz zu verbessern und die Komplexität zu reduzieren, wurden Controller Area Network (CAN)-Busse eingeführt. In der Folge wurde auch an Technologien wie Local Interconnect Network (LIN) und FlexRay entwickelt und eingesetzt, um die verschiedenen Kommunikationsanforderungen zu erfüllen, sodass Komponenten innerhalb einer Domäne Informationen gemeinsam nutzen.

Das Konzept der Funktionsdomäne rückte, aufgrund der wachsenden Komplexität und Funktionen immer stärker in den Vordergrund. ECUs und Funktionen wurden in Domänen wie Fahrwerk, Karosserie und Infotainment unterteilt. Durch die Nachfrage nach domänenübergreifender Kommunikation wurde die Gateway-Funktion, welche zuvor von einzelnen ECUs übernommen wurde, zunehmend in einem dedizierten, zentralen Gateway gebündelt. Obwohl dieser Ansatz die Mainstream-Lösung war, führte die Zunahme der Funktionen und Rechenanforderungen zu einer starken Zunahme der ECUs. Dies hatte komplexe Kabelbäume mit erhöhtem Gewicht und Kosten zur Folge, außerdem konnte der Ausfall durch das ausfallen des zentralen Gateways zu einem Zusammenbruch des gesamten Netzwerks führen.

Um diesem Problem entgegenzuwirken, wurden Domain Control Units (DCUs) eingeführt, welche Funktionen einiger ECUs innerhalb einer jeweiligen Funktionsdomäne integrieren und somit die Anzahl der ECUs und die Belastung der Gateways reduzieren. Um die bestmögliche Kommunikation zu garantieren, werden DCUs über das Automotive Ethernet mit dem zentralen Gateway verbunden, welches eine höhere Bandbreite bietet.

Ein weiterer Ansatz, der neben der domänenbasierten Architektur existiert, ist die zonenbasierte Architektur. Um Komplexität und das Gewicht der Verkabelung zu reduzieren, werden Komponenten anstatt nach Funktionen, basierend auf ihrer physischen Position innerhalb des Fahrzeuges unterteilt. Anstatt wie bei DCUs die Funktionen einer Domäne zu übernehmen, übernehmen Zone Control Units (ZCUs) die Funktionen, die in den entsprechenden physischen Zonen stattfinden. Auch hier wird die Kommunikation über das Automotive Ethernet durchgeführt. Dieser Ansatz reduziert die Anzahl der ECUs und Kabelaufwand (Anzahl, Länge, Gewicht) und verringert so die Komplexität der E/E-Architektur deutlich

Erst die Evolution der E/E-Architektur ermöglicht moderne Softwarekonzepte. Das folgende Unterkapitel beschreibt daher die Softwarearchitekturen, die diese neue Hardware-Basis nutzt.

2.2. Softwarearchitekturen in Fahrzeugen

Die allgemeine Definition einer Softwarearchitektur ist nach Bass et al. [BCK21] eine systematische Gliederung eines Softwaresystems in einzelne Bausteine, die Festlegung ihrer Schnittstelle sowie die Prinzipien, nach denen diese Komponenten zusammenwirken und organisiert sind. Seitdem sich der Schwerpunkt von Fahrzeugen weg von mechanikzentrierten Systemen bewegt hat und hin zu intelligenten, softwaredefinierten Plattformen, gibt es innerhalb der Automobilindustrie einen großen Fortschritt bezüglich der Softwarefunktionalität. Um bei diesem Fortschritt den Überblick zu behalten, ist eine fundierte Softwarearchitektur unerlässlich.

Zu Beginn setzte man auf klassische, geschichtete Architekturen wie OSEK/VDX und AUTOSAR Classic, jedoch merkte man mit dem Aufkommen des Konzepts eines Software-Defined Vehicle (SDV) sowie der steigenden Software-Komplexität, dass die klassische Softwarearchitektur an ihre Grenzen stieß. Es gibt einige Gründe, wie die starre, monolithische Schichtenstruktur: Funktionen sind fest an Schichten gebunden, was Anpassungen einzelner Komponenten oder den Austausch von Modulen kaum umsetzbar machte.

In diesem Unterkapitel wird ein Einblick in die Service-orientierte Architekturen (SOAs) gegeben, die beschriebenen Schwächen der klassischen Softwarearchitektur überwinden. Anschließend wird die Middleware erläutert, die als technische Vermittlungsschicht dient.

2.2.1. Serviceorientierte Architektur

Die Notwendigkeit, Softwaremodule dynamisch bereitzustellen erfordert einen flexiblen Ansatz. Hier setzt die SOA, welche bereits in der Webentwicklung fester Bestandteil ist, an. Nach Rumez et al.[RGKS20] ein Architekturmuster, bei dem Anwendung als lose gekoppelte, wiederverwendbare Dienste organisiert werden. Jeder Dienst bietet eine klar definierte Funktionalität über gut beschriebene Schnittstellen und kann zur Laufzeit von beliebigen Clients zugegriffen werden.

Im Gegensatz zum klassischen signalorientierten Ansatz, bei dem die Konfiguration der Kommunikationspfade vollständig statisch zur Design-Zeit erfolgt, setzt der SOA-Ansatz auf eine dynamische Konfiguration während der Laufzeit. Durch das dynamische Aufrufen von Diensten zur Laufzeit erhält die Architektur mehr Flexibilität. So lassen sich Funktionen einfacher hinzufügen, entfernen oder bearbeiten, ohne dass eine komplette Neuzuweisung der Software erforderlich ist. Dank der Middleware-Schicht, auf die im nächsten Abschnitt näher eingegangen wird, sind die verschiedenen Anwendungen unabhängig von dem Steuergerät, auf dem ein Dienst läuft, sowie vom zugrunde liegenden Netzwerk aufgebaut. Diese Entkoppelung von der Plattform ermöglicht es der Software, wiederverwendbar und portierbar zu sein. Ein weiteres Merkmal des SOA-Ansatzes sind Kommunikationsmuster, die Daten nur bei Bedarf übertragen. Im Gegensatz dazu verschickt die klassische Softwarearchitektur in zyklischen Abständen Daten – selbst wenn kein Empfänger vorhanden ist. Dadurch kann die entstehende Netzwerklast auf das Nötigste reduziert werden, was die Effizienz steigert. Darüber hinaus kapselt jeder Dienst hinter einer Schnittstelle eine klar definierte Funktionalität. So lassen sich diese Dienste in unterschiedliche Kontexte und Projekten wiederverwenden, wodurch der Entwicklungs- und Testaufwand gesenkt wird. Ein weiterer Vorteil ist die Integration externer und On-Demand-Dienste: SOA unterstützen die dynamische Anbindung von Backend- und Cloud-Diensten, welche weitgehend transparent ist. Außerdem lassen sich SOA an wachsenden Anforderungen anpassen ohne dass das Gesamte System neu zu strukturieren. Aus den genannten

Gründen erweißt sich die SOA als flexiblere und anpassungsfähiger als die klassische Variante der Softwarearchitektur.

Um die Vorteile der SOA realisieren zu können benötigt es eine Zwischenschicht zwischen Anwendung und Netz. Im folgenden Abschnitt wird die Middleware-Schicht untersucht, welche die SOA-Prinzipen technisch umsetzt. Dabei wird zunächst auf Aufbau und Funktionen eingegangen. Anschließend wird die ASOA-Middleware genauer untersucht.

2.2.2. Automobile Middleware

Innerhalb moderner, verteilter Softwaresysteme ist die Middleware-Schicht eine wichtige Vermittlungsebene. Neely et al. [NDN06] definieren die Middleware als Schicht zwischen Anwendungssoftware und Systemsoftware. In Systemen mit Netzwerkanbindung vermittelt sie zusätzlich zwischen Anwendungssoftware, Betriebssystem und Netzwerkkommunikationsschichten. Dabei liegt ihre Hauptaufgabe darin, die Komplexität des zugrunde liegenden Systems zu abstrahieren [NDN06]. Im Bereich der Fahrzeugtechnik wird die Middleware noch einmal in zwei Sub-Schichten unterteilt [KMK+24]:

- Kommunikations-Middlewares (untere Schicht), die den Datenaustausch zwischen ECUs organisieren und Quality-of-Service (QoS)-Funktionen bereitstellen.
- Architekturplattformen (obere Schicht), die umfassende Frameworks für die Entwicklung und Bereitstellung automobiler Softwaresysteme liefern.

Nun wird untersucht, wie die Kommunikations-Middlewares und Architekturplattformen konkret arbeiten.

Als untere Sub-Schicht sorgt die Kommunikations-Middleware dafür, dass sämtliche Details des Datenaustauschs im In-Vehicle Network (IVN) abstrahiert und Anwendungen von der zugrunde liegenden Netzwerktopologie entkoppelt sind. Sie sorgt zudem dafür, dass strukturierte Daten wie z.B. Sensormesswerte automatisch seralisiert und über Ethernet, CAN-Gateway oder Shared-Memory-Kanäle verteilt werden. Dies wird durchgeführt, ohne dass Entwickler sich mit den Kommunikationsprotokollen befassen müssen. Außerdem ermöglicht sie die Definition und Einhaltung von QoS-Parametern. Des Weiteren sorgt ein intelligentes Routing- und Topologie-Management auf der Netzwerk-Ebene dafür, dass Nachrichten den bestmöglichen Pfad für die Übertragung wählen, während ein Real-Time-Scheduler die Übertragungen koordiniert.

Mit der Kommunikations-Middleware als Basis, löst die Architekturplattform Herausforderungen in Bezug auf die Entwicklung von Automobil-Softwaresystemen. Sie stellt eine umfassende Laufzeitumgebung bereit und bietet Entwicklern standardisierte Dienste und APIs. Das Ziel ist es die bestehende Komplexität weiter zu verringern, sodass Entwickler den Fokus auf die Anwendungslogik setzen können, anstatt sich mit

systemnahen Aufgaben zu beschäftigen. Abgesehen von der Kommunikation, gehört zunächst das umsetzen der Software-Designmustern, wie das zuvor angesprochene SOA-Prinzip, zu den Kernaufgaben. Ebenso Ressourcenmanagement und Orchestrierung , bei der die Ressourcen von inaktiven Diensten freigegeben werden. Das Ziel hierbei ist eine situationsabhängige optimale Ressourcenzuweisung. Als nächstes ist die deterministische Ausführung sowie die Unterstützung bei der Einhaltung von Echtzeitanforderung zu nennen, um Verzögerungen im Softwaresystem zu vermeiden. Des Weiteren bietet sie oft Funktionen zur Cybersicherheit, wie Authentifizierung, Zugriffskontrolle, Verschlüsserung etc. an um Schwachstellen innerhalb des IVNs zu reduzieren. Darüber hinaus sorgt die obere Schicht dafür, dass Over-the-Air-Updates, Installation/Deinstallation von Software-Paketen, sowie die Konfiguration von Anwendungen während der laufzeit durchgeführt werden und bietet dementsprechend Funktionen an. Eine weitere wichtige Aufgabe ist die Verwaltung des Systemzustands und die Überwachung des Lebenszyklus von Softwarekomponenten. Nicht zuletzt bietet die Architekturplattform Entwicklungswerkzeuge an, welche für die Entwicklung und Bereitstellung verwendet werden.

2.3. Validierung von Architekturen

Wie bereits zu Beginn der Arbeit dargelegt, ist die frühzeitige Prüfung von Architekturen von entscheidender Bedeutung. Das technische Ziel der Architekturvalidierung ist daher, die Korrektheit und Konsistenz eines Systementwurfs anhand definierter Qualitätsmethoden, wie ISO 26262 für funktionale Sicherheit oder ISO 25010 was sind die Zielefür Softwarequalität, zu bewerten, noch bevor die Implementierung beginnt [VRST15]. Dieses Ziel lässt sich in die folgenden Teilziele unterteilen: Früherkennung von Fehlern, Kostenreduzierung, Sicherstellung von Qualität und Standardkonformität, Verifikation des Systemverhaltens und der Anforderungen, Beherrschung der Systemkomplexität und Verbesserung des Verständnisses[VRST15] [Kan19] [BKB19]. Im folgenden werden die unterschiedlichen Ansätze zum validieren der Architekturen genauer betrachtet.

2.3.1. Validierungsverfahren

Es existieren veschiedene Ansätze zur Validierung eingesetzt, um die Gültigkeit von automobilen Systemen zu gewährleisten. Diese lassen sich in Vier Kategorien unterteilen[KPKS10] [VRST15] [WPK+16]:

- Analytische Validierung

  Bei diesem Ansatz werden formale mathematische Methoden, wie Model Checking oder SymTA/s, um garantierte Ober-/Untergrenzen für bestimmte Systemeigenschaften zu berechnen, insbesondere bei Echzeitverhalten.

Mit analytischen Methoden lässt sich auch das Abdeckungsproblem (coverage problem) lösen, in dem sie einen Worst Case konstruieren und die entsprechende Grenze berechnen [KPKS10]. Nachteil dieser Ansätze ist jedoch, dass sie eher zu pessimistischen Ergebnissen tendieren oder Probleme haben,komplexe Systemkontexte wie Offsets oder Korrelation von Aufgaben zu berücksichtigen [KPKS10].

- Simulatonsbasierte Validierung

  Dieser Ansatz beinhaltet die Erstellung von ausführbaren Modellen der Systemarchitektur und deren Simulation. Dabei wird das Verhalten des Systems beobachtet, um detaillierten Einblick bei Interaktionen und möglichen Nebenwirkungen zu gewinnen [BKB19] [KPKS10]. Da die Abdeckung aller möglichen Szenarien aufwendig und schwierig ist, kann nicht garantiert werden, dass der "Worst-Caseïmmer gefunden wird.

2.4. Webentwicklungs-Stack

3. Verwandte Arbeiten

Their are many books on other guidelines describing how a bachelor or master thesis should be structured which cannot all be mentioned here. Hence, this chapter limits its scope to a single example. For instance the university of columbia provides some guidelines on writing theses. According to the guideline a thesis starts with a title page with information about the title, author, department, delivery date, research mentor(s), advisors, their institutions and email adresses. The next structuring elements are the table of contents, the list of figures and the list of tables before the introduction of the thesis.

...

The main difference between the described guideline and this document is that the guideline addresses some more questions of methodological nature to be answered. This document however provides a more specific structure and layout template for theses written with latex at i11 in the field of computer science. Additionally, this document gives advice on some more practical topics.

4. Inhaltsspezifische Überschrift

As already stated before in chapter , every chapter or section should start with some introducing words. Due to the fact that this chapter (or potentially multiple chapters) presents the authors efforts, each section and chapter shall provide analogous to its introduction some concluding words, summarizing the findings and achievements of the specific section or chapter.

4.1. Different Citation Types

During the writing of a thesis work, ideas and contributions provided by other people then the author have to be declared as such. This is done using citations. In general, there are two different types of citations. The first type of citation is the literal citation. It is only used if text is copied directly without modification from others. In this case the citation is surrounded by quotes . For instance:

> Wikipedia states that „Wörtliche Zitate sollten eingesetzt werden, wenn nicht nur der Inhalt der Aussage, sondern auch deren Formulierung von Bedeutung ist“ .

Nonetheless, the author has to make unambiguously clear where the citation originates from. In case of the wikipedia citation above this is done with: .

The second citation type is the more frequent used reference using the \cite{} command presented in Sect. This citation type is used if no literal citation is used. For instance:

> If not only the content of a statement is of importance but also the original wording, then literal citation using quotes has to be used .

Such references have to follow directly the first statement taken from other sources.

The following example shows how this looks for bigger paragraphs. The same colored parts refer to the same source.

Welche Theorien standardmäßig von einem Programm zur Lösung von SMT-Problemen unterstützt werden hängt vom konkret verwendeten Programm ab . Zur Vereinheitlichung der Beschreibung von Theorien, so wie der Eingabe- und Ausgabesprache für Programme die SMT-Probleme lösen wurde der SMT-LIB Standard verfasst. Aktuell befindet sich dieser von der SMT-LIB Initiative

formulierte Standard in Version 2.0 . Zu diesen Theorien gehören die Theorie der Felder (engl. *arrays*), der Bit-Vektoren fester Größe, der booleschen Operatoren, der Fließkommazahlen, der Ganzzahlen, der reellen Zahlen und der Kombination von reellen und ganzen Zahlen.

Please ensure that the visual look of the automatically generated bibliography is consistent and the printed information is complete. For instance:

- the separation of words at the end of a line should be adequate
- the ISBN, DOI, etc. are formatted the same way for every bibliography entry
- every web-sources accessing-date is stated
- URLs are displayed the way they are intended to (e.g. underscores in copied links are interpreted by latex as commands and not displayed unless the url is surrounded by \url{} or backslashes are interpreted as escaping characters and should be replaced with \textbackslash)

It is the responsibility of the author of a thesis to ensure that all relevant information about the used literature is readable in the printed version of the thesis.

The next section gives advice on figures used in the document and extends the citation concept accordingly.

4.2. Citations and Advice on Illustrations

Similar to textual content, pictures may originate from other publications, too. Thus, they have to be declared as results of work from others. In case the figure is not modified in any way this can be done by adding a correspondent citation in the figures caption. If the figure has been redrawn or modified the reference can be given as shown below:

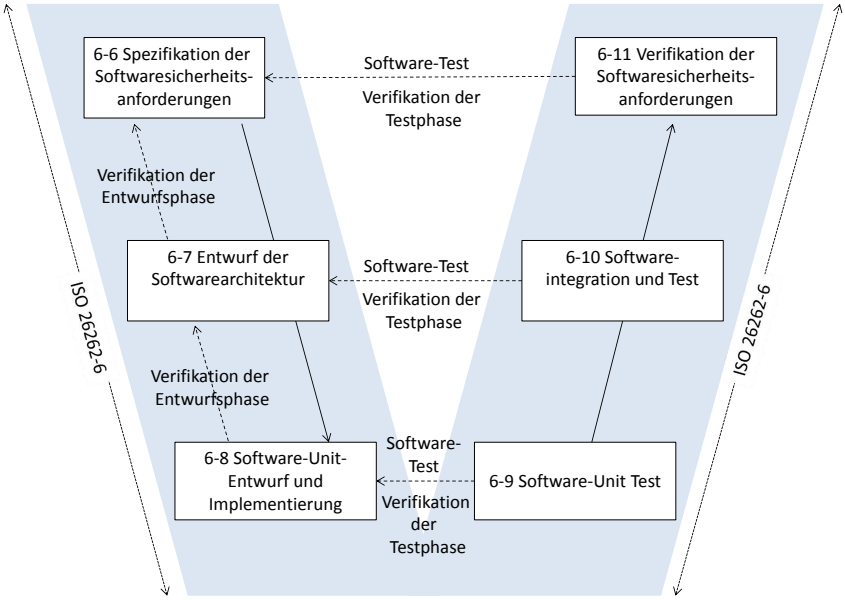

Abbildung 4.1.: V-Modell correspondent to ISO 26262-6 (drawing after)

The author might even differentiate between different figure-citation-types. Those could be:

- taken from *(exact copy of the figure)*
- partly taken from *(parts of the figure are taken from the specified source, others were added by the author)*
- adaption/extension of
- redrawing of *(the figure has been redrawn)*

If such descriptions or correspondent abbreviations are used the author should specify upfront the semantic and syntax at the beginning of the thesis. The important thing here is not which words were used but that it is clear which parts are whose work.

It follows an example for a bad structuring of paragraphs and transition between such since the quality of figures does not relate to figure citations. Also it is not recommendable to use fonts, colors and other *styling* elements (even in figures) which cannot be read easily.

Figures should always be provided as vector graphics, e.g. *.svg or *.pdf files. The benefit of such formats is their scalability without loss of quality. Unfortunately, many

pictures are not provided as such. Hence, schemata etc. need often to be redrawn. Some tools supporting the creation of vector graphics are:

- inkscape (open source)
- powerpoint (create a slide composed of shapes and store it as *.pdf file) or word/excel
  (this should work with open office too)
- to be completed ...

Please note that importing non vector graphics, e.g. bitmaps, by the mentioned tools will not convert them into vector graphics. Another aspect which should be addressed regarding figures in latex documents is the size of text inside figures. Due to scale operations on figures (see 4.1 where the width of the figure is set to 80 percent of the width of the region where text can be shown) the size of text labels in figures may change. However, the size of the smallest text on a figure should never be smaller than the smallest text-size of the rest of the document. Moreover the sizes of texts on figures should not change arbitrary between illustrations.

After this part of many theses follows an evaluation. Some important points regarding the evaluation part are mentioned in the following chapter.

5. Evaluation

Since there is no evaluation for this document, this chapter lists some points which may be considered when evaluating software/algorithms/procedures.

- Comparison with other approaches to achieve the same goal as this work
- Measure resources needed by a developed tool/procedure/etc. regarding (time consumption, cpu usage, memory usage, ...)
- What about the quality of the results?
- Are there any problems with the procedure/applicability or restrictions?
- Evaluate (if possible) using „real world examples“ and not only academic ones.

Please note that they do not all have to apply or be suitable for any thesis topic. It might be that evaluation results are not presentable in a structured and readable way. For instance, if tables become to large. In such cases it might be helpful to place the results in the appendix A and refer to them. This applies also for figures, test specifications and tables at any other place in the thesis and is the only exception allowing forward references. The evaluation should state very clear what is possible and what is not. If there are any, the limits of the developed approach should be shown using suitable examples. Do not be afraid to show limitations of approaches. This does generally not undermine the authors achievement if it is clear and can be explained why the limitations exist. The author should avoid to use vague words like better, worse, etc. to describe the evaluation results. This applies in general for the other chapters of the thesis, too.

The next chapter of this document is the final chapter, which should always conclude the work presented before.

6. Conclusion

The aim of this document was to provide students at the end of their studies with a template for their written thesis. Due to the common lack of experience in the field of academic writing this work is intended to provide a template for structured writing. Therefore, this entire document is structured as a thesis should usually be.

Moreover it provides some hints how citations should be used and how figures should be dealt with to achieve high quality versions of latex documents. In contrary to this sentence, a conclusion should not introduce new information, about the topics discussed before, which has not yet been presented.

The following (optional) section provides some further ideas for potential extension of this work.

6.1. Future Work

There are many possible ways how this short document could be extended in the future. One may think of additional explanations regarding latex and its use, with the extend to an entire latex tutorial. A further extension could be the definition of helpful latex commands or a documentation on commonly used latex commands and packages.

Literaturverzeichnis

[BCK21] Bass, Len ; Clements, Paul ; Kazman, Rick: Software Architecture in Practice, 4th Edition. 1st edition. Erscheinungsort nicht ermittelbar : Addison-Wesley Professional, 2021. – ISBN 978–0–13–688597–9

[BKB19] Bucher, Harald ; Kamm, Simon ; Becker, Jürgen: Cross-Layer Behavioral Modeling and Simulation of E/E-Architectures Using PREEvision and Ptolemy II. In: SNE Simulation Notes Europe 29 (2019), Juni, Nr. 2, S. 73–78. http://dx.doi.org/10.11128/sne.29.tn.10472. – DOI 10.11128/sne.29.tn.10472. – ISSN 23059974, 23060271

[Jia19] Jiang, Shugang: Vehicle E/E Architecture and Its Adaptation to New Technical Trends. In: WCX SAE World Congress Experience, 2019, S. 2019–01–0862

[Kan19] Kang, Eun-Young: A Formal Verification Technique for Architecture-based Embedded Systems in EAST-ADL

[KMK+24] Klüner, David P. ; Molz, Marius ; Kampmann, Alexandru ; Kowalewski, Stefan ; Alrifaee, Bassam: Modern Middlewares for Automated Vehicles: A Tutorial. https://arxiv.org/abs/2412.07817v1, Dezember 2024

[KPKS10] Kollmann, Steffen ; Pollex, Victor ; Kempf, Kilian ; Slomka, Frank: Comparative Application of Real-Time Verification Methods to an Automotive Architecture. (2010)

[NDN06] Neely, Steve ; Dobson, Simon ; Nixon, Paddy: Adaptive Middleware for Autonomic Systems. In: annals of telecommunications - annales des télécommunications 61 (2006), Oktober, Nr. 9-10, S. 1099–1118. http://dx.doi.org/10.1007/BF03219883. – DOI 10.1007/BF03219883. – ISSN 0003–4347, 1958–9395

[RGKS20] Rumez, Marcel ; Grimm, Daniel ; Kriesten, Reiner ; Sax, Eric: An Overview of Automotive Service-Oriented Architectures and Implications for Security Countermeasures. In: IEEE Access 8 (2020), S. 221852–221870. http://dx.doi.org/10.1109/ACCESS.2020.3043070. – DOI 10.1109/ACCESS.2020.3043070. – ISSN 2169–3536

[VRST15] Venkitachalam, Hariharan ; Richenhagen, Johannes ; Schlosser, Axel ; Tasky, Thomas: Metrics for Verification and Validation of Architecture in Powertrain Software Development. In: Proceedings of the First International Workshop on Automotive Software Architecture. New York, NY, USA : Association for Computing Machinery, Mai 2015 (WASA '15). – ISBN 978–1–4503–3444–0, S. 27–33

[WPK+16] Weissnegger, Ralph ; Pistauer, Markus ; Kreiner, Christian ; Schuss, Markus ; Römer, Kay ; Steger, Christian: Automatic Testbench Generation for Simulation-based Verification of Safety-critical Systems in UML:. In: Proceedings of the 6th International Joint Conference on Pervasive and Embedded Computing and Communication Systems. Lisbon, Portugal : SCITEPRESS - Science and Technology Publications, 2016. – ISBN 978–989–758–195–3, S. 70–75

A. Appendix1

Here could be a large figure or a very big table like this:

Tabelle A.1.: Table caption

column1	column2	column3
row 1	X	X
row 2	X	X
row 3	X	X
row 4	X	X
row 5	X	X
row 6	X	X
row 7	X	X
row 8	X	X
row 9	X	X
row 10	X	X
row 11	X	X
row 12	X	X
row 13	X	X
row 14	X	X
row 15	X	X
row 16	X	X
row 17	X	X
row 18	X	X
row 19	X	X
row 20	X	X
row 21	X	X
row 22	X	X
row 23	X	X
row 24	X	X
row 25	X	X
row 26	X	X
row 27	X	X
row 28	X	X

row 29	X	X
row 30	X	X
row 31	X	X
row 32	X	X
row 33	X	X

B. Digital Mediums as part of the thesis

This appendix is not empty but not referenced before. Such things shall not happen in a final version of a thesis. The same applies for figures, tables and other provided materials.

If you provide digital data with the printed version of you thesis (e.g. a CD/DVD) the contents of the digital recording have to be organized in a structured way to. The digital medium should contain a README file in the root folder. This textfile (*.txt) should state how the data is organized on the medium. Potential content for an attached digital recording is a digital version of the thesis or digital copies of the web-sources (can be created using a pdf printer). In case the digital medium contains program code or executables form third parties, please ensure that you do not violate any licenses.

C. Use of Symbols, Abbreviations and Index

Symbols can be introduced using the following Latex commands:

```
\newglossaryentry{symbol:pi}{ name=\ensuremath{\pi},
description={Kreiszahl [einheitenlos]},
sort=symbolpi,type=symbolslist,
}
```

The created symbol can then be referenced by:

```
\sym{pi}
```

which will be displayed as π. Symbols being created this way are automatically added to the list of symbols behind the list of contents.

In a similar manner abbreviations can be used. To define a new abbreviation, use

```
\newacronym{i11}{I11}{Lehrstuhl Informatik 11}
```

which defines a new acronym. It can be used with the command \abk{i11} and looks like I11 when referenced multiple times. The first use of the command, however, will appear as can be seen in the first sentence of section 1.1. For the full command-reference see the documentation of the acronym package. Alternatively, see `http://texblog.org/2014/01/15/glossary-and-list-of-acronyms-with-latex/` for explanations to capitalize and pluralize acronyms.

An index (germ. Stichwortverzeichnis) can be created using the \printindex command. To add text phrases to the index write \index{phrase for the index} which will be displayed as but create an entry in the index chapter referring to the page the phrase is placed to (see next page). The index creation requires two latex compilations and the use of makeindex. However, this should be no issue using this template and compiling with compile.bat.

Index