Thesis for The Degree of Licentiate of Engineering

Understanding Technical Debt and Assumption-Related Challenges in the Domain of Cyber-Physical Systems

Md Abdullah Al Mamun

Division of Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and University of Gothenburg
Göteborg, Sweden, 2015

Understanding Technical Debt and Assumption-Related Challenges in the Domain of Cyber-Physical Systems

Md Abdullah Al Mamun

Technical Report No 137L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Software Engineering
Chalmers University of Technology and University of Gothenburg
Göteborg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Göteborg, Sweden 2015.

To Arina, the Reason of My Enlightenment

Abstract

In today's world software is contributing a substantial part of the new functionalities and innovations of the automotive industry. So the size and the complexity of the software is dramatically increasing with time, which comes with additional challenges, e.g., implicit assumptions or technical debt. The problems of assumptions have been identified as one of the key reasons to software system projects and catastrophic system failures. On the other hand, unreasonable accumulation of technical debt can seriously reduce the reusability and maintainability of the software.

This thesis elaborates the idea of unveiling and understanding technical debt and challenges of assumptions for software by applying model-driven engineering (MDE) on the example of a self-driving car. First, we explore the challenges of assumptions in various areas of software and system development and show how assumptions are related to other knowledge artifacts. Then we focus on reporting advantages and disadvantages of different approaches to capture and automatically check architectural assumptions.

Incorporating formal methods in a rigorous assumption management tool supports various aspects of assumptions such as traceability. From our experience with formalizing assumptions, such an approach toward managing assumptions needs to combine several formal methods/languages. Developing and using a dedicated tool for assumption management is possible but the practicality of using such a tool in terms of time, budget, and manpower needs to be researched.

Considering the heavy weight of a dedicated rigorous formal approach, a practical step toward managing assumptions is to better incorporate the capabilities of the tools that are already being used in a project. Using existing capabilities is a light-weight approach because it does not cost extra money to procure new tools, train developers to use the new tools, and adjust the development process to incorporate it. To check the feasibility of such a light-weight assumption management approach, we perform a study on the development of a self-driving miniature vehicle. We particularly check whether MDE tools can be leveraged to capture assumptions related to the sensor management. We also explain how capturing assumptions reduces technical debt related to knowledge distribution and documentation.

From our experience with the self-driving miniature vehicle development, we see that MDE reduces knowledge debt through successfully capturing structural architectural assumptions and it reduces code debt and environmental debt through automated code generation. Thus, MDE is able to leverage the challenges of assumptions to the extent of capturing and checking them automatically, hence, reducing knowledge debt without necessarily using a dedicated assumptions management tool.

Our contributions include realizing a light-weight assumption management approach through MDE and our preliminary results show that legacy or 3^{rd}-party code has influence in the development of technical debt.

Keywords

Software Engineering, Technical Debt, Assumptions, Cyber-Physical Systems

Acknowledgments

First, I would like to express my heartfelt gratitude to my main supervisor and mentor Professor Jörgen Hansson who has always been by my side like a friend at times when I needed support, encouragement, and guidance.

Second, my humble thanks go out to my second supervisor Assoc. Professor Christian Berger for his invaluable guidance, feedback, and positiveness that were extremely helpful to reach this milestone.

I also would like to thank my examiner Assoc. Professor Laura Kovács for her valuable feedback. Special thanks to Professor Matthias Tichy for his altruistic help related to EMF and guideline toward formalizing assumptions.

I would like to thank Professor Michael Chaudron, the head of the Software Engineering division, and all my colleagues for creating a nice work environment. I am very grateful to Rebecca Cyren to whom I always ran with loads of practical questions. Thanks to my ex-colleagues Ali Shahrokni, Ana Magazinius, and Niklas Mellegård for their help, suggestions, and kindness. Special thanks to Håkan Burden for his utmost friendliness and Antonio Martini for brainstorming regarding technical debt.

Many thanks to Ulf Eliasson and Jonn Lantz from Volvo Cars for taking part in long discussions related to assumptions. I would like to specially mention Ulf Eliasson for being a trusted listener.

I am ever grateful to my parents, whom I have deprived of my company and care. I am so thankful to my sisters for their responsibilities toward my parents in my absence. Special thanks to my wife for her unconditional support, love and care for our daughters and me. I am thankful to my friends and neighbours, who I have not explicitly mentioned but have been a big part of my social life.

The research presented in this thesis has been funded by VINNOVA, the Swedish Governmental Agency for Innovation Systems.

List of Publications

Included papers

This thesis is based on the following papers:

I Md Abdullah Al Mamun and Jörgen Hansson. "Review and Challenges of Assumptions in Software Development". In the proceedings of the Second Analytic Virtual Integration of Cyber-Physical Systems Workshop (AVICPS), R. Mangharam, and P. Feiler, Eds., pp. 53-60, Vienna, Austria, 2011.

II Md Abdullah Al Mamun, Matthias Tichy, and Jörgen Hansson. "Towards Formalizing Assumptions on Architectural Level: A Proof-of-Concept". Technical Report, ISBN/ISSN: 1654-4870, University of Gothenburg. Göteborg, Sweden, 2012

III Christian Berger, Md Abdullah Al Mamun, and Jörgen Hansson. "COTS-Architecture with a Real-Time OS for a Self-Driving Miniature Vehicle". In the proceedings of the 2nd Workshop on Architecting Safety in Collaborative Mobile Systems (ASCoMS), E. Schiller and H. Lönn, Eds., pp. 411-422, Toulouse, France, Sept. 2013.

IV Md Abdullah Al Mamun, Christian Berger, and Jörgen Hansson. "MDE-based Sensor Management and Verification for a Self-Driving Miniature Vehicle". In the proceedings of the 13th Domain-Specific Modeling (DSM) workshop at SPLASH 2013, J. Gray, S. Kelly and J. Sprinkle, Eds., pp. 1-6, Indianapolis, USA, 2013.

V Md Abdullah Al Mamun, Christian Berger, and Jörgen Hansson. "Engineering the Hardware/Software Interface for Robotic Platforms - A Comparison of Applied Model Checking with Prolog and Alloy". In the proceedings of the Fourth International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLRob 2013), C. Schlegel, U. P. Schultz and S. Stinckwich, Eds., pp. 26-34, Tokyo, Japan, 2013.

VI Md Abdullah Al Mamun, Christian Berger, and Jörgen Hansson. "Explicating, Understanding and Managing Technical Debt from Self-Driving Miniature Car Projects". In the proceedings of the Sixth International Workshop on Managing Technical Debt (MTD 2014), C. Seaman Ed., Victoria, British Columbia, Canada, 2014.

Other papers

The following paper is published but not appended to this thesis due to contents not related to the thesis.

I Md Abdullah Al Mamun and Jörgen Hansson. "Reducing Simulation Testing Time by Parallel Execution of Loosely Coupled Segments of a Test Scenario". In the proceedings of the International Workshop on Engineering Simulations for Cyber-Physical Systems (ES4CPS'14), C. Berger and I. Schaefer Eds., pp. 33-37, Dresden, Germany, 2014.

Contents

Abstract v

Acknowledgment vii

List of Publications ix

1 Introduction 1
- 1.1 Background and Definitions . . . 1
  - 1.1.1 Assumption . . . 1
  - 1.1.2 Classification of Assumptions . . . 2
  - 1.1.3 Technical Debt . . . 4
  - 1.1.4 Classification of Technical Debt . . . 5
- 1.2 Motivation and Problem Domain . . . 7
- 1.3 Research Questions . . . 12
- 1.4 Overview of Papers . . . 12
  - Paper I: Review and Challenges of Assumptions in Software Development . . . 12
  - Paper II: Towards Formalizing Assumptions on Architectural Level: A Proof-of-Concept . . . 13
  - Paper III: COTS-Architecture with a Real-Time OS for a Self-Driving Miniature Vehicle . . . 13
  - Paper IV: MDE-based Sensor Management and Verification for a Self-Driving Miniature Vehicle . . . 14
  - Paper V: Engineering the Hardware/Software Interface for Robotic Platforms - A Comparison of Applied Model Checking with Prolog and Alloy . . . 15
  - Paper VI: Explicating, Understanding and Managing Technical Debt from Self-Driving Miniature Car Projects . . . 16
- 1.5 Research Methodology . . . 16
  - Content Analysis . . . 16
  - Case Study . . . 17
  - Design Science . . . 18
  - Experiment . . . 18
  - Interview . . . 19
  - Validity Evaluation . . . 19
- 1.6 Conclusion and Outlook . . . 21
  - Contributions . . . 21
  - Future Directions . . . 23

List of References **25**

2 Paper I **37**
2.1 Introduction . . . 38
2.2 Background . . . 39
2.2.1 Definition of Assumption . . . 39
2.2.2 Requirements, Constraints, Assumptions, Design Rationale 39
2.3 Assumptions in Software Engineering . . . 40
2.3.1 Assumption Modeling . . . 40
2.3.2 Architectural Mismatch . . . 42
2.3.3 Assumptions and Requirements . . . 43
2.3.4 Software Security . . . 44
2.3.5 Architectural Design Decision & Rationale Management 44
2.3.6 Other Work . . . 45
2.4 Challenges of Assumptions . . . 46
2.4.1 Assumption-Aware Component Development . . . 46
2.4.2 Separation of Assumptions from Artifacts . . . 46
2.4.3 Evidence-based Software Engineering . . . 47
2.4.4 Assumptions in the Organizations Safety Culture . . . 47
2.4.5 A Holistic Assumption Management Approach . . . 47
2.4.6 Prioritization of Assumptions . . . 48
2.4.7 Assumption-based Verification and Validation . . . 48
2.4.8 Assumption-based Trust Building and Maintenance . . 49
2.5 Summary . . . 49

3 Paper II **51**
3.1 Introduction . . . 52
3.2 Related Work . . . 53
3.2.1 Formal Approaches . . . 53
3.2.2 Semi-Formal Approaches . . . 54
3.3 Taxonomy of Assumptions . . . 54
3.4 Formalizing Assumptions . . . 56
3.4.1 Capturing the Architecture . . . 58
3.4.2 Capturing and Checking Assumptions . . . 60
3.5 Conclusions and Future Work . . . 65
Appendix . . . 67

4 Paper III **71**
4.1 Introduction . . . 72
4.2 Related Work . . . 72
4.3 COTS-Architecture for the Hardware . . . 73
4.3.1 Overall Architecture . . . 73
4.3.2 Application Board . . . 75
4.3.3 Hardware/Software Interface Board . . . 75
4.4 COTS-Architecture for the Real-Time Software with ChibiOS/RT 75
4.5 Development and Evaluation of Self-Driving Software . . . 76
4.6 Conclusion and Outlook . . . 79
Appendix . . . 81

5 Paper IV **83**
5.1 Introduction and Motivation 84
5.2 Sensor Management Language 85
5.2.1 Domain Meta-Model 85
5.2.2 Ensuring Static Semantics 86
5.3 Model Transformations 87
5.3.1 Model-to-Text Transformation 87
5.3.2 Use Case A: Validating a Sensor Layout 88
5.3.3 Use Case B: Verifying a Sensor Layout for a Target Hardware 88
5.4 Related Work 91
5.5 Conclusion and Outlook 93

6 Paper V **95**
6.1 Introduction and Motivation 96
6.2 Engineering The Robotic Hardware/Software Interface– A Combinatorial Optimization Problem 98
6.2.1 The Domain of Pin Assignment Configurations 99
6.2.2 Complexity Considerations 100
6.3 Evaluating Applied Model Checking for Pin Assignment Config. 101
6.3.1 Designing the Formal Experiment 102
6.3.2 Verification Approach Using Prolog 103
6.3.3 Verification Approach Using Alloy 105
6.3.4 Analysis and Discussion 107
6.3.5 Threats to Validity 109
6.4 Related Work 110
6.5 Conclusion and Outlook 111

7 Paper VI **113**
7.1 Introduction 114
7.1.1 Research Goal and Research Questions 115
7.1.2 Contributions of the Article 115
7.1.3 Structure of the Article 115
7.2 Related work 116
7.3 Methodology 116
7.3.1 Design of the study 117
7.3.2 Data Collection 118
7.4 Results 120
7.5 Analysis and Discussion 124
7.6 Conclusion and Future Work 126

Chapter 1

Introduction

The automotive industry is going through a major transition where all of the main car OEMs (original equipment manufacturer) are putting efforts toward self-driving vehicles. Some of the OEMs have the vision to sell fully functional driver-less cars by 2020 [1]. Software is the primary driving force for implementing different functionalities of today's cyber-physical systems, e.g., automated parking or automated braking. Such vehicles are highly dependent on the use of different types of sensors to realize various driving conditions, e.g., weather, traffic and road markings.

The dramatic transformation of today's cars into self-driving traffic actors is bringing a higher level of complexity which researchers and engineers need to address. Unlike other engineering disciplines, software engineering is yet to widely adopt sophisticated engineering practices to estimate, predict, and build software both at the product- and the project-level.

1.1 Background and Definitions

System development, and specifically software development for highly complex systems, suffers from invalid assumptions, which have been reported as the root cause of complex system failures [2, 3].

1.1.1 Assumption

Dewar et al. [4] have worked on assumption-based planning and define assumption in the context of organizational planning. They argue that considering assumptions are of significance for planning where the present environment is uncertain to predict the probable future. According to their definition, an assumption is an assertion about some characteristics of the future that underlies the current operations or plans of an organization. Here assertion can be a fact or a judgment. Examples of a fact and a judgment can respectively be "the sun has been the major source of energy on earth for more than one million years" and "the sun will be the major source of energy on earth for the next ten thousand years". The examples of fact and judgment are backed by scientific evidences and, thus, they are less challenging than assertions not supported or less supported by direct evidences. This definition of assumptions is

intentionally kept broad to cover assumptions of different types like descriptive, evaluative, predictive, explanatory, explicit, and implicit.

Hofmann [5] defines assumptions in the context of modeling. He considers the term model an abstraction of an original system where the original system can be anything including a software system. According to his definition, an assumption is a statement in the sense of an assertion that is not yet empirically confirmed. Furthermore, an assumption is:

- concerned with a subject S
- about an original O
- with the intention (purpose) I relative to
  - a problem P situated in O and
  - a time-frame T.

King and Turnitsa [2008] define an assumption in the context of models, simulations and software agents as follows:

$$Assumption <=> (assumeFN, referent, scope, proposition) \tag{1.1}$$

This can be read as "an assumption is a proposition about a referent with a given scope and a description how the assumption is used by the model or the system".

Here, *assumeFN* is the assumption function that describes how the assumption is used by the model or system, *referent* is the model or system component that the assumption is about, *scope* is a description of which parts of the system the assumption refers to, and *proposition* is a statement about the *referent*'s existence, relations, or quality.

1.1.2 Classification of Assumptions

This section focuses on the classification of assumptions by researchers from different viewpoints. Garlan et al. [6] focused on architectural assumptions. They discuss four main assumptions categories and some subcategories related to components and connectors that can result in architectural mismatch. The categories are:

- Nature of component: This category describes three subcategories infrastructure, control model, and data model. Infrastructure assumptions are about the substrate on which a component is built. Control model assumptions are about which components will control the sequencing of computation, and data model assumptions are about the way data, managed by a component, would be manipulated by the environment.

- Nature of connectors: The two subcategories of this category are protocols and data model. Protocol is concerned with assumptions about the characteristics of a connector's patterns and interactions, and data model is concerned with the kind of data being communicated.

- Global architecture structure: This category captures assumptions about the topology of the system communication. It also covers existence, i.e., presence- or absence-related assumptions of particular components and connectors.

- Construction process: This category includes assumptions related to the order in which building blocks are instantiated.

Dewar et al. [4] define assumptions in the context of assumption-based planning. In a later work, Dewar [7] has classified assumptions as shown in Fig. 1.1. The classification is broad in the sense that it covers a wide variety of assumptions including facts, constraints, design decisions, and design rationales. It should be mentioned that assumptions often overlap with the concepts of requirements, constraints, and design rationales. A brief description on how assumptions are related to these concepts can be found in [8].

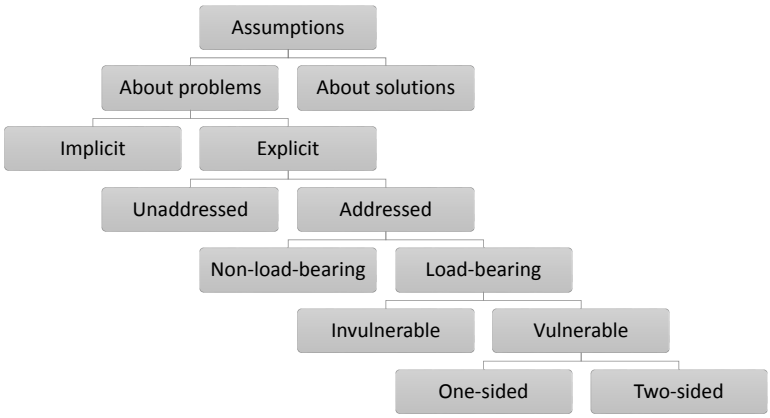

Figure 1.1: Classification of assumptions by Dewar [7]

Assumptions are implicit when they are not documented or communicated in the development process, and explicit otherwise. Assumptions can be implicit in at least two ways. First, when people are aware of the assumptions but do not document them because of lack of consciousness about the pitfalls of implicit assumptions. Second, there is no awareness of the implicit assumptions among the stakeholders [9].

Lewis et al. [10] have presented a classification of assumptions from the viewpoint of software developers while they are coding. The assumptions types are control, environment, data, usage, and convention. Steingruebl and Peterson [11] support the classification of Lewis et al. [10] and suggest adding a detailed level, e.g., checklists for the major assumption types. They also mention security as an assumption type. A classification of three assumptions classes are presented by Lago and Vliet [12], motivated by the general information system literature. Their study focuses on the architectural assumptions: technical, organizational, and managerial. They also mention that an assumption is cross-cutting if it relates more than one structural element or component. Spiegel et al. [13] identify three major classes of constraints that are based on the types of object attributes in the constraints (invariant vs. dynamic) and on the object scope of the constraints (one vs. many) that are invariant, dynamic, and inter-object.

Tirumala [14] classifies assumptions based on three dimensions namely time-frame, criticality, and abstraction. Three assumption types static, system configurations, and dynamic are described under time-frame. The validity

of static assumptions remains the same during the lifetime of the software. However, validity can be changed as the system evolves. For the system configuration assumptions, validity does not change during a single execution of the system, though, validity can be changed between different executions. Dynamic assumptions are those whose validity might be changed during the system's execution.

King and Turnitsa [15] mentioned some possible assumptions classes, which are intension vs. extension, primary vs. derivative, joint vs. disjoint with others, exogenous vs. endogenous, deterministic vs. probabilistic, and controllable vs. non-controllable.

This thesis classifies assumptions according to their validity states, e.g., *unchecked* vs. *checked*, *conflicting*, *mismatched*, and *invalid* vs. *valid*. An assumption can be in different states of validity at different times of a project's life cycle. In addition, an assumption can be in different validity states at the same time based on whether it is checked against artifacts within its scope or beyond its scope e.g., other assumptions. The history of validity states can be used to understand the evolution of assumptions. Such history can be useful for failure investigations due to mismatched assumptions for example.

- *Unchecked* vs. *Checked assumptions*: An assumption is unchecked when it has not been verified for its validity, and checked otherwise.

- *Conflicting assumptions*: An assumption is conflicting if it contradicts or conflicts with one or more other assumptions or artifacts beyond the scope of the assumption.

- *Mismatched assumptions*: An assumption is mismatched if we cannot determine whether the associated artifacts of the assumption will fulfill the statement or fact or not. In other words, there is no evidence provided by the artifacts that can be matched against what is assumed.

- *Invalid* vs. *Valid assumptions*: An assumption is considered invalid if the statement or fact of the assumption is false or incorrect; it is valid otherwise, i.e., the stated statement holds. The validity/invalidity of an assumption can most often be determined by verifying its statement or fact without necessarily looking at any other assumptions or artifacts other than what the assumption itself is related to. On the other hand, conflicting assumptions are determined from the aggregation of more than one assumption.

1.1.3 Technical Debt

Invalid assumptions may cause sub-optimal design decisions, of a complex software system, that need to be corrected later. Also, the loss of implicit assumptions over time has a negative impact on long-lived software systems. These circumstances are connected to the concept technical debt.

Technical debt is a metaphor that uses concepts from financial debt to describe the trend of increasing software development costs over time due to suboptimal decisions taken at various phases of software development, e.g., architecture and design, implementation usually to ensure speedy release.

An unreasonably high amount of technical debt may significantly hurt software evolution and quality.

In software engineering, the metaphor technical debt was first introduced two decades ago by Ward Cunningham who described it as "Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite... The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt." [16].

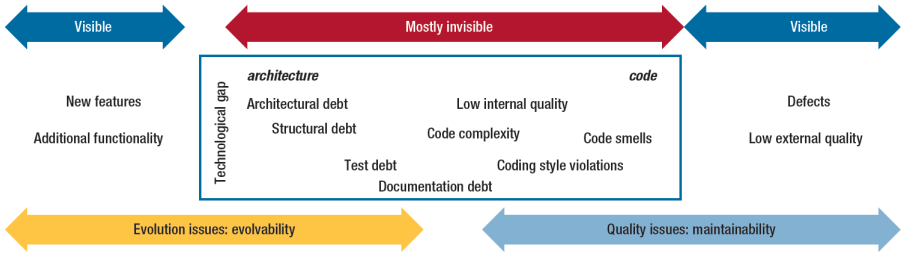

Figure 1.2: Landscape of technical debt by Kruchten et al. [17]

Since Cunningham introduced this metaphor, people have related technical debt with many different kinds of problems of software development in a wide range of spectrum ranging from deploying, selling, evolving to anything that has an ill effect on software development. Associating technical debt with too many concerns has somewhat diluted the concept of this metaphor and consequently this metaphor is losing some of its strengths [17]. As a response to this development, Kruchten et al. proposed a landscape of technical debt toward a better definition and theoretical foundation of the term technical debt in software engineering. This landscape, as shown in Fig. 1.2, arranges different elements in the ranges of *visible* and *invisible* and proposes to limit debt to the invisible elements within the rectangular box.

1.1.4 Classification of Technical Debt

McConnell classified technical debt from the viewpoint (i.e., whether the debt is intentional or not) of the way technical debts are introduced [18] as shown in Fig. 1.3. The unintentional technical debt refers to debt that is incurred unknowingly. Examples of such debts are an inexperienced programmer's code that is of low quality and does not meet the industry standards, and acquiring a company that had accumulated significant amount of technical debt where the debt is identified after the acquisition. On the other hand, intentional technical debt is a result of a deliberate decision made by the organization to optimize for the present rather than the future, e.g., such as writing unit tests for some code after the release because it was not possible to do it earlier due to time pressure.

Intentional debt can be both short and long-term, focusing on the strategy of incurring and paying-off debt. Short-term debt is taken on in a reactive manner "usually as a late-stage measure to get a specific release out of the door" [18]. Short-term debt is a form of a quick-fix that is expected to be paid

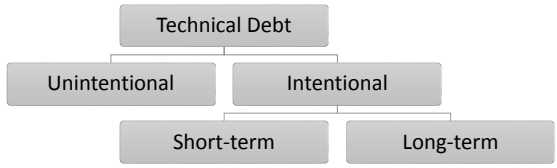

Figure 1.3: Classification of technical debt by McConnell [18]

off frequently such as "We don't have time to implement this the right way; just hack it in and we'll fix it after we ship" [18]. Long-term debt is a result of proactive strategic decisions by an organization. Being related to strategic decisions, long-term debt is sizable and is paid off in the long term such as "We don't think we're going to need to support a second platform for at least five years, so this release can be built on the assumption that we're supporting only one platform" [18].

The classification of Fowler [19], as shown in Fig. 1.4 as a quadrant, uses two dimensions: deliberate/inadvertent and reckless/prudent. The deliberate/inadvertent is similar to what McConnell has described as intentional/unintentional in his classification discussed earlier in this section.

	Reckless	**Prudent**
Deliberate	"We don't have time for design"	"We must ship now and deal with consequences"
Inadvertent	"What's Layering?"	"Now we know how we should have done it"

Figure 1.4: Classification of technical debt by Fowler [19]

Considering the other dimensions, i.e., reckless/prudent, the debt classes are:

- Deliberate and Reckless debt: The debt is taken on intentionally but the organization is unable to estimate the realistic accumulation of interest over time. For example, if the debt is about taking a "quick and dirty" way instead of a good design principle where the organization knows the design principle, and even is capable of practicing it but they do not have enough time to implement it. This debt is reckless if the organization is unable to measure the accumulation of interest over time.
- Deliberate and Prudent debt: For example, from the preceding dimension

if the organization is able to estimate the consequence of taking on the debt, it is deliberate and prudent.

- Inadvertent and Reckless debt: The organization is neither aware that it is taking a sub-optimal decision, nor it is able to estimate the consequence of taking the sub-optimal decision.
- Inadvertent and Prudent debt: At the time when the debt was introduced, the organization was unaware of it, but as they gained more experience over time, they learn what they should have done differently and realize that they created an inadvertent debt.

Tom et al. reported in a literature review [20] different types of debt which are code debt, design and documentation debt, environmental debt, knowledge distribution and documentation debt, and testing debt. The idea of the knowledge distribution and documentation debt is that technical debt increases as the organizational knowledge gets lost. They also reported different attributes of technical debt from the literature which are monetary cost, amnesty, bankruptcy, interest and principal, leverage, and repayment and withdrawal. Li et al. reported additional classes of technical debt in a systematic mapping study [21] which are requirements debt, build debt, infrastructure debt, versioning debt, and defect debt.

Both assumptions and technical debt are relatively new concepts in software engineering. They have been used in different contexts, which is reflected in this section. Researchers in the area of technical debt are trying to confine the metaphor related to issues that will mostly benefit software development because they believe relating too many issues with technical debt will make the concept very generalized and will weaken the strength of this metaphor. Similarly, the concept of assumption also needs to be focused for a better definition.

1.2 Motivation and Problem Domain

Today's Cyber-Physical Systems (CPSs) intrinsically combine many domains and areas of expertise to achieve system goals [22]. This demands significant interaction among people, environment, software, and hardware artifacts from different domains, which is an indication of a highly complex system. The multi-disciplinary nature of such complex systems deals with various concerns that are required to be addressed. Based on research in psychological science, the maximum number of concerns that the human brain can consciously process at the same time is limited which has a negative effect on decision making process that deals with multiple concerns[1].

Assumptions are an integral part of software systems. It is thus challenging for software and system developers building CPSs to consider all relevant and significant assumptions among various components to make good decisions.

Assumptions are existent in almost all aspects of the software development, from human factors to core software development activities like requirements

[1]The limited capacity of human brain was first described in 1956 by George Miller as 7±2 in article [23]. Newer research suggests that the human brain can simultaneously process as low as 4 items [24].

engineering, designing, implementation, verification, and validation. Thus, they have influence on quality attributes like security, safety, and dependability.

Attempts have been made to capture assumptions formally in machine-checkable format [14, 25] and semi-formally [10, 26, 27] which are not amenable to automated checking. However, the formal approaches suffer from the lack of scope of assumption categories, e.g., global architecture style or do not address cross-cutting assumptions.

Farenhorst and Boer [28] have in a systematic literature review on knowledge management in software architecture identified assumptions as a type of tacit knowledge. The knowledge engineering discipline considers tacit knowledge as volatile and challenging to preserve and transfer [29]. Among different assumption classes, implicit assumptions are identified as the root cause of widespread software reuse problems [6]. On the other hand, if assumptions are not encoded in a way that they can be checked automatically as the system evolves, validation of such assumptions would less likely be practical due to higher cost of human labor.

To understand the area of assumptions, its current state, and the existing challenges of assumptions, we have conducted a literature review which is reported in paper I. The review suggests that a machine-checkable holistic assumption management approach has the potential to minimize many challenges related to assumptions.

In paper II, we worked on a proof-of-concept on formalizing architectural assumptions from a real case named Aesop, reported by Garlan et al. [6]. The reason we want to capture assumptions formally is that it is amenable to automated checking. The Aesop system was planned to build a design environment generator tool supporting architectural design and analysis from an extensive amount of COTS components so that software development cost and time can be minimized. The estimated development time of the Aesop project was six months and one person-year. However, in reality, only a prototype of the system was built in about two-and-a-half person-years, which was realized in a prototypical and low-performance way resulting in a product difficult to maintain. Mismatched architectural assumptions were identified as the primary source of problems for the failure of the project.

The Aesop project has considerable size, complexity, and extensive amount of COTS components with a good description of the implicit assumptions that caused project failure. Thus, as a proof-of-concept, we wanted to investigate whether it is possible to formally capture and automatically check real world cross-cutting assumptions from such a project. It is important to be able to formally capture cross-cutting assumptions because pair-wise composition of an assumption with its guarantee is considered too simplistic [14]. The real-world assumptions reported in the Aesop project are either not possible or not feasible to capture with a pair-wise composition technique. For example, the assumption "Component X assumes that all components having a property P are completely independent of each other i.e., there is no connection between them" is not possible to model using a pair-wise approach. A pair-wise approach can create inconsistencies while maintaining assumptions that are cross-cutting in nature. For example, the assumption "Component X expects that all other components have component Y as a library" can be modeled as a single cross-cutting assumption. However, a pair-wise approach would require writing

one assumptions for each of the other components. Hence, modifying such assumptions might cause inconsistencies in the assumption model.

As we tried to capture the structural cross-cutting architectural assumptions of the Aesop system, we were able to capture and check them automatically to find mismatches using the Alloy language and toolkit. It should be noted that few process-related assumptions are omitted because they do not fit in the category of structural assumptions, and it is not possible to model some of the assumptions using Alloy, because a single language cannot capture all types of assumptions.

The wide variety of assumption classes discussed in section 1.1.2 tells us that there are many different types of assumptions. From our experience of formalizing architectural assumptions in paper II, formally modeling different assumption classes requires several formal languages and tools. Apart from the resources and time for building such a tool, a holistic approach based on several formal languages and tools has challenges like availability of people who are familiar with the formal languages and tools, learning such languages and tools, and adjusting the development process to incorporate them.

While the practicability of a heavy-weight holistic approach on assumption management needs to be researched, which to the best of our knowledge is not currently reported in the existing literature, we focused on working on a light-weight approach. The essence of the light-weight approach is that it should not introduce additional tooling or languages for managing assumptions but should leverage tools or languages already available in the project to facilitate quick adoption. For example, if a project uses Alloy for specifying an architecture, then using the same tool to specify assumptions as well would be a light-weight approach. However, there are limitations regarding a light-weight approach, for example, storing additional information like the validity state.

To explore the applicability of an MDE-based approach, we used self-driving miniature cars as an example. We developed a meta-model for capturing the sensor layout of a self-driving miniature vehicle; this is described in paper IV. In the meta-model, we focused on similar structural assumptions as modeled in paper II with the formal language Alloy, and found some related assumptions which are shown in table 1.1.

Since the identified assumptions from the self-driving miniature vehicle project are static in nature, we investigated the applicability of OCL (Object Constraint Language) [30] as an existing standard to capture and validate these assumptions. Using OCL, we successfully modeled and validated the violations of the identified assumptions. Since OCL is available as a default mechanism under the ECORE modeling workbench [31], we have not introduced any additional tool or technique for capturing and validating the identified assumptions, which is the reason why we call this a light-weight assumption management approach. ECORE is a meta-model and it is a part of the core eclipse modeling framework (EMF).

The possibility of a light-weight assumption management relies on the availability of existing tools in a project that are capable of capturing and validating assumptions. In general, we can expect that the more syntactically and semantically rich an architecture is then the more applicable the light-weight assumption management approach is. Because it is expected that a formally rigorous architecture is built using tools or languages that can also

Table 1.1: Relating architectural assumption categories from paper II to assumptions in paper IV

Architectural assumption category (Garlan et al. [6], paper II)	Related assumptions from the self-driving miniature vehicle project (paper IV)
Nature of component (control model), Nature of connector (protocol, data model)	The *connectionType* property of the sensor "SRF08" must use I^2C bus as ConnType.
Global architecture style	*SensorClass* "SRF08" can only be associated with an *Ultrasonic* sensor.
	The maximum number of *Ultrasonic* sensors is 48 in case of using STM32F4 as the target hardware environment because three I^2C buses are available each hosting up to 16 devices.

be used to capture assumptions. Since MDE approaches support architecture models with rich syntax and semantics, such approaches can easily leverage the benefits of capturing assumptions formally and checking them automatically using existing tools.

In this thesis, we also focus how assumptions influence technical debt. Tom et al. reported *knowledge distribution and documentation debt* as a type of technical debt [20]. The landscape of technical debt illustrated by Kruchten et al., as shown in Fig. 1.2, shows *documentation debt* as a technical debt category that is mostly invisible, which on the other hand indicates the tacitness of the artifacts knowledge and documentation.

Implicit assumptions are correlated with technical debt. When implicit assumptions are not documented, they get lost over time. This happens due to that developers forget about the assumptions they made in the past, or that assumptions are not available at present because people have left the organization. When the rate of the loss of assumptions is low, the technical debt associated with the assumptions increases gradually and when a developer with implicit knowledge leaves the organization, the technical debt increases significantly. Thus, the loss of implicit assumption or not verifying assumptions, as the system evolves, accumulates technical debt.

Fig. 1.5 shows how MDE is acting as a bridge between assumptions and technical debt in general. It also shows the foci of the papers included in this thesis. In Fig. 1.5, we have assumptions and technical debt in two filled (blue) rounded rectangles of the figure, and they are further classified according to different dimensions and types. The filled (green) rounded rectangles in the type columns indicate types that this thesis has dealt with. All the rounded rectangles (green) in the center task column indicate major tasks or concepts realized in this thesis. The circle (blue) labeled "model-driven engineering", in the task column relates the major tasks that are connected to MDE. The small black circles indicate associated papers. The links connecting black circles and rounded rectangles (green) mean that the indicated paper is related to the

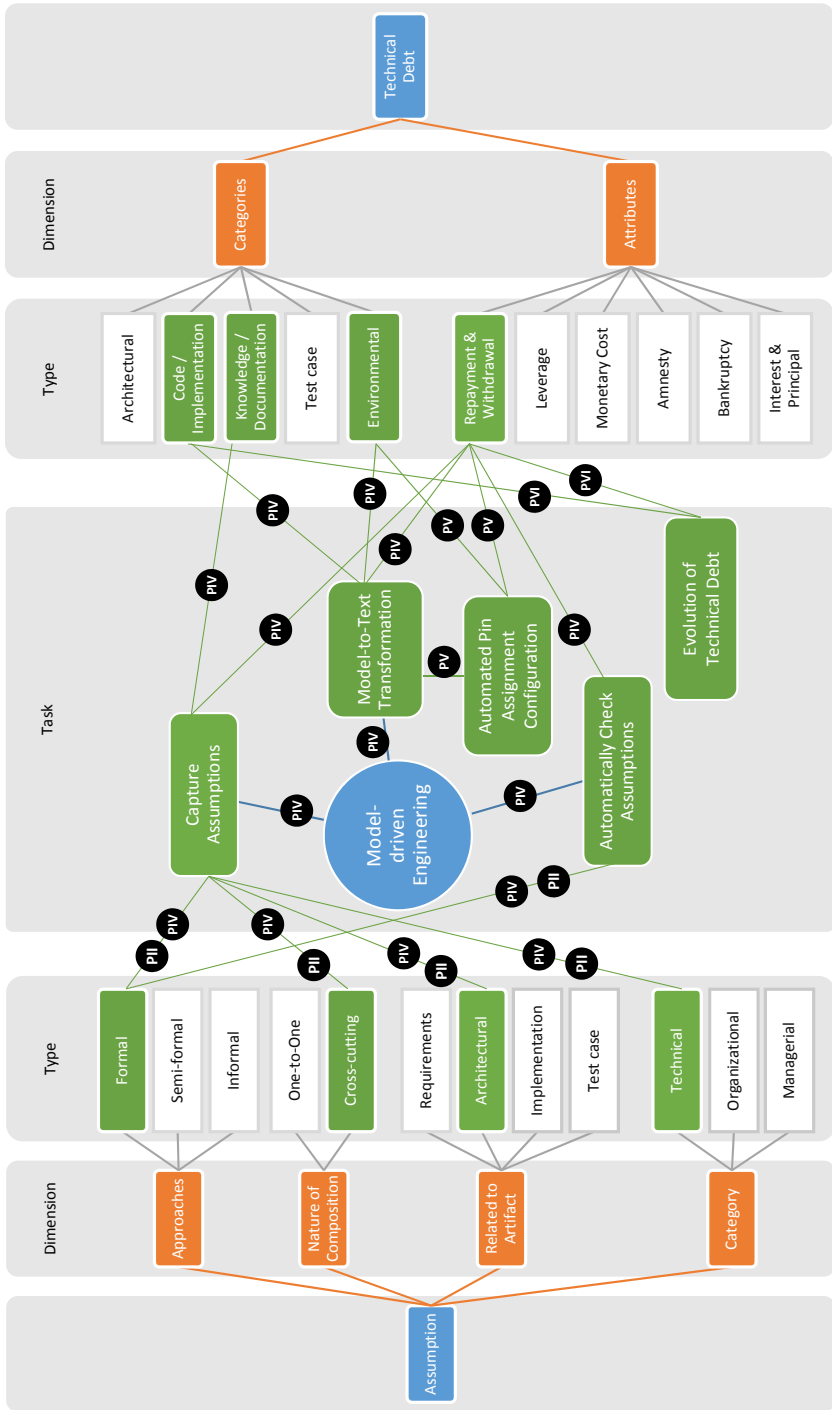

Figure 1.5: The figure shows how MDE is mediating between assumptions and technical debt.

associated type or task.

Manual tasks that are repetitive over time and that have the possibility of being automated are a form of technical debt in the *environmental debt* category that accrues interest over time whenever a manual task is repeated [20]. On the other hand, source code that is subject to refactoring form technical debt in the category "code debt". Automated code generation for the simulation and the vehicle also falls under this.

1.3 Research Questions

Starting with the identification of various challenges of assumptions in software engineering, this thesis takes a step to formally capture and automatically check cross-cutting architectural assumptions extracted from a project that was primarily built from COTS components. To apply the idea of a light-weight assumption management approach, we selected a miniature self-driving car project, and successfully captured and checked assumptions in the architecture of the sensor management part of the car project through an MDE approach. MDE facilitates generating code for the configuration of a 3D-simulation environment. It also supports the automatic generation of sensor code. Such automation reduces technical debt. Toward a more practical solution and to further reduce technical debt, we implemented solutions to find the best pin assignment configuration which are both faster and support higher number of pins than our previous work. Finally, we studied the evolution of technical debt in the development of two miniature self-driving cars. In particular, we explore the contributing factors for the accumulation of technical debt.

The research questions are:

- **RQ1** What are the challenges of assumptions in software engineering?
- **RQ2** How can an MDE approach leverage light-weight assumption management and reduce technical debt during the development of a self-driving miniature car?
    1. How can the sensor management for a self-driving miniature vehicle be supported by using MDE?
    2. How can a light-weight approach toward assumptions be realized with MDE?
    3. How can an MDE approach contribute to reduce technical debt?
- **RQ3** How does the use of legacy/3^{rd}-party/open source software components affect the evolution of the technical debt on the example of the development of a self-driving miniature car?

1.4 Overview of Papers

Paper I: Review and Challenges of Assumptions in Software Development

Paper I reviews existing work in assumption management and finds assumption challenges that should be mitigated in order to build better systems. The results

show that assumptions are concerned with many different areas of software engineering, e.g., requirements engineering, software architecture, software security, and knowledge management. This paper also identifies various concerns where assumption management can make positive contributions, e.g., continuous deployment through virtual integration, composability of COTS (commercial-off-the-shelf) components, evidence-based software engineering, safety and security, and verification and validation.

It is clear that there is a lack of integrated approaches toward systematic assumption management, enabling quantitative analysis and checks of assumptions, which would ultimately mitigate the key challenges associated with the assumptions. Mitigation of the challenges would support virtual integration of components, continuous deployment, and more loosely coupled CPS development.

A holistic assumption management framework can offer different services such as accessing the assumptions and their properties, on-request assumptions validation, on-request assumption updates, report errors, warnings to other tools or frameworks that want to utilize these services.

Paper II: Towards Formalizing Assumptions on Architectural Level: A Proof-of-Concept

Motivated by the idea that a holistic assumption management system based on formal methods would mitigate challenges related to assumptions (from paper I), we have conducted an initial study to assess the possibility of capturing real-world assumptions amenable to be fully checked automatically. This study focuses on the cross-cutting architectural assumptions and modeling them using Alloy [32], which is a checker based on SAT solvers. The assumptions are selected from a study by Garlan et al. [6] reporting assumptions identified from a project that was aimed to build a tool for generating design environment supporting architectural design and analysis from an extensive amount of COTS components. When tested, our developed models successfully explored all of the architectural mismatches related to the selected assumptions. In general, the proposed approach supports capturing assumptions about the architectural structure including relations and connections among different artifacts. More specifically, the proposed approach can capture both general/high-level assumptions and application-specific assumptions. General/high-level assumptions assumptions are reusable because they are less dependant on specific instances of architectural artifacts. On the other hand, application-specific assumptions are more tightly coupled with certain artifacts of an instance of an architecture.

Paper III: COTS-Architecture with a Real-Time OS for a Self-Driving Miniature Vehicle

Based on our experience from paper II, which formally modeled assumptions from a project that extensively used COTS components, we looked for a project that extensively has used COTS components and selected a self-driving miniature vehicle development project. This paper introduces an experimental environment of a self-driving miniature vehicle on which we performed our later research of relating MDE with assumption and technical debt.

Paper III presents the hardware and software architecture of a self-driving miniature vehicle, which is based entirely on COTS components. The choice of COTS hardware and software components has strategic values for the long-term evolution of self-driving miniature vehicles. These vehicles deal with a lot of factors concerning the vehicles and their surrounding environment. The high variability of such factors requires a thorough selection of the COTS components for a good product line architecture for self-driving vehicles.

In order to get immediate feedback on the partially developed software before the vehicle hardware was ready to run, the simulation environment "OpenDaVINCI" [33] was used from the beginning of the software development. The seamless transferability of the vehicle software from the simulation to the hardware and early feedback from the simulation environment helped in speeding up the project development. Our experience from this project indicates that such an approach can significantly reduce the development time and, thus, enable the development of an experimental self-driving miniature vehicle in only three months.

Paper IV: MDE-based Sensor Management and Verification for a Self-Driving Miniature Vehicle

During the realization of the design for the self-driving miniature vehicle in paper III, it was apparent that maintaining, debugging, and testing of the software in the real-time operating system ChibiOS/RT [34] to interface with the sensors and actuators is error-prone, time-consuming, and tedious. Since sensor types and their layout can vary with changes in the surrounding environment of the vehicle and also with the increased functionalities of the vehicle, it is worthwhile to adopt a model-based approach to maintain the embedded software for the STM32F4 Discovery Board running on ChibiOS/RT.

In this paper, we present an MDE-based approach for managing different sensor setups in a CPS development environment to leverage automated model verification, support system testing, and enable code generation. The MDE-based sensor management approach relies on a DSL to describe the domain model of possible sensor layouts. We identified several architectural assumptions and captured them using OCL to make them automatically checkable while verifying an instance of the DSL model. Thus, assumptions are captured and checked using languages or tools which are readily available within the capacity of MDE. Throughout capturing and checking of assumptions, technical debt in the category *knowledge* is reduced. To validate the system in a 3D-simulation environment, we engineered a model-to-text (M2T) transformation. The M2T considers an instance model of a sensor layout as input and then generate a configuration file for the 3D-simulation environment containing various parameters and setup information regarding the sensors. Being a manual task, automating the generation of the configuration file for the 3D-simulation environment reduces technical debt in the category *environmental*.

When the validation is completed, a considered sensor configuration is transformed into a constraint satisfaction model to be solved by the logical programming language Prolog. Based on this transformation, the conformance to the embedded system specification is formally verified and possible pin assignments for how to connect the required sensors are calculated. The

approach was validated during the development of a self-driving miniature vehicle using an STM32F4-based embedded system running the real-time operating system ChibiOS/RT as the hardware/software interface to the sensors and actuators. Performing this task manually is also time consuming, thus, automating it reduces technical debt in the category *environmental*.

Paper V: Engineering the Hardware/Software Interface for Robotic Platforms - A Comparison of Applied Model Checking with Prolog and Alloy

This paper further extends the problem of finding a feasible pin assignment configuration outlined in paper IV. Specifically, this paper focuses on the problem of finding *a feasible*, *all possible*, or *the best* pin assignment configuration for a hardware/software interface board. Furthermore, the length of the desired configuration is also increased compared to paper IV. Researchers and developers dealing with embedded systems for robotic platforms have to address this task to define how a set of sensors (e.g. ultra-sonic, infrared range finders) and actuators (e.g. steering, acceleration motors) need to be connected in the most efficient way. We performed a formal experiment with the declarative languages Prolog and Alloy to find the desired configuration of a length up to ten pins with a configuration space greater than 14.5 million possibilities. Apart from reducing technical debt through automatically finding a possible pin assignment configuration (in paper IV), finding the best pin assignment configuration is strategically important to reduce refactoring of the sensor code as the system evolves in the future, i.e., adopting additional sensors. Thus, selecting a possible pin configuration incurs an intentional short-term technical debt. On the other hand, selecting the best pin configuration either avoids or repays such a debt.

We have modeled the domain of possible pin configurations for such boards and analyzed its complexity. On the example of the hardware/software interface STM32F4 Discovery Board, which we are using on our self-driving miniature vehicles, we have modeled the interface board's pin configuration possibilities into a graph-based representation. To verify a desired configuration to be matched with a possible pin assignment, we traversed the graph and created an equivalent target model for the declarative languages Prolog and Alloy, respectively. Using our example resulted in about 14.5 million configuration possibilities, we ran an experiment for the aforementioned three use cases.

We show that the number of possible configurations increases when either the number of pins or the number of functions per pin are increased. However, increasing the former lets the size of the problem space grow significantly faster than increasing the latter. Furthermore, adding more physical pins is also a costly factor; thus, researchers and engineers continuously have to deal with the problem of finding a feasible, all possible, or the best pin assignment configuration for their specific robotic platform. Being a repetitive and time-consuming task, the automation of finding the pin assignment configuration reduces technical debt.

Paper VI: Explicating, Understanding and Managing Technical Debt from Self-Driving Miniature Car Projects

Integrating a light-weight assumption management approach with the existing software process is a proactive way of avoiding effects of invalid assumptions such as technical debt in the long run. This paper is a first attempt to investigate the amount of technical debt accumulated in the already built self-driving miniature cars. The objective is to understand how technical debt evolved throughout the development period and plan necessary actions for the future development of such cars to reduce technical debt. We performed a case study on two versions of a core feature to understand the state of technical debt in the feature implementation. Based on the case study results, a structured interview was performed to see whether the feature developers understand specific issues accumulating technical debt in their own source code. The structured interview also looked for the developers' thoughts on the possible organizational, process-, and domain-related factors that contributed to accumulate technical debt.

The results of this study show that lack of knowledge related to programming techniques is not the foremost driving factors for technical debt. The causes are rather implied in factors like time pressure, hardware/software integration, incomplete refactoring, and use of 3^{rd} party/open source/freely available code from the Internet.

1.5 Research Methodology

This section describes the research methodologies used in this thesis to answer our research questions. We have used content analysis, case study, design science, experiment, and structured interview research methods. Paper I uses content analysis research method to identify the state of the art of assumptions in different areas of software engineering and to gain insights regarding assumption challenges. Case study research method is used in paper II and IV to investigate the Aesop project and a self-driving miniature car project respectively. Design science is used to develop a meta-model and artifact related to its verification and code generation because it is a exploratory and improving type of research method. In Paper V, we have used an experiment because of its rigorous nature to compare two model-checking approaches. To understand developers' knowledge about code smells related to technical debt, structured interviews are performed, and are reported in paper VI.

Content Analysis

Content analysis, also called document analysis, is a research method used to find and classify information from documents such as records, reports, standards, etc. Content analysis allows discovering and describing focus groups of individual, group, institutional, or social attention [35].

Krippendorff [36] defines content analysis as:

> "A research technique for making replicable and valid inferences from data to their context."

Weber [35] defines content analysis as:

> "A research method that uses a set of procedures to make valid inferences from text. These inferences are about the sender(s) of the message, the message itself, or the audience of the message."

The definition of Krippendorff [36] is broader in the sense that content type is not restricted to the domain of textual analysis only but radio broadcasts, films, TV programs, etc. However, for content analysis the most obvious source of data is text.

The central idea of content analysis is to classify many words systematically into fewer content categories based on explicit rules of coding. It is also possible to examine trends and patterns in the target documents through content analysis. Content analysis is a powerful technique to determine authorship [37]. It also provides an empirical basis to monitor changes in public opinion [37] which makes it a popular research technique in the social and political science.

Computer-aided content analysis has been growing in popularity, and it is a common technique to use. Computer-aided content analysis supports analysing huge amounts of electronic documents automatically to sort out or group relevant data for further analysis by the researchers [35].

Paper I of this thesis uses content analysis in the form of document analysis to explore different types of assumptions, approaches to manage them, classify the existing research based on the approaches and understand the state-of-the-art of assumptions management in different areas of software engineering from where we can infer existing open challenges related to assumptions. This research method was used to answer RQ1.

Case Study

Case study is an empirical research method generally used for observation. It is used to monitor projects, activities, or assignments. Throughout the observation, data is collected on which statistical analysis can be performed. It is normally used to establish relationships between different attributes or tracking a specific attribute. Case study is suitable for understanding a particular system within its context, however it suffers from low power of generalizability [38].

Scientific disciplines such as sociology, medicine, and psychology use case study as a standard method for empirical studies [38]. Since a case study examines contemporary phenomena in its natural context, which is hard to study in isolation, case study is suitable for software engineering research [39].

Researchers have different understanding of what constitutes a case study. Positivist, critical, and interpretive are three different types of case study, defined by Klein and Myers, based on the research perspective [40].

- A positivist case study is close to the natural science research model [41]. It looks for evidences for formal propositions, measures variables, tests hypotheses, and makes inferences from a sample to a stated population. Explanatory type software engineering case studies tend to lean towards a positivist perspective.

- A critical case study targets the society with a specific focus on the social justice, equity, enforcement of law etc., with the aim of being critical and

emancipatory. Critical case studies identify different forms of legal and political restrictions that can hinder human abilities by controlling them.

- An interpretive case study aims to understand phenomena where the participants' interpretation of their context is used to understand phenomena.

Case study research method is used in Paper II toward gaining a better understanding of applying formal methods for real-world cross-cutting structural architectural assumptions with the aim of capturing and checking them automatically. Paper VI has also used case study for investigating the evolution of technical debt in the development of a core feature of self-driving miniature cars.

Design Science

Much of the research in the information system discipline can be characterized through behavioral science and design science paradigms [42]. While behavioral science paradigm is used to develop and verify theories explaining or predicting human or organizational behavior, Hevner et al. describe design science paradigm as:

> "The design science paradigm seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts" [42].

In the early 1990s, the importance of design science research was recognized by the information system community to improve the usage and effectiveness of the IT artifact by solving real-world business challenges. Researchers in information systems adapted the design research traditions of other fields and refined design research according to the unique characteristics of the information system domain [42].

Inherently, design science is a problem-solving process. Hevner et al. prescribed a guideline for conducting and evaluating design science research based on the principal "knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact" [42].

In this thesis, design science is used to develop a meta-model for sensor management of a self-driving miniature vehicle. The meta-model works as a core of a model-driven engineering approach toward the validation of sensor models and automated code generation for the simulation environment, and the actual execution platform in which the vehicle functionalities are deployed.

Experiment

An experiment is a formal, rigorous, and controlled investigation that is usually performed in a laboratory settings [38]. At the planning phase of an experiment, independent and dependent variables are identified. During an experiment, manipulating one or more independent variables while keeping all the other independent variables at a fixed level, the effect of the manipulation on the dependent variable is measured. Statistical analysis is then performed on the

collected data for revealing relationships between the variables. Experiments are purely quantitative because they involve measuring variables, manipulating them, remeasuring them, and recording quantitative data on which statistical analysis is performed [38].

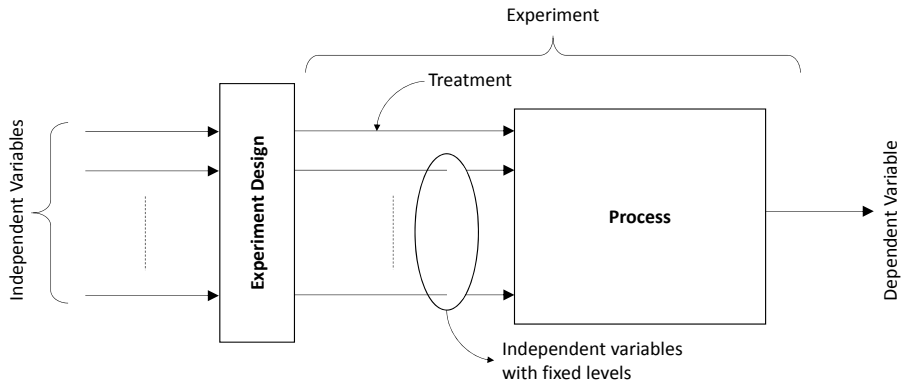

Figure 1.6: Illustration of an experiment [38]

Paper V uses an experiment to compare two model-checking approaches, using Alloy and Prolog, to find *a feasible*, *all possible*, and *the best* pin assignment configuration for a hardware/software interface board.

Interview

Interviewing is a research method that can be performed in different forms for descriptive, explanatory, and experimental research design. Interviews are mainly of three types: structured (formal), semi-structured (semi-formal), and unstructured (informal). In a structured interview, all questions are completely predefined; in a semi-structured interview some of the questions are predefined and then these questions can lead to further discussion based on the response of the interviewees; an unstructured interview is completely open for discussion. Interviews can be face-to-face, telephone-, mail-, email-, and web-based, and also be a combination of these.

We have used a structured interview in the form of web-based questionnaires for the research presented in Paper VI. We used two different questionnaires where the first one is used to check the knowledge of developers related to code smells accumulating technical debt in their own source code, and to know the developers' view on organizational, process-related issues that contributed to the accumulation of technical debt. The second questionnaire was used to check the level of agreement among the developers with respect to the criticality of the code smells ranked by a commercial tool, which was used to measure technical debt.

Validity Evaluation

We discuss threats to validity to the results of this thesis according to the definition reported by Runeson and Höst [43]:

- *Construct validity.* The review in paper I was performed with a specific focus on finding challenges of assumptions in software engineering. The rest of the studies were performed focusing on assumptions, MDE, and technical debt. The included papers were incrementally planned and put into the overarching picture delivered in this thesis.

- *Internal validity.* The observation that a MDE-based approach reduces *knowledge debt* through capturing structural architectural assumptions is valid. However, the *knowledge debt* within an architecture reduces with the capturing and checking of other knowledge artifacts, e.g., design decision, design rationale, and constraints as well. Section 1.2 discusses how these knowledge artifacts are closely connected to the concept of assumptions, thus, reducing the threat of our claim. In paper II we worked with assumptions reported from a real project. In paper IV we identified assumptions in the sensor management part of a self-driving vehicle development. The nature of the studied project in paper IV, V, and VI are similar in nature. In paper VI we have studied two projects both addressing the same competition with the same rules and conditions, and the settings for both were the same. Therefore, the evolution of technical debt could be studied for our specific setting.

- *External validity.* Since MDE-based tools are generally founded on rich syntax and semantics, they are quite likely to be able to capture and check assumptions, and automate tasks including code generation. Thus, in our view, the idea of MDE being a facilitator of a light-weight assumption management approach, and a facilitator of project automation is general and, thus, threat to the external validity is low. Rigorous approaches toward assumptions are not common in practice possibly due to their demand of procuring new tools, training people, and adjusting software process to incorporate the new tools. The idea of light-weight assumption management is practicable and such an approach is able to overcome the aforementioned obstacles.

- *Reliability.* The possibility of employing a light-weight approach toward managing assumptions depends on the availability of proper tools or languages in the project environment. The more rigorous an engineering approach is, the higher the chance of employing a light-weight approach is, because of the higher possibility of available tools or languages within the capacity of the engineering approach. Rigorous engineering approaches like MDE have the capability to reduce technical debt by capturing, automatically checking assumptions, and through automation at different levels. Our observation of the accumulation of technical debt sourcing from legacy/3^{rd}-party/open source code is valid, because it was the finding from the source code review even though the developers did not rank this as the most important source. We have reported possible reasons why holistic or heavy-weight approaches toward managing assumption are not common in practice. However, to give a definite answer, more research is required.

1.6 Conclusion and Outlook

This thesis presents a light-weight approach toward assumption management and shows how engineering approaches, based on sound syntax and semantics leverages capturing structural architectural assumptions, check them automatically for verification, and reduce technical debt.

The practitioners are aware of the problems sourcing from the assumptions, yet, there is a lack of rigorous approaches toward assumptions. A practical step toward managing assumptions is using existing language or tool capabilities that can be used for capturing and checking assumptions. The availability of tools and the formal basis of MDE not only makes it a good candidate for realizing such a light-weight assumption management approach but also reduces technical debt throughout automation.

Contributions

The contributions of this thesis are presented below.

- **C1:** Formalized real-world cross-cutting structural architectural assumptions and automatically checked for verification. **(RQ2)**

  We have captured and automatically-checked cross-cutting structural architectural assumptions, extracted from a project that was primarily built from COTS components, using the Alloy language and toolkit. In our approach, the composition of an assumption with its guarantee is not restricted between two components. Thus, we are able to relate more than one artifact into a single assumption, which resembles the reality as seen in the investigated real project and which is more expressible in composition compared to a pair-wise composition of assumptions.

- **C2:** Realized a light-weight assumption management concept through MDE. **(RQ2)**

  A light-weight assumption management approach is presented in this thesis. The idea of this approach is to use existing capabilities, e.g., tools and techniques in the project environment without introducing new tools and techniques to capture assumptions formally. A light-weight approach toward assumptions is easier to adopt and implement unlike a dedicated assumption management approach which is burdened with the cost of procuring new tools, training people using such tools, maintaining the tools, etc. In the example of a miniature self-driving vehicle project, we showed how existing tools within the development environment can be used to formally capture and automatically check assumptions rather than using dedicated tools and techniques for capturing and checking assumptions. The realization of the light-weight assumption management approach is comprehended through the use of an MDE-based approach. Details of the MDE-based approach is available in paper IV.

This thesis has discussed how managing assumption can reduce technical debt and how an MDE-based approach is capable of reducing technical debt. In addition, the preliminary investigation in paper VI shows that legacy/3^{rd}-party/open source code has influence in the development of technical debt.

Feature developers of the self-driving car ranked legacy/3^{rd}- party/open source code as less important contributing factor for the accumulation of technical debt. However, the review of the source code reveals that the mentioned sources significantly contributed to the accumulation of technical debt.

Future Directions

Our study with the evolution of technical debt in the development of a self-driving miniature vehicle sheds light on some interesting results. One of the observations of this study was that a big portion of the accumulated technical debt originated from the legacy/3^{rd}-party/open source/freely available code from the Internet. We also found that issues related to imported code remained almost the same from their introduction to the end of the project. We want to understand why the developers did not consider refactoring the debt from the imported code bases. From our experience with the full-scale vehicle system development, we know that some of the OEMs use many 3^{rd}-party components from contractors. Therefore, we are interested to know whether car manufacturers are also burdened with technical debt incurred by the 3^{rd}-party components or not.

We will continue working on the evolution of the technical debt in our miniature self-driving vehicle development projects. We will also investigate the state of the 3^{rd}-party components and compare it with the past projects data to see how they correlate. In particular, we want to investigate how technical debt in-flows together with 3^{rd}-party components affects the evolution of the overall technical debt of the product. We also want to investigate how a potential cost or impact model for calculating the technical debt differs incorporating 3^{rd}-party components could look like.

List of References

[1] "Forecasts | Driverless car market watch." [Online]. Available: http://web.archive.org/web/20150316211647/http://www.driverless-future.com/?page_id=384

[2] W. P. Rogers, "Report of the presidential commission on the space shuttle challenger accident," U.S. Government Accounting Office, Washington, D.C., Tech. Rep., 1986.

[3] I. Board, "ARIANE 5 flight 501 failure," European Space Agency, Paris, France, Technical Report, Jul. 1996. [Online]. Available: http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf

[4] J. A. Dewar, C. H. Builder, W. M. Hix, and M. H. Levin, "Assumption-Based planning; a planning tool for very uncertain times," Tech. Rep., 1993. [Online]. Available: http://stinet.dtic.mil/oai/oai?&verb=getRecord&metadataPrefix=html&identifier=ADA282517

[5] M. A. Hofmann, "Modeling assumptions: how they affect validation and interoperability," in *Proceedings of the 2005 European Simulation Interoperability Workshop (Euro SIW)*, Toulouse, France, 2005.

[6] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: Why reuse is so hard," *IEEE Software*, vol. 12, no. 6, pp. 17–26, 1995.

[7] J. A. Dewar, *Assumption-Based Planning: A Tool for Reducing Avoidable Surprises*. Cambridge University Press, Oct. 2002.

[8] M. A. Al-Mamum and J. Hansson, "Review and challenges of assumptions in software development," in *Proc. of the Second Analytic Virtual Integration of Cyber-Physical Systems Workshop (AVICPS)*, R. Mangharam and P. Feiler, Eds., 2011, pp. 53–60.

[9] R. Roeller, P. Lago, and H. van Vliet, "Recovering architectural assumptions," *Journal of Systems and Software*, vol. 79, no. 4, pp. 552–573, Apr. 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121205001615

[10] G. A. Lewis, T. Mahatham, and L. Wrage, "Assumptions management in software development," DTIC Document, Technical Report CMU/SEI-2004-TN-021, 2004. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA443152

[11] A. Steingruebl and G. Peterson, "Software assumptions lead to preventable errors," *Security Privacy, IEEE*, vol. 7, no. 4, pp. 84 –87, Aug. 2009.

[12] P. Lago and H. van Vliet, "Explicit assumptions enrich architectural models," in *Proceedings of the 27th international conference on Software engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, p. 206214. [Online]. Available: http://doi.acm.org/10.1145/1062455.1062503

[13] M. Spiegel, P. F. Reynolds, Jr., and D. C. Brogan, "A case study of model context for simulation composability and reusability," in *Proceedings of the 37th Conference on Winter Simulation*, ser. WSC '05. Winter Simulation Conference, 2005, pp. 437–444. [Online]. Available: http://dl.acm.org/citation.cfm?id=1162708.1162784

[14] A. S. Tirumala, "An assumptions management framework for systems software," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2006.

[15] R. D. King and C. D. Turnitsa, "The landscape of assumptions," in *Proceedings of the 2008 Spring simulation multiconference*, ser. SpringSim '08. San Diego, CA, USA: Society for Computer Simulation International, 2008, p. 8188. [Online]. Available: http://dl.acm.org/citation.cfm?id=1400549.1400560

[16] W. Cunningham, "The WyCash portfolio management system," in *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*, ser. OOPSLA '92. New York, NY, USA: ACM, 1992, p. 2930. [Online]. Available: http://doi.acm.org/10.1145/157709.157715

[17] P. Kruchten, R. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, vol. 29, no. 6, pp. 18–21, Nov. 2012.

[18] S. McConnell, "Technical debt-10x software development," Nov. 2007. [Online]. Available: http://www.construx.com/10x_Software_Development/Technical_Debt/

[19] M. Fowler, "TechnicalDebtQuadrant," Oct. 2009. [Online]. Available: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html

[20] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498–1516, Jun. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121213000022

[21] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, no. 0, pp. 193 – 220, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121214002854

[22] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, p. 731736. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837461

[23] G. A. Miller, "The magical number seven, plus or minus two: some limits on our capacity for processing information." *Psychological review*, vol. 63, no. 2, p. 81, 1956. [Online]. Available: http://psycnet.apa.org/journals/rev/63/2/81/

[24] N. Cowan, "The magical mystery four how is working memory capacity limited, and why?" *Current Directions in Psychological Science*, vol. 19, no. 1, pp. 51–57, Feb. 2010. [Online]. Available: http://cdp.sagepub.com/content/19/1/51

[25] S. Fickas and M. Feather, "Requirements monitoring in dynamic environments," in *Proceedings of the Second IEEE International Symposium on Requirements Engineering, 1995.*, Mar 1995, pp. 140–147.

[26] P. Lago and H. van Vliet, "Explicit assumptions enrich architectural models," in *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, G.-C. Roman, W. G. Griswold, and B. Nuseibeh, Eds. ACM, 2005, pp. 206–214.

[27] H. Ordibehesht, "Explicating critical assumptions in software architectures using AADL," Master's thesis, University of Gothenburg, Gothenburg, Sweden, 2010.

[28] R. Farenhorst and R. C. Boer, "Knowledge management in software architecture: State of the art," in *Software Architecture Knowledge Management*, M. Ali Babar, T. Dingsyr, P. Lago, and H. Vliet, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 21–38. [Online]. Available: http://www.springerlink.com/content/l206714955372378/

[29] D. Hislop, *Knowledge management in organizations: A critical introduction.* Oxford University Press, 2005.

[30] J. B. Warmer and A. G. Kleppe, *The Object Constraint Language: Precise Modeling With Uml (Addison-Wesley Object Technology Series).* Addison-Wesley Professional, Oct. 1998. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0201379406

[31] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework.* Pearson Education, Dec. 2008.

[32] D. Jackson, "Alloy: a lightweight object modelling notation," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 2, pp. 256–290, Apr. 2002. [Online]. Available: http://doi.acm.org/10.1145/505145.505149

[33] "OpenDaVINCI, webpage- http://www.christianberger.net/opendavinci," Jun. 2013. [Online]. Available: http://www.christianberger.net/opendavinci

[34] G. D. Sirio, "ChibiOS/RT," http://www.chibios.org/, Jun. 2013.

[35] R. P. Weber, *Basic Content Analysis*, 2nd ed., ser. Quantitative Applications in the Social Sciences. Sage Publications, 1990, no. 49. [Online]. Available: https://books.google.se/books?id=nLhZm7Lw2FwC

[36] K. Krippendorff, "Content analysis," in *International Encyclopedia of Communication*, E. Barnouw, G. Gerbner, W. Schramm, T. L. Worth, and L. Gross, Eds. New York: Oxford University Press, 1989, vol. 1, pp. 403–407. [Online]. Available: http://repository.upenn.edu/asc_papers/226

[37] S. Stemler, "An overview of content analysis," *Practical assessment, research & evaluation*, vol. 7, no. 17, pp. 137–146, 2001. [Online]. Available: http://pareonline.net/getvn.asp?v=7&n=17

[38] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering: An Introduction*. Kluwer Academic, 2000.

[39] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Dec. 2008. [Online]. Available: http://link.springer.com/content/pdf/10.1007%2Fs10664-008-9102-8.pdf

[40] H. K. Klein and M. D. Myers, "A set of principles for conducting and evaluating interpretive field studies in information systems," *MIS Quarterly*, vol. 23, no. 1, pp. 67–93, Mar. 1999. [Online]. Available: http://www.jstor.org/stable/249410

[41] A. S. Lee, "A scientific methodology for MIS case studies," *MIS Quarterly*, vol. 13, no. 1, pp. 33–50, Mar. 1989. [Online]. Available: http://www.jstor.org/stable/248698

[42] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, Mar. 2004. [Online]. Available: http://www.jstor.org/stable/25148625

[43] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Dec. 2008. [Online]. Available: http://link.springer.com/10.1007/s10664-008-9102-8

[44] P. Kruchten, P. Lago, and H. Vliet, "Building Up and Reasoning About Architectural Knowledge," in *Quality of Software Architectures*, C. Hofmeister, I. Crnkovic, and R. Reussner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 4214, pp. 43–58. [Online]. Available: http://www.springerlink.com/content/k2928ll017kq1516/

[45] T. Dingsyr and H. Vliet, "Introduction to Software Architecture and Knowledge Management," in *Software Architecture Knowledge Management*, M. Ali Babar, T. Dingsyr, P. Lago, and H. Vliet, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–17. [Online]. Available: http://www.springerlink.com/content/p337j716k70k1523/

[46] J. Tyree and A. Akerman, "Architecture decisions: demystifying architecture," *IEEE Software*, vol. 22, no. 2, pp. 19– 27, Apr. 2005.

[47] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (AADL): an introduction," DTIC Document, Technical Report CMU/SEI-2006-TN-011, 2006. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA455842

[48] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: Why reuse is still so hard," *IEEE Software*, vol. 26, no. 4, pp. 66–69, 2009.

[49] L. R. Cai, J. S. Bradbury, and J. Dingel, "Discovering architectural mismatch in distributed event-based systems using software model checking," School of Computing, Queens University, Kingston, Ontario, Canada, Tech. Rep., 2006. [Online]. Available: http://ftp.qucis.queensu.ca/TechReports/Reports/2006-524.pdf

[50] S. Uchitel and D. Yankelevich, "Enhancing architectural mismatch detection with assumptions," in *Proceedings of the 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2000. (ECBS 2000)*, 2000, pp. 138–146.

[51] D. L. Parnas, "Information distribution aspects of design methodology," in *Proceedings of the 1971 IFIP Congress*, November 1971, pp. 26–30.

[52] R. de Lemos, C. Gacek, and A. Romanovsky, "Architectural Mismatch Tolerance," in *Architecting Dependable Systems*, ser. Lecture Notes in Computer Science, R. de Lemos, C. Gacek, and A. Romanovsky, Eds. Springer Berlin Heidelberg, 2003, vol. 2677, pp. 175–194. [Online]. Available: http://dx.doi.org/10.1007/3-540-45177-3_8

[53] A. van Lamsweerde, "Requirements engineering in the year 00: A research perspective," in *Proceedings of the 22nd International Conference on Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 5–19. [Online]. Available: http://doi.acm.org/10.1145/337180.337184

[54] A. Miranskyya, N. Madhavji, M. Davison, and M. Reesor, "Modelling assumptions and requirements in the context of project risk," in *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, ser. RE '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 471–472. [Online]. Available: http://dx.doi.org/10.1109/RE.2005.44

[55] J. Bosch, "Software Architecture: The Next Step," in *Software Architecture*, F. Oquendo, B. C. Warboys, and R. Morrison, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, vol. 3047, pp. 194–199. [Online]. Available: http://www.springerlink.com/content/0vhfq6u9dt6n3jb7/

[56] K. Thompson, "Reflections on trusting trust," *Commun. ACM*, vol. 27, no. 8, pp. 761–763, Aug. 1984. [Online]. Available: http://doi.acm.org/10.1145/358198.358210

[57] J. Viega, T. Kohno, and B. Potter, "Trust (and mistrust) in secure applications," *Commun. ACM*, vol. 44, no. 2, pp. 31–36, Feb. 2001. [Online]. Available: http://doi.acm.org/10.1145/359205.359223

[58] B. Haley, C. Laney, D. Moffett, and B. Nuseibeh, "Using trust assumptions with security requirements," *Requir. Eng.*, vol. 11, no. 2, pp. 138–151, Feb. 2006. [Online]. Available: http://dx.doi.org/10.1007/s00766-005-0023-4

[59] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh, "Security requirements engineering: A framework for representation and analysis," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 133–153, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1109/TSE.2007.70754

[60] V. Page, M. Dixon, and I. Choudhury, "Security risk mitigation for information systems," *BT Technology Journal*, vol. 25, no. 1, pp. 118–127, Jan. 2007. [Online]. Available: http://dx.doi.org/10.1007/s10550-007-0014-8

[61] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[62] A. Tang, "A rationale-based model for architecture design reasoning," Ph.D. dissertation, Faculty of ICT, Swinburne University of Technology, 2007.

[63] I. Ostacchini and M. Wermelinger, "Managing assumptions during agile development," in *Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, ser. SHARK '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 9–16. [Online]. Available: http://dx.doi.org/10.1109/SHARK.2009.5069110

[64] P. Feiler, L. Wrage, and J. Hansson, "System architecture virtual integration: A case study," Software Engineering Institute, CMU, Technical Report CMU/SEI-2009-TR-017, 2009. [Online]. Available: http://erts2010.org/Site/0ANDGY78/Fichier/PAPIERS%20ERTS%202010%202/ERTS2010_0105_final.pdf

[65] N. G. Leveson, "Role of software in spacecraft accidents," *Journal of spacecraft and Rockets*, vol. 41, no. 4, pp. 564–575, 2004.

[66] D. Jackson, M. Thomas, and L. I. Millett, *Software for Dependable Systems: Sufficient Evidence?* National Academies Press, 2007.

[67] M. Ali Babar, J. M. Verner, and P. T. Nguyen, "Establishing and maintaining trust in software outsourcing relationships: An empirical investigation," *J. Syst. Softw.*, vol. 80, no. 9, pp. 1438–1449, Sep. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2006.10.038

[68] N. B. Moe and D. Šmite, "Understanding a lack of trust in global software teams: A multiple-case study," *Softw. Process*, vol. 13, no. 3, pp. 217–231, May 2008. [Online]. Available: http://dx.doi.org/10.1002/spip.v13:3

[69] P.-A. Quinones, S. R. Fussell, L. Soibelman, and B. Akinci, "Bridging the gap: Discovering mental models in globally collaborative contexts," in *Proceedings of the 2009 International Workshop on Intercultural Collaboration*, ser. IWIC '09. New York, NY, USA: ACM, 2009, pp. 101–110. [Online]. Available: http://doi.acm.org/10.1145/1499224.1499241

[70] P. J. Agerfalk, B. Fitzgerald, H. Holmstrom Olsson, B. Lings, B. Lundell, and E. Ó Conchúir, "A framework for considering opportunities and threats in distributed software development," in *Proceedings of the International Workshop on Distributed Software Development.* France: Austrian Computer Society, 2005, pp. 47–61.

[71] E. Rocco, "Trust breaks down in electronic contexts but can be repaired by some initial face-to-face contact," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '98. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1998, pp. 496–502. [Online]. Available: http://dx.doi.org/10.1145/274644.274711

[72] D. Bandow, "Time to create sound teamwork," *The Journal for quality and participation*, vol. 24, no. 2, p. 41, 2001.

[73] G. Piccoli and B. Ives, "Trust and the unintended effects of behavior control in virtual teams," *MIS Q.*, vol. 27, no. 3, pp. 365–395, Sep. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=2017197.2017200

[74] R. Sabherwal, "The role of trust in outsourced is development projects," *Commun. ACM*, vol. 42, no. 2, pp. 80–86, Feb. 1999. [Online]. Available: http://doi.acm.org/10.1145/293411.293485

[75] F. W. Rauskolb, K. Berger, C. Lipski, M. Magnor, K. Cornelsen, J. Effertz, T. Form, F. Graefe, S. Ohl, W. Schumacher, J.-M. Wille, P. Hecker, T. Nothdurft, M. Doering, K. Homeier, J. Morgenroth, L. Wolf, C. Basarke, C. Berger, T. Gülke, F. Klose, and B. Rumpe, "Caroline: An Autonomously Driving Vehicle for Urban Environments," *Journal of Field Robotics*, vol. 25, no. 9, pp. 674–724, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1002/rob.20254

[76] C. Berger and B. Rumpe, "Autonomous Driving - 5 Years after the Urban Challenge: The Anticipatory Vehicle as a Cyber-Physical System," in *Proceedings of the INFORMATIK 2012*, U. Goltz, M. Magnor, H.-J. Appelrath, H. K. Matthies, W.-T. Balke, and L. Wolf, Eds., Braunschweig, Germany, Sep. 2012, pp. 789–798.

[77] P. Hoek, T. Pop, T. Bure, P. Hntynka, and M. Malohlava, "Comparison of component frameworks for real-time embedded systems," in *Component-Based Software Engineering*, ser. Lecture Notes in Computer Science, L. Grunske, R. Reussner, and F. Plasil, Eds. Springer Berlin Heidelberg, Jan. 2010, no. 6092, pp. 21–36. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-13238-4_2

[78] ARCCORE AB, "Arctic Core Open Source AUTOSAR Embedded Platform," http://www.arccore.com/products/arctic-core/, Jun. 2013.

[79] OpenJAUS LLC, "OpenJAUS," http://www.openjaus.com, Jul. 2012.

[80] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, D. Anderson, S. Cacciola, P. Currier, A. Dalton, J. Farmer, J. Hurdus, S. Kimmel, P. King, A. Taylor, D. V. Covern, and M. Webster, "Odin: Team VictorTango's Entry in the DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 467–492, Sep. 2008.

[81] A. Makarenko, A. Brooks, and T. Kaupp, "Orca: Components for Robotics," in *International Conference on Intelligent Robots and Systems*, Beijing, China, 2006, pp. 163–168.

[82] CARMEN, "CARMEN - Robot Navigation Toolkit," http://carmen.sourceforge.net/, Jun. 2013.

[83] B. Finkemeyer, T. Kröger, D. Kubus, M. Olschewski, and F. Wahl, "MiRPA: Middleware for Robotic and Process Control Applications," 2007.

[84] Willow Garage, "ROS - Robot Operating System," http://www.ros.org/, Jun. 2013.

[85] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," in *Proceedings of the 11th International Conference on Advanced Robotics*, Coimbra, Portugal, 2003, pp. 317–323.

[86] C. Berger, M. Chaudron, R. Heldal, O. Landsiedel, and E. M. Schiller, "Model-based, Composable Simulation for the Development of Autonomous Miniature Vehicles," in *Proceedings of the SCS/IEEE Symposium on Theory of Modeling and Simulation*, San Diego, CA, USA, Apr. 2013.

[87] C. Berger and B. Rumpe, "Engineering Autonomous Driving Software," in *Experience from the DARPA Urban Challenge*, C. Rouff and M. Hinchey, Eds. London, UK: Springer-Verlag, 2012, pp. 243–271. [Online]. Available: http://dx.doi.org/10.1007/978-0-85729-772-3_10

[88] M. K. Berkenbusch, "Quad with optical flow position hold," http://www.diydrones.com/profiles/blogs/quad-with-optical-flow, Jun. 2013.

[89] C. Berger, "From Autonomous Vehicles to Safer Cars: Selected Challenges for the Software Engineering," in *Proceedings of the SAFECOMP 2012 Workshops, LNCS 7613*, F. Ortmeier and P. Daniel, Eds. Magdeburg, Germany: Springer-Verlag Berlin Heidelberg, Sep. 2012, pp. 180–189.

[90] C. Berger, M. A. Al Mamun, and J. Hansson, "COTS-Architecture with a Real-Time OS for a Self-Driving Miniature Vehicle," in *Proceedings of the 2nd Workshop on Architecting Safety in Collaborative Mobile Systems (ASCoMS)*, E. Schiller and H. Lönn, Eds., Toulouse, France, Sep. 2013, pp. 411–422.

[91] STMicroelectronics, "Discovery kit for STM32F407/417 line," http://goo.gl/hs7X28, Aug. 2013.

[92] "Eclipse Modeling - EMF," http://www.eclipse.org/modeling/emf/?project=emf.

[93] M. A. A. Mamun, M. Tichy, and J. Hansson, "Towards formalizing assumptions on architectural level: A proof-of-concept," Chalmers University of Technology and Gothenburg University, Gothenburg, Sweden, Technical Report 2012:02, 2012.

[94] "Acceleo," http://projects.eclipse.org/projects/modeling.m2t.acceleo.

[95] M. Botts and A. Robin, "OpenGIS sensor model language (SensorML) implementation specification," Open Geospatial Consortium Inc., OpenGIS Implementation Specification OGC 07-000, 2007. [Online]. Available: portal.opengeospatial.org/files/?artifact_id=21273

[96] K. L. Headley, D. Davis, D. Edgington, L. McBride, T. C. O'Reilly, and M. Risi, "Managing Sensor Network Configuration and Metadata in Ocean Observatories Using Instrument Puck," in *Proceedings of the 3rd International Workshop on Scientific Use of Submarine Cables and Related Technologies*, 2003, pp. 67–70. [Online]. Available: http://www.mbari.org/pw/SSC03_submitted.pdf

[97] "IEEE standard for a smart transducer interface for sensors and actuators wireless communication protocols and transducer electronic data sheet (TEDS) formats," *IEEE Std 1451.5-2007*, pp. C1–236, 2007.

[98] C. Basarke, C. Berger, K. Berger, K. Cornelsen, M. Doering, J. Effertz, T. Form, T. Gülke, F. Graefe, P. Hecker, K. Homeier, F. Klose, C. Lipski, M. Magnor, J. Morgenroth, T. Nothdurft, S. Ohl, F. W. Rauskolb, B. Rumpe, W. Schumacher, J. M. Wille, and L. Wolf, "Team CarOLO - Technical Paper," Technische Universität Braunschweig, Braunschweig, Germany, Informatik-Bericht 2008-07, Oct. 2008.

[99] A. Schuster and J. Sprinkle, "Synthesizing Executable Simulations from Structural Models of Component-Based Systems," in *Proceedings of the 3rd International Workshop on Multi-Paradigm Modeling*, vol. 21, 2009, pp. 1–10.

[100] STMicroelectronics, "MicroXplorerMCU graphical configuration tool," http://goo.gl/3UUgdh, Aug. 2013.

[101] CooCox, "CoSmart," http://www.coocox.org/CoSmart.html, Aug. 2013.

[102] B. Joshi, F. M. Rizwan, and R. Shettar, "MICROCONTROLLER PIN CONFIGURATION TOOL," *International Journal on Computer Science and Engineering*, vol. 4, no. 05, pp. 886–891, 2012.

[103] J. A. Berlier and J. M. McCollum, "A Constraint Satisfaction Algorithm for Microcontroller Selection and Pin Assignment," in *Proceedings of the 2010 IEEE SoutheastCon*, Concord, NC, Mar. 2010, pp. 348–351.

[104] S. Siegl, K.-S. Hielscher, R. German, and C. Berger, "Formal Specification and Systematic Model-Driven Testing of Embedded Automotive Systems," in *Proceedings of the Conference on Design, Automation, and Test in Europe*. Grenoble, France: European Design and Automation Association, Mar. 2011, pp. 1530–1591.

[105] M. A. A. Mamun, C. Berger, and J. Hansson, "Engineering the hardware/software interface for robotic platforms - a comparison of applied model checking with prolog and alloy," in *arXiv:1401.3985 [cs]*, C. Schlegel, U. P. Schultz, and S. Stinckwich, Eds., Tokyo, Japan. [Online]. Available: http://arxiv.org/abs/1401.3985

[106] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, 5th ed. Springer Berlin Heidelberg, 2003.

[107] S. L. Pfleeger, "Design and analysis in software engineering: The language of case studies and formal experiments," *SIGSOFT Softw. Eng. Notes*, vol. 19, no. 4, pp. 16–20, Oct. 1994. [Online]. Available: http://doi.acm.org/10.1145/190679.190680

[108] M. K. Sein, O. Henfridsson, S. Purao, M. Rossi, and R. Lindgren, "Action Design Research," *MIS Quaterly*, vol. 35, no. 1, pp. 37–56, Mar. 2011.

[109] G. Gupta and E. Pontelli, "A Constraint-based Approach for Specification and Verification of Real-time Systems," in *Proceedings of the Real-Time Systems Symposium*. IEEE Comput. Soc, Dec. 1997, pp. 230–239. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=641285

[110] C. Ghezzi, D. Mandrioli, and A. Morzenti, "TRIO : A Logic Language for Executable Specifications of Real-Time Systems," *Journal of Systems and Software*, vol. 12, no. 2, pp. 107–123, May 1990.

[111] J. Hirsch, "Self-driving cars inch closer to mainstream availability," Oct. 2013. [Online]. Available: http://www.latimes.com/business/autos/la-fi-adv-hy-self-driving-cars-20131013,0,5094627.story

[112] S. Thrun, "What we're driving at," p. 1, Apr. 2010. [Online]. Available: http://googleblog.blogspot.se/2010/10/what-were-driving-at.html

[113] A. Kyte, "Measure and manage your IT debt," Gartner and CAST, Research Report, 2010. [Online]. Available: http://imagesrv.gartner.com/media-products/pdf/cast_software/gartner2.pdf

[114] Z. Codabux and B. Williams, "Managing technical debt: An industrial case study," in *Proceedings of the 4th International Workshop on Managing Technical Debt (MTD)*, May 2013, pp. 8–15.

[115] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. da Silva, A. L. M. Santos, and C. Siebra, "Tracking technical debt - an exploratory case study," in *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2011, pp. 528–531.

[116] N. Zazworka, M. A. Shaw, F. Shull, and C. Seaman, "Investigating the impact of design debt on software quality," in *Proceedings of the 2nd Workshop on Managing Technical Debt (MTD)*. New York, NY, USA: ACM, 2011, pp. 17–23. [Online]. Available: http://doi.acm.org/10.1145/1985362.1985366

[117] J. Xuan, Y. Hu, and H. Jiang, "Debt-prone bugs: technical debt in software maintenance," *International Journal of Advancements in Computing Technology 2012a*, vol. 4, no. 19, pp. 453–461, 2012.

[118] J. Letouzey and M. Ilkiewicz, "Managing technical debt with the sqale method," *Software, IEEE*, vol. 29, no. 6, pp. 44–51, Nov 2012.

[119] A. Martini, J. Bosch, and M. Chaudron, "Architecture technical debt: Understanding causes and a qualitative model," in *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, Verona, Italy, Aug 2014, pp. 85–92.

[120] C. Berger, "From a Competition for Self-Driving Miniature Cars to a Standardized Experimental Platform: Concept, Models, Architecture, and Evaluation," *Journal of Software Engineering for Robotics*, vol. 5, no. 1, pp. 63–79, Jun. 2014. [Online]. Available: http://arxiv.org/abs/1406.7768

[121] C. Berger, E. Dahlgren, J. Grunden, D. Gunnarsson, N. Holtryd, A. Khazal, M. Mustafa, M. Papatriantafilou, E. M. Schiller, C. Steup, V. Swantesson, and P. Tsigas, "Bridging Physical and Digital Traffic System Simulations with the Gulliver Test-bed," in *Proceedings of 5th International Workshop on Communication Technologies for Vehicles 2013*, Lille, France, May 2013, pp. 169–184. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37974-1_14

[122] M. Fowler, K. Beck, J. Brant, and W. Opdyke, *Refactoring: Improving the Design of Existing Code*.

[123] M. Buehler, K. Iagnemma, and S. Singh, Eds., *The 2005 DARPA Grand Challenge: The Great Robot Race*. Berlin Heidelberg: Springer Verlag, 2007.

[124] C. Berger and M. Dukaczewski, "Comparison of Architectural Design Decisions for Resource-Constrained Self-Driving Cars - A Multiple Case-Study," in *Proceedings of the INFORMATIK 2014*, Stuttgart, Germany, Sep. 2014, p. 12.