Tool Integration and Safety

A Foundation for Analysing the Impact of Tool Integration on Non-functional Properties

FREDRIK ASPLUND

Licentiate Thesis
Stockholm, Sweden, 2012
Academic thesis, which with the approval of the Royal Institute of Technology, will be presented for public review in fulfillment of the requirements for a Licentiate of Technology in Machine Design. The public review is held in Room B319, Department for Machine Design, Royal Institute of Technology, Brinellvägen 85, Stockholm on 2012-10-05 at 09.00.

© Fredrik Asplund, 2012

Print: Universitetsservice US AB
Abstract

The increasing complexity of embedded systems development is becoming difficult to handle with development environments based on disjoint engineering tools. Support for interactions between various engineering tools, especially through automated means, has therefore received an increased amount of attention during the last few years. The subsequent increase in the amount of tool integration is leading to an increased impact of tool integration on non-functional properties of development efforts, development environments and end products. At the same time there is a lack of methods and tools for analysing the relationship between these properties and tool integration. To establish a foundation for analysing this generic relationship, the specific relationship between tool integration and the safety of end products is analysed in this thesis.

A survey was conducted to analyze the State of the Art of tool integration as related to safety. This survey specifically identified the lack of an efficient handling of tool integration by modern safety standards as an important concern. In relation to this survey, three theories were identified as of specific importance. These are the school of thought known as Systems Thinking, the Systems-Theoretic Accident Model and Processes (STAMP) causality model and the System-Theoretic Process Analysis (STPA) hazard analysis technique.

Building on these theories, this thesis provides original contributions intended to (1) describe concepts and models related to tool integration and safety (the first and second contribution), (2) link tool integration to safety in a way that reduces complexity during analysis (the third contribution) and (3) propose how to interpret and make use of the implications of the presented theories and the first three contributions (the fourth and fifth contribution).

- The first contribution is a new conceptual model of a development effort that emphasizes tool integration.
- The second contribution is a new reference model for tool integration in highly heterogeneous environments.
- The third contribution consists of nine safety-related tool chain properties, i.e. properties of tool chains that could mitigate at least part of the risks introduced by tool integration.
- The fourth contribution is a proposition on how to identify safety implications due to a high level of automation of tool integration.
- The fifth contribution is a proposition for a new software tool qualification process.

Keywords. Tool Integration, Integrated Development Environments, Embedded Systems, Safety Standards, Certification, Qualification, Systems Thinking, System Safety, STAMP, STPA
Terminology

The meaning of many terms found in this paper differs depending on the context in which they are used or who is using them. For the reason of providing a common starting point for readers and avoiding misunderstandings some of the most important terms, as well as some of the most commonly used terms, are defined here.

- **Accident:** An undesired and unplanned (but not necessarily unexpected) event that results in (at least) a specified level of loss [1].

- **Certification/Qualification:** To confirm, according to predefined rules, a certain set of characteristics that an end product has been claimed to exhibit. In this thesis certification is used to refer to the whole set of activities to accomplish this confirmation, while qualification refers to a smaller subset of these activities linked to a specific area of concern.

- **Control Loop:** A loop formed by (1) a controller, (2) a process that the controller is trying to control according to some goal, (3) the variables of the controlled process that the controller can manipulate and (4) the information on the current process state that the controller can obtain from measurable variables (feedback). If the controller is using feedback to control the controlled process the control loop is defined as closed, otherwise it is defined as open.

- **Development Effort:** Everyone involved, all items required and the activities involved in creating an end product.

- **Development Environment:** The integrated set of all engineering tools and any supporting software used during an development effort.

- **Embedded System:** A system that is part of a larger system and performs some special purpose for that system (in contrast to a general-purpose part that is meant to be continuously configured to meet the demands of its users) [2].
• **Emergence**: When a property of an upper level (see **Hierarchy**) cannot be reduced to isolated parts at lower levels, since this property depends on the interaction between the parts at the lower levels.

• **End product**: An artifact, i.e. an object formed by humans.

• **Engineering Tool**: A software tool used to perform one or several engineering tasks in the product life-cycle.

• **Hazard**: A system state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object), will lead inevitably to an accident (loss event) [1].

• **Hazard Level**: The combination of the severity and likelihood of the occurrence of a hazard.

• **Hierarchy**: A number of hierarchical levels, which all contain entities. The entities of each level will depend on the criteria used to link the levels to each other and the properties of said entities will characterize the different levels [3].

• **Holon**: Entities on a specific hierarchical level that are identifiable as parts when viewed from an upper level, but as self-contained wholes when viewed from a lower level [4].

• **Level of organization**: A hierarchical level of a specific type, which place in a hierarchy is defined solely based on definitions that relate the level to those above and below [3].

• **Operator**: A person responsible for carrying out some (set of) task(s) within a development effort.

• **Product Life-cycle**: The entire lifetime of an end product, from the very first specification of its requirements up to the point at which it is disposed of. This time span is usually divided into a number of phases according to the sequence of engineering activities that usually take place across the lifetime of an end product.

• **Risk**: The hazard level combined with (1) the likelihood of the hazard leading to an accident (sometimes called danger) and (2) hazard exposure or duration (sometimes called latency) [1]. In short, a risk can be described as a measure of the harmful effects of an event. The use of the term risk is often limited only to **product risks**, i.e. the risks that are inherent to end products and direct in nature. An example of a product risk would be the failure of the lock on the baggage door to an aircraft leading to the baggage door opening in mid-flight. A simple model of the risk of the lock failing could be expressed as a function of (1) the probability that the lock is released in mid-flight,
The probability that the baggage door opens if the lock is released, the probability that this is not noted by the pilots or that they do not have enough time to react to this occurrence and the severity of the worst-case consequences. The definition of a risk is however usually less rigorous, only mentioning what might go wrong and the consequences of this (leaving out the probabilities). In this thesis the systems that are examined include more than end products and the term risk is therefore used to include risk of an indirect nature. An example of this would for instance be the process risk that an electrical engineer fails to understand that the baggage door lock he is designing might be influenced by electromagnetic disturbances. This is line with the wider use of the term risk in [5].

- **Safety**: Freedom from accidents or losses [1].

- **Safety Control Structure**: A system description with all system components relevant to safety and their interactions included [5].

- **Tool Chain**: An ordering of tools that supports a development process.

- **Tool Integration**: The activities and automation aimed at supporting the transition between different engineering tools across the product life-cycle, i.e. activities and automation with this purpose that are prepared and executed without the possibility to directly affect (for instance through source code updates) the internals of the engineering tools at hand.

  - **Manual Tool Integration**: Tool integration which relies on operators making all the decisions necessary to carry out the transition in question. An example is when an operator transfers data to a document using a word processor and therefore has to make all necessary decisions in regard to semantics and syntax.

  - **Automated Tool Integration**: Tool integration which relies on software making some or all of the decisions necessary to carry out the transition in question. An example is when an engineering tool delivers its output directly to another engineering tool and therefore has to contain the necessary logic to handle the semantics and syntax of the output.

- **Tool Integration Aspects**: The tool integration aspects referred to in this thesis are platform, control, data, process and presentation integration. While tool integration was first divided into these aspects by Wasserman in a reference model presented in [6], the definitions of these aspects used in this thesis differs from the original definitions (Paper A).

  - **Platform Integration** is the degree to which tools share a common environment.
– *Control Integration* is the degree to which tools can issue commands and notifications correctly to one another.

– *Data Integration* is the degree to which tools can, in a meaningful way, access the complete data set available within the system of interest.

– *Process Integration* is the degree to which the interaction with tools can be supervised.

– *Presentation Integration* is the degree to which tools, in different contexts, can set up user interaction so that the tool input/output is perceived correctly.

• *Tool Integration Mechanism (TIM):* The smallest identifiable parts of a development environment that provide functionality beneficial to tool integration, i.e. a TIM is a software tool or part of a software tool intended to support the transition between different engineering tools. Examples include transformation engines (that transform the output data of one engineering tool to a data format readable by another engineering tool), bug tracking tools (that support the decisions to move from one stage of the product life-cycle to another), versioning tools (that ensure the integrity of the data handled by engineering tools) and specialized data viewers (that support the correct interpretation of data prepared within one engineering domain).
Acknowledgments

First I would like to thank my academic supervisors, Jad El-khoury and Martin Törngren. Without your guidance and support I could not have finished this thesis or the papers it is based on.

Secondly I would like to thank all the participants of the ARTEMIS iFEST project, who have shared their expertise on and experience of tool integration with me. Without your help this thesis would have been of far less value and the work towards completing it not nearly as efficient.

Last but not least, my thanks go to my family, both immediate and extended. To Sofie for her love and support. To my daughter Majken for pointing out that a rainy day is sometimes better spent outdoors, than indoors writing a thesis.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminology</td>
<td>v</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>ix</td>
</tr>
<tr>
<td>Contents</td>
<td>x</td>
</tr>
<tr>
<td>Appended Publications</td>
<td>xiii</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Background .................................. 1
1.2 Problem Definition .......................... 4
1.3 Thesis Scope ................................ 6
1.4 Thesis Objective ............................ 7
1.5 Thesis Approach ................................ 8
1.6 Thesis Outline ................................ 9
1.7 Chapter Summary ................................ 11

## 2 State of the Art

2.1 Tool Integration and Software Engineering .... 13
2.2 Tool Integration as a Specialized Field of Study ... 14
2.3 Tool Integration and Safety .................... 16
2.4 Chapter Summary ................................ 20

## 3 Systems Thinking, System Safety

3.1 The Origins of Systems Thinking ............... 21
3.2 Hierarchy and Emergence ........................ 22
3.3 Communication and Control ..................... 23
3.4 Systems-Theoretic Accident Model and Processes ... 24
3.5 System-Theoretic Process Analysis ............... 25
   System-Theoretic Process Analysis, Step 1 .......... 25
   System-Theoretic Process Analysis, Step 2 .......... 27
3.6 Chapter Summary ................................ 28
4 Tool Integration Concepts and Models 31

4.1 A New Conceptual Model of an Embedded Systems Development Effort 31
   The Management Level 32
   The Operator Level 33
   The Tool Chain Level 33
   The Tool Level 33

4.2 A New Reference Model for Tool Integration 33
   Platform Integration 34
   Control Integration 35
   Data Integration 35
   Process Integration 35
   Presentation Integration 36

4.3 A Mapping of the Integration Aspects 37

4.4 Chapter Summary 37

5 Tool Integration and Safety 41

5.1 Analysing an Industrial Tool Chain using STPA 41
   Step 1 - Identify the General Types of Risk Associated with Tool integration 42
   Step 2 - Construction and Analysis of a Control Structure 42
   Step 3 - Identify the Causes to the Programmatic Risks 43
   Step 4 - Analysis and Reiterations 47

5.2 Chapter Summary 49

6 Towards New Methods for Software Tool Qualification 53

6.1 The Implications of the Tool Integration Metrics 53
   Platform Integration 54
   Control Integration 55
   Data Integration 56
   Process Integration 57
   Presentation Integration 58

6.2 A New Software Tool Qualification Process 58
   The Proposition for a New Software Tool Qualification Process 59
   Dealing with the Shortcomings 60
   Combining with Modern Safety Standards 60
   Dealing with the Complexity 60
   Additional Benefits 61

6.3 Chapter Summary 62

7 Summary, Contributions, Future Work and Conclusions 65

7.1 Summary 65

7.2 Contributions 66
| CONTENTS |
|-----------------------------------------------|---|
| A New Conceptual Model of a Development Effort Emphasizing Tool Integration | 66 |
| A New Reference Model for Tool Integration | 67 |
| Safety-related Tool Chain Properties | 68 |
| The Implications for Safety due to a High Level of Automation of Tool Integration | 69 |
| A New Software Tool Qualification Process | 70 |
| 7.3 Future Work | 71 |
| Concerns | 71 |
| Future Venues of Research | 72 |
| 7.4 Conclusions | 76 |
| Bibliography | 79 |
| Appended Paper A | A1 |
| Appended Paper B | B1 |
| Appended Paper C | C1 |
| Appended Paper D | D1 |
Appended Publications

• **Paper A**
  Fredrik proposed the changes to Wasserman’s reference model, conducted the case study and wrote the paper. Jad, Martin and Matthias provided feedback.

• **Paper B**
  Fredrik Asplund, Matthias Biehl, Jad El-khoury, Daniel Frede, and Martin Törngren, “Tool Integration, from Tool to Tool Chain with ISO 26262,” in *SAE 2012 World Congress & Exhibition*, 2012.
  Fredrik defined the case study and wrote the paper. Daniel conducted the case study together with Fredrik. Jad, Martin and Matthias provided feedback.

• **Paper C**
  Fredrik wrote the paper. Jad and Martin provided feedback.

• **Paper D**
  Fredrik wrote the paper. Jad and Martin provided feedback.
Chapter 1

Introduction

This thesis discusses the connection between tool integration in modern development environments and safety. Chapter 1 details the background of the state of tool integration today, the problems with analysing the impact tool integration has on non-functional properties and the limitations and focus of this thesis in dealing with these problems. Additionally this chapter positions all the chapters of this thesis by defining their relationship to the research publications that the thesis is based on and their relationship to each other during reading.

1.1 Background

*Systems Engineering* went from the periphery of established engineering practice to being recognized as a core part of modern development efforts during the first part of the 20th century. While the foundation was laid in the communication and aircraft industries, in the 1950s Systems Engineering had reached general acceptance in many fields. The primary reason for this general acceptance was the failed development efforts associated with the increasing complexity\(^1\) of the systems that were becoming possible to realize [11]. Systems Engineering gave engineers the tools to handle this increase in complexity by providing a systematic approach to the product life-cycle, as the *International Council On Systems Engineering* (INCOSE) highlights in their definition of Systems Engineering:

“Systems Engineering (SE) is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle,

\(^1\)Sussman provides an organized survey into different definitions of complexity in [7], mentioning Moses’s view that “a system is complicated when it is composed of many parts interconnected in intricate ways” [8], Senge’s “when an action has one set of consequences locally and a different set of consequences in another part of the system, there is dynamic complexity” [9] and Sterman’s enumeration of system characteristics that create complexity (tightly coupled, governed by feedback, nonlinear, adaptive, counterintuitive, etc.) [10].
documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance, training and support, test, manufacturing, and disposal. SE considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs.” [12]

Similarly, during the second half of the 20th century the Software Engineering discipline went from first being defined in 1968 to wider acceptance in the 1990s. The reason for the establishment of this discipline was roughly the same as that of Systems Engineering, i.e. the problems associated with the pressure to produce even bigger and more sophisticated systems [13]. The approach of Software Engineering is also similar to that of Systems Engineering, as seen in the definition of Software Engineering by the Institute of Electrical and Electronics Engineers (IEEE):

“(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).” [14].

These two disciplines have had a strong influence on each other, noticeable for instance in the many formalisms currently used in Systems Engineering that have their roots in Software Engineering [15] and the standards for systems development that have defined the applicability of different Software Engineering principles in a systems development context (see for instance [16]).

Another strong connection can be traced to the use of engineering tools, i.e. the software tools used to perform the different engineering tasks in the product life-cycle, by system and software engineers. Without these it would be not be possible to cost-effectively handle the complexity of a modern development effort [17].

An engineering tool executes in the context of an integrated set of all other engineering tools and any supporting software used during development, i.e. the development environment. An important aspect of a development environment is that the development processes it supports will define orderings of the engineering tools it contains, i.e. tool chains. System and software engineers can use different engineering tools due to differences in emphasis and activities. However, the need to ensure that development environments support an efficient and correct transition between engineering tools across the tool chains is the same, since handling complexity across the whole product life-cycle would otherwise be difficult.

The research into tool integration is the research into the activities and automation aimed at supporting this transition between different engineering tools across the different development phases that make up the product life-cycle. In other words, tool integration is defined by (1) a focus on activities and automation aimed at supporting the transition between engineering tools when (2) the internals of these engineering tools cannot be directly affected (for instance through source
code updates) during the preparation and execution of these activities and automation. This topic has been studied intensively in relation to Software Engineering research since the 1980s \cite{18}, especially in relation to the discussion on Software Engineering Environments (SEE), Computer-Aided Software Engineering (CASE) and Integrated Programming Support Environments (IPSE) (see \cite{14} for a definition of these terms). As such, the research into tool integration has had a clear purpose in the question on how to best implement such environments to support their goals, i.e. ensuring adherence to standards, enabling efficiency gains, increasing productivity, etc. This may be one of the primary reasons why the research into tool integration has almost exclusively focused on technology (\cite{18} identifies an almost universal focus on technology among the papers published on tool integration during the last three decades).

Today, however, the increasing complexity of realizable systems is pushing the boundaries of what is cost-effectively possible to accomplish with a development environment consisting of disjoint engineering tools. Envision, for instance, placing an embedded microcontroller in an hostile environment, such as in close proximity to an engine. This may have strong implications on electrical, mechanical and software engineering. A change required by the engineers working within one of the engineering disciplines may have unforeseen and adverse effects within the domains of the other engineering disciplines. The problem of coordinating the different disciplines to avoid these effects is currently (typically) handled by relying on experts, i.e. engineers that have worked for a very long time on the same (kind of) system. These engineers can, through their intuitive understanding of the system, foresee how changes will affect the system and ensure that engineers from all relevant disciplines are notified of the new circumstances. Such a setup is vulnerable in several ways, experts may for instance choose to leave a company or simply fail to identify a particularly obscure relationship between the domains of two engineering disciplines. It is also an ineffective way of developing an end product, since development artifacts belonging to different engineering disciplines and software phases will be built independently even though they will have to contain much overlapping information\(^2\).

This vulnerability and ineffectiveness is aggravated by the increasing complexity, since there is a limit to how much any single expert can comprehend through detailed knowledge or intuitive deduction. An increased need for interaction between different engineering disciplines is thus likely. Therefore, even if an imperative requirement for tool integration is never realized, a trade-off decision between how much effort and resources should be spent on tool integration versus the benefits that will come out of these investments is increasingly likely to come out in the favour of an increase in tool integration. In the (common) case where the subsets of data that the different engineering tools are dealing with are also complex this increase of tool integration is likely to be realized in the form of automated tool

\(^2\)This point is made in the context of Model-Based Development, which is often hampered by a lack of tool integration, in \cite{19}.
integration.

In addition to the effects of an increase in interaction between engineers, there is also a requirement in modern development efforts for functionality that in itself requires integration of tools throughout large parts of the product life-cycle. Examples of properties of tool chains that are in demand include:

- Traceability between requirements and other types of artifacts to ensure completeness and consistency (required by requirements engineers).

- Bridging between designs at high and low levels of abstraction to ensure a fast turnaround time (required by system engineers).

- Support for impact analysis across the whole product life-cycle to guide decision making (required by managers).

This increase in tool integration is likely to lead to a greater impact of tool integration on non-functional properties (such as safety, cost and security) of the development environments. Furthermore, the non-functional properties of the end products associated with these development efforts are also likely to be affected, albeit indirectly. These non-functional properties are tightly tied to questions concerning the behaviour of operators, economy, law, the product life-cycle, etc. Due to Systems Engineering in turn taking this type of questions into consideration, it is therefore likely that the increase in tool integration will lead to a stronger need for Systems Engineers to take tool integration into account as an important factor of a development effort. There will therefore be a need for further analyses of tool integration that takes a wider perspective, a systems perspective.

1.2 Problem Definition

The focus on technology described in the previous section is understandable. The benefits are immediately obvious to users, the limitation of verification efforts to only include technological aspects can be reasonable well justified, the need is readily seen by those procuring development software, etc. Many important questions related to tool integration are, however, not necessarily possible to answer even with access to a detailed knowledge of all Tool Integration Mechanisms\(^3\) (TIMs) in a development environment.

\(^3\)A TIM is one of the smallest identifiable parts of a development environment that provide a functionality beneficial to tool integration, i.e. a TIM is a software tool or part of a software tool intended to support the transition between different engineering tools. The TIM concept is therefore very broad and encompasses such diverse software tools as transformation engines (that transform the output data of one engineering tool to a data format readable by another engineering tool), bug tracking tools (that support the decisions to move from one stage of the product life-cycle to another), versioning tools (that ensure the integrity of the data handled by engineering tools) and specialized data viewers (that support the correct interpretation of data prepared within one engineering domain).
1.2. PROBLEM DEFINITION

- TIMs impact the cost of a development effort greatly and therefore indirectly impact the cost of the resulting end product. TIMs can be bought as Commercial Of The Shelf (COTS) products, but will usually require at least in-house configuration support. To introduce TIMs will therefore incur a cost related to both tool chain development and software support. On the other hand, to not introduce TIMs will incur a cost related to inefficient manual efforts needed to move between engineering tools. To know the optimal number and type of TIMs to introduce is difficult, since the cost of such a choice is continuous and related to as diverse factors as the skill level of in-house software support, the difference in domains handled by the engineering tools used throughout development, etc.

- The impact that TIMs have on the security of a development environment is tied to the context in which it is deployed and therefore needs to be analysed on a case-by-case basis. For instance, a large multinational organization may experience great benefit from TIMs that gather information from multiple sites and provide analyses regarding the status of project deadlines, the trends in software bugs, available expert knowledge, etc. However, if not handled in a secure fashion, such information may leak and severely jeopardize market competitiveness.

- The safety of end products is influenced by how engineering tools and TIMs interrelate with the development artifacts produced throughout the product life-cycle. A TIM may for instance introduce incorrect or unintended information in a development artifact, information that may propagate into the end product and cause unsafe behaviour.

As seen above, there are not always direct, clearly discernible relationships between TIMs and the non-functional properties of development environments and end products. Instead there can exist a complex web of interdependencies related to technical, social, process, organizational and management aspects of development efforts, to name just a few of the factors with a potential impact on these relationships. These interdependencies prohibit the reduction of a development effort into parts that can be studied in isolation, since the interaction between these parts are important when describing their impact on properties of the development effort.

In addition, TIMs typically exist at different levels of a software hierarchy in which higher layers build their functionality on top of the services provided by lower layers. The context, i.e. the information available for the software to act on, is different at each level. This means that even if the complexity was possible to dismantle one would have to identify the value of introducing changes at a particular level based on what is possible to achieve at that level.\(^4\)

To identify the wider implications of choosing to introduce a certain set of TIMs is therefore difficult based on the current State of the Art in tool integration

\(^4\)Saltzer, Reed and Clark discusses the problems with this and how changes at lower levels might be redundant or of little value compared to the cost of introducing them in [20].
research. There is a lack of methods and tools for analysing how non-functional properties of development efforts, development environments and end products are related to tool integration. At least for those non-functional properties that are not possible to deduce by observing the different parts of the development effort, development environment or end product in isolation.

1.3 Thesis Scope

The overall aim of this thesis is to provide a foundation for analysing the relationship between tool integration and the non-functional properties of development efforts, development environments and end products. This has prompted the following limitations in regard to the contents of this thesis:

- To allow for a more in-depth analysis the focus is solely on the relationship between tool integration and the safety of end products. This choice was based on the knowledge that safety has been studied extensively in a system context, hopefully allowing for much relevant theory to relate to during the research into the wider implications of tool integration.

- The emphasis is placed on changes in tool integration. Specifically, due to the use of increasingly elaborate automation in modern development environments, the emphasis is placed on the change from manual to more or fully automated tool integration. In other words, the focus is on the change from a situation where the decisions on how to execute the transition between the engineering tools in question are made by operators, to a situation where some or all of the decisions are made in software. In this way one can start with generalizing the move from manual to automated tool integration and therefore potentially avoid the problem of first having to generalize the virtually unlimited possibilities of realizing and combining tool integration activities and automation.

- The case studies conducted during the research towards this thesis were exclusively concerned with embedded systems\(^5\). Therefore, of all realizable development environments, this thesis limits itself to consider those that are set up to produce embedded systems.

- The case studies conducted towards this thesis were also more complete in regard to activities in the phases from the input of requirements to the development effort to the output of a product fit for release to manufacturing\[21\]. Therefore this thesis considers these phases of the product life-cycle. Later phases, such as phases concerned with production and maintenance, are excluded.

\(^5\)An embedded system is a system that is part of a larger system and performs some special purpose for that system\[2\].
1.4. **THESIS OBJECTIVE**

The intention is that the findings within these boundaries will later support a more universal analysis.

### 1.4 Thesis Objective

Based on the problem definition and the scope of the thesis one main research question and an associated hypothesis can be defined.

- **Main Research Question:** What is the relationship between tool integration in development environments and the safety of embedded systems developed by use of said environments?

- **Main Research Question Hypothesis:** Tool integration activities and automation affect the development artifacts produced throughout the whole product life-cycle. An end product may manifest unsafe behaviour through incorrect handling of, or erroneous information entered into, development artifacts by tool integration activities or automation. This introduction of unsafe behaviour may be avoided by constraining different parts of an embedded systems development effort. Ultimately, these constraints will define the relationship between tool integration and the safety of embedded systems.

Additionally, two derivative research questions and hypotheses can be based on the main research question above.

- **Derivative Research Question 1:** Safety standards based on Systems and Software Engineering principles define the boundaries for development efforts targeting safety-critical, embedded systems. Do these safety standards provide guidance on tool integration? If they do, what form does this guidance take and is it different in standards targeting different industries or between different versions of the same standards?

- **Derivative Research Question 1 Hypothesis:** Safety standards are likely to define activities to ensure safety when transitioning between different phases of the product life-cycle, but at the same time they are likely to strive towards being technology agnostic.

- **Derivative Research Question 2:** What are the differences between manual and automated tool integration in relation to the safety of embedded systems? Is one to prefer over the other in the context of the safety of end products?

- **Derivative Research Question 2 Hypothesis:** The differences will probably depend on whether the automated tool integration is meant to replace or support human operators, but neither solution is likely to be a silver bullet. Most likely these different solutions will rather be interrelated in some way.
1.5 Thesis Approach

The research results described in this thesis are based on three iterations of research conducted across the last two years.

1. The aim of the first iteration was to enable an understanding of the basic concepts of tool integration and it therefore started with a general survey into tool integration. The findings of this survey were primarily published in an internal iFEST project deliverable (see [22] for a description of the iFEST project). The survey also, however, enabled the identification of the reference model published by Wasserman in [6] as a likely candidate from which one could create a reference model for reasoning about tool integration in highly heterogeneous environments (such as in embedded systems development efforts). Such a reference model was presented in (Paper A) together with a case study of an industrial tool chain for the development of embedded, closed-loop control systems. All weaknesses of this tool chain that were identified by expert knowledge and at all possible to identify through the information given were also identified by the new reference model. Additionally, some of the identified weaknesses would not have been obvious if the unmodified reference model presented in [6] had been used. The results of this iteration, i.e. the details of the new reference model, are found in section 4.2 of this thesis.

2. The use of the new reference model requires adequate knowledge of how different non-functional properties relate to tool integration. In a second iteration a case study was therefore conducted to identify the relationship between tool integration and functional safety as defined by the ISO 26262 safety standard. This case study focused on the tool chain used in the Spiros project to develop a hydrogen leakage warning system for an Urban Concept vehicle competing in the 2011 Shell Eco-Marathon (see [23] for a description of the Spiros project). The results of this case study (published in (Paper B)) pointed at the guidance on software tool qualification found in modern safety standards as defining much of the relationship between tool integration and functional safety, but also at limitations in the approach of using safety standards to explore this relationship. The limitations were primarily due to the results not being immediately usable outside of the domain specified by the safety standard in question. The results of this iteration have, in regard to this thesis, primarily influenced the State of the Art description in section 2.3.

3. The third iteration was aimed at overcoming the limitations of the approach used in the second iteration. This was achieved by use of the System-Theoretic Process Analysis (STPA) hazard analysis technique. STPA allowed for the use of safety-guided design to both analyse the tool chain from the first iteration and update it twice to mitigate the risks associated with tool integration that
were identified. The details of this case study and analysis were published in [24], while the wider implications of the analysis on STPA itself were published in (Paper D). The risks identified for the three different versions of the tool chain defined a relationship between safety and tool integration through the properties of tool chains that were needed to mitigate these risks. These properties and a proposition on how to interpret their implications on safety certification were published in (Paper C). The theory that the analysis of this iteration is based on is presented in chapter 3. The steps necessary to adhere this theory to the context of tool integration is presented in chapter 4. Chapter 5 presents the relevant details of the case study. The discussion in chapter 6 then deals with the implications of the research related to the case study, including the proposition on how to interpret the implications of the results on safety certification published in (Paper C).

This thesis presents the findings at a level of detail that was not possible in the attached papers due to space limitations. This allows for a more thorough definition of the relationship between tool integration in development environments and the safety of embedded systems developed by use of said environments.

1.6 Thesis Outline

This thesis is organized as follows:

- Chapter 1 presents the background, definition and limitation of the problem that is discussed in the thesis. The definition of the problem is described both at a high level and at a detailed level, the latter as a set of research questions to be answered by the thesis. Additionally this chapter positions all the chapters of this thesis by defining their relationship to the research publications that this thesis is based on and their relationship to each other during reading.

- Chapter 2 presents the State of the Art of software engineering with implications on tool integration, tool integration as a research field on its own and system safety standards within the embedded systems domain.

- Chapter 3 presents Systems Thinking and some important concepts within this theory. The Systems-Theoretic Accident Model and Processes (STAMP) causality model and the associated STPA hazard analysis technique are presented in the same chapter, since these two are based on theory developed within Systems Thinking.

- Chapter 4 presents three concepts within the domain of tool integration. These concepts were developed during the research towards this thesis and

---

6Systems Thinking is a scientific approach based on repeatability, refutation and dealing with complex phenomena on the level of complete systems (i.e. not through reduction).
will become important to describe the relationship between tool integration and safety later on in this thesis. These three concepts are two models and a mapping:

– A new conceptual model of a development effort, based on Systems Thinking.

– A new reference model for tool integration, developed to be usable as a reference model for reasoning about tool integration in highly heterogeneous environments.

– A mapping of the integration aspects of the new reference model to a generic control loop\(^7\).

• Chapter 5 presents the results of a case study, based on the theory presented in earlier chapters, to investigate the risks that tool integration introduces to an embedded systems development effort. The most important results presented are:

  – The identification of three different types of control loops in a development effort that are relevant to tool integration as related to safety.

  – The identification of nine properties of tool chains that could mitigate at least part of the risks introduced by tool integration.

• Chapter 6 discusses the results of the case study presented in chapter 5 to propose two ways to support future versions of industrial safety standards for embedded systems. The first proposition is to apply the mapping from chapter 4 to three different types of control loops identified in chapter 5. This would allow the identification of how the metrics described in chapter 4 can provide guidance in regard to safety when moving from manual to automated tool integration. The second proposition is a new software tool qualification process based on translating the properties found in chapter 5 into safety goals. The primary reason for utilizing this new software tool qualification process is the possibility to deal with shortcomings of the system safety standards described in chapter 2 in regard to tool integration and safety.

• Chapter 7 wraps up the thesis by:

  – Summarizing the thesis.

  – Listing the the contributions found within the thesis:

\(^7\)A control loop is formed by (1) a controller, (2) a process that the controller is trying to control according to some goal, (3) the variables of the controlled process that the controller can manipulate and (4) the information on the current process state that the controller can obtain from measurable variables (feedback). If the controller is using feedback to control the controlled process the control loop is defined as closed, otherwise it is defined as open.
1.7 Chapter Summary

This chapter started with providing a background to how tool integration has benefited Systems Engineering, although the research into tool integration has not been influenced to any large degree by the wider perspective set by this engineering discipline. The chapter then highlighted how the focus on technology within the research into tool integration has lead to a lack of methods and tools for analysing how non-functional properties of development efforts, development environments and end products are related to tool integration. The limitation of this problem into a part that is reasonable to handle in a Licentiate Thesis is defined and described as three research questions, which this thesis will attempt to answer:

- What is the relationship between tool integration in development environments and the safety of embedded systems developed by use of said environments?

- Safety standards based on Systems and Software Engineering principles define the boundaries for development efforts targeting safety-critical, embedded systems. Do these safety standards provide guidance on tool integration? If they do, what form does this guidance take and is it different in standards targeting different industries or between different versions of the same standards?

- What are the differences between manual and automated tool integration in relation to the safety of embedded systems? Is one to prefer over the other in the context of the safety of end products?

Additionally the approach of the research presented in this thesis is described, the research findings related to the different chapters of the thesis and a thesis outline given.
Chapter 2

State of the Art

Chapter 2 presents the State of the Art in Software and Systems Engineering that will be of value throughout the discussions in this thesis. This presentation is partitioned into three parts. The first details the prominent State of the Art of Software Engineering that has the potential to impact tool integration to a major degree. The second details the State of the Art of tool integration as a separate field of study. The third details the State of the Art of research into an area close to tool integration and safety, namely that of software tool qualification in the embedded domain. The third part is based on a review, performed during the preparation of this thesis, of several industrial safety standards dealing with the functional safety of embedded systems. This last part also includes a discussion on why the relationship between tool integration and safety has received so little attention over the years.

2.1 Tool Integration and Software Engineering

As mentioned in section 1.1 research into tool integration is tightly linked to Software Engineering in general. During a survey published in an internal iFEST project deliverable (see [22] for a description of the iFEST project) two areas that have seen large breakthroughs during the last decade and have the potential to impact tool integration to a large degree were identified as being of special interest. The first was Web-based Architectural Styles and the second Model-Based Development (MBD). This section presents the findings of this survey in regard to these two areas.

Three web-based architectural styles were identified as being of interest, namely Representational State Transfer (REST) ([25], [26], [27]), Service-oriented architecture (SOA) ([28], [29], [30]) and Thin Client (TC) ([31], [32]). These have all gained wide-spread use during the last decade. The trend towards this kind of architectures has moved industry and academia towards a greater consensus on common ways of accessing the control and data of tools (affecting what will be referred to as plat-
form integration and its relationship to control integration and data integration in section 4.2. This trend is, however, less likely to affect the form of control and data directly, since traditional SOA architectures approach this differently from REST [33] and TC does not address it at all. The large use of web services in the context of SOA and REST has also lead to an increased use of graphical representations for process modeling such as Business Process Modeling Notation (BPMN) [34] and languages such as Web Services Business Process Execution Language (WS-BPEL) [35]. This promises an easier way of handling automated control of tools across product life-cycles (what is referred to as process integration in section 4.2). REST alone promises to facilitate new ways of integrating the user interfaces between tools, since this architecture increases the separation between user interface concerns and data storage concerns [25] (affecting what is referred to as presentation integration in section 4.2).

MBD is gaining more attention, especially in the domain of embedded systems due to the ever increasing complexity of design, strictness of requirements, etc. [36]. While MBD is available to parts of the industry in all phases of the product life-cycle [37], a large research community is producing (1) frameworks for developing with models or integrating tools using models ([38], [39], [40], [41], [42], [43], [44]), (2) MBD tools or functionality supporting said tools ([45], [46], [47]) and (3) overviews of the area ([36], [37], [48]). MBD promises a more effective and correct transfer of data through the use of data transformations between different life-cycle phases and engineering disciplines [49]. This enables easier data sharing, data linkage and data interchange (affecting what is referred to as data integration in section 4.2).

In short, all aspects of tool integration have been affected in one way or another during the last decade by technological innovations in the field of Software Engineering. The introduction of web technology has made it possible to form more uniform and coherent links between tools within each development effort, but the technology itself is not aimed at achieving a greater consensus on the form of these links. The data that is transferred between different tools in similar projects can still be qualitatively different, the product life-cycle orchestrated in different ways, etc. MBD on the other hand, promises a uniform approach on how to handle the contents of these links, at least in regard to the data transferred.

2.2 Tool Integration as a Specialized Field of Study

The focus of tool integration research has largely been on describing or suggesting technical solutions [18]. Even when the focus have been on the broader picture, many details regarding now outdated technology often obscure the message that different authors try to convey (see for instance [17], [50] and [51]). Java programs implemented on top of a common platform are for instance often discussed to a great deal in older papers, an approach distinctly different from that of the web-based architectural styles mentioned in the previous section. This narrow view creates and reinforces the assumption in readers that the ultimate goal of tool integration
is simply to create one uniform development environment (see for instance [6], [52] and [53]).

It is therefore not so surprising that the common denominator between several currently running research projects that deal with tool integration is in fact an industrial technical standardization initiative named Open Services for Life-cycle Collaboration (OSLC) [54] (adopted by research projects such as Cesar [55], iFEST [22], MBAT [56], etc.). OSLC is focused on standardizing the way that tools throughout the product life-cycle can share data with one another. To that end OSLC publishes open and free to use specifications for standardizing data and the interaction to retrieve or manipulate that data. The standardization is primarily aimed at providing communication between tools and elects not to enforce any hard constraints on tools to adhere to specific versions of the specifications. The point of a system following such a philosophy is that it allows for asynchronous evolution.

Unfortunately, for methods such as MBD, which rely on clearly defined semantics and syntax between tools, this is not a viable solution. The necessary properties of a development environment capable of supporting MBD throughout the whole product life-cycle are however discussed in [19]. While the discussion in this paper is a bit simplified\(^1\), it touches upon the problems that make the realization of the described tool integration difficult. The main problem is described as being “political”.

> “Those who require an integrated development environment - mainly the tool users - do not feel responsible for driving the integration. Those who seem to be responsible about tooling - mainly the tool vendors - do not profit.” [19]

The current State of the Art can therefore be characterized as one of slow progress towards yet another common technological environment for reaching and manipulating tools and data. However, ways to realize more meaningful interactions on top of this common infrastructure are still to be put forward by stakeholders that are deeply divided on the subject of why this should be done and by whom. This means that a large number of urgent issues related to tool integration is currently not dealt with to any large extent. Some examples of such issues can be found in the research conducted within the iFEST project [22], for instance in regard to the possible generalizations of software tools into different types to simplify the standardization of communication, the elicitation of requirements of different engineering disciplines on tool integration and how to best use the standardization of technology to deal with high level requirements on tool integration.

\(^1\) [19] promotes some solutions without discussing their potential drawbacks. An example of this is the suggestion that a workflow engine should be used to prohibit deviations from a stipulated process. Process deviations exist for a reason and a too strict process control by a development environment may result in an abandoning of the development environment rather than the adherence to a process.
CHAPTER 2. STATE OF THE ART

2.3 Tool Integration and Safety

Given the slow progress in the field of tool integration in general, it is not surprising that there is not much progress in such a specialized area as the study of how safety relates tool integration. The scientific community has however spent a large effort on how safety relates to engineering tools, analysing them, categorizing them, discussing how to certify them, etc. [57]. When tool integration has been discussed it has been in the context of how to enable reliable communication between a complementing combination of safety-related engineering tools, while the safety implications of the tool integration itself have not been explicitly discussed (an example is [58]).

This is not surprising, since the primary source of pressure to look into these issues are industrial safety standards which focus on engineering tools instead of entire development environments. To illustrate why this circumstance has a negative impact on safety in regard to tool integration, a number of widely used safety standards within the domain of functional safety of embedded systems are discussed below. This discussion is based on a review and analysis of industrial safety standards done in preparation for this thesis. Both old and new standards are discussed to present a view of the development of the guidelines given within this domain. The list of standards discussed is, however, not exhaustive. The choice of restricting the number of standards that were analysed was done based on two reasons. The first was a practical one, it is simply not possible to analyse all industrial safety standards in existence due to the limited resources and time available. The second was the issue of similarity, i.e. to enable a fair comparison of industrial safety standards those considered must share roughly the same kind of objectives. The choice therefore fell on a group of industrial safety standards that all focus on the complete product life-cycle of embedded systems belonging to a specific domain, even if some of them only consider one aspect of these embedded systems.

Two decades ago, in 1992, DO-178B was published to “provide guidelines for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements” [59], i.e. to describe the processes needed to ensure safety throughout the product life-cycle when producing software for airborne systems. This standard also recognizes the impact engineering tools might have on such processes. Therefore, if any of the prescribed processes are eliminated, reduced or automated by use of a software tool and the output of said tool is not verified according to DO-178B, then said tool needs to be qualified². DO-178B is very specific in pointing out two categories of tools that are of interest for qualification, namely development and verification tools. It thereby excludes TIMs as primary targets for a qualification effort. However, the standard allows several tools to be qualified together and de-

²A qualification process is a potentially smaller subset of the activities required to certify a tool. These activities are all linked to a specific area of concern. In this case the qualification referred to is the process that ensures that the tool provides confidence at least equivalent to that of the process(es) eliminated, reduced or automated.
fines the operational environment of tools as an important factor to qualify tools against. In regard to the latter DO-178B stipulates that tools should be shown to execute safely even if the operational environment fails or operates in an abnormal way.

In 1998 IEC 61508 was published as a base standard which “sets out a generic approach for all safety life-cycle activities for systems comprised of electrical and/or electronic and/or programmable electronic components (electrical/electronic/programmable electronic systems (E/E/PESs)) that are used to perform safety functions” [60]. IEC 61508:1998 lists a few types of engineering tools, but mostly focuses on prescribing requirements for languages and coding standards. This standard also stresses the importance of planning which tools to use prior to the start of development and opens up for qualification by tool vendors. However, there is no guidance on whether TIMs are of importance or how the actual tool qualification procedure should be carried out.

There are many similarities between IEC 61508:1998 and EN 50128 [61], which was released in 2001 and “specifies procedures and technical requirements for the development of programmable electronic systems for use in railway control and protection applications”. Both standards stress the importance of planning which tools to use up front and recommends the use of certified3 tools. EN 50128 in addition stresses the importance of using the same set of tools across the whole product life-cycle and defines certified tools as having been determined to be of a particular quality. Most unfortunately however, EN 50128, like IEC 61508:1998, provide no guidance on whether TIMs are of importance or any specification of how the tool qualification procedure should be carried out.

In the next version of IEC 61508 [62], released in 2010, there is an increased awareness of the fact that tools are playing an increasing role in all the phases of the product life-cycle. IEC 61508:2010 views this as a positive development and in fact requires that appropriate tools should be chosen to reduce the “likelihood of introducing or not detecting faults during the development”. The integration of the chosen tools are also viewed as very positive, since it is assumed that if the output from one tool has a suitable content and format for automatic input to a subsequent tool then the possibility of introducing human error can be minimized. The dangers of this development are, however, also recognized. IEC 61508:2010 therefore requires the competence of users of the tools to be considered. IEC 61508:2010 is, as the standards previously discussed, not clear on how TIMs should be treated. The division into different classes of tools depending on how directly their output may affect the end product and the lack of guidelines for tool integration seems to indicate that TIMs should not be considered on their own during tool qualification.

In 2011 a domain adaptation of IEC 61508 for the automotive industry was released. This standard, ISO 26262 [16], aims “to comply with needs specific to the application sector of Electrical and/or Electronic (E/E) systems within road

---

3A certified tool has been confirmed, according to predefined rules, to exhibit a certain set of characteristics.
vehicles”. The differences between IEC 61508:2010 and ISO 26262 in regard to tool qualification are striking. The most noticeable difference is that ISO 26262 provides generic definitions for why tools should be evaluated for tool qualification. These definitions point out the importance of (1) knowing the possibility that tools can introduce or fail to detect errors in a safety-related item and (2) the confidence in preventing or detecting this from happening. Based on these definitions a detailed specification for a tool qualification process is laid out, a specification which takes into account both internal and external measures to the tools (the latter includes reviews, guidelines, etc.). The definition of a software tool\footnote{ISO 26262 defines a software tool as a “computer program used in the development of an item or element” \cite{iso26262}.} is sufficiently generic to imply an approach where TIMs are also included. Additionally, if this generic definition is ignored then the case for this approach is reinforced by the tool qualification process requiring the context of all analysed tools to be defined. The seriousness of ISO 26262 in regard to tool qualification is also stressed by requirements on the independence of the reviewer of the tool qualification efforts. Regardless of all this support for a tool qualification approach which includes TIMs, the discussion in regard to tool qualification as specified by ISO 26262 has shown a strong push towards limiting the tools (and TIMs) that need to be taken into account (Paper B). There exists some concerns regarding this whole issue, however, noticeable in the calls for some independent party to provide guidance on the correct level of qualification \cite{paperb}.

In 2011 a new version of DO-178 was released. This version, DO-178C \cite{do178c}, expands the guidance on tool qualification significantly by (1) dropping the two tool categories defined in DO-178B in favour for three more generic ones and (2) introducing DO-330 \cite{do330} (a separate document for software tool qualification considerations). Although the three new categories for defining which tools that could be the subject of tool qualification are more generic, they all exclude TIMs as primary targets for qualification. DO-330 goes further than the standards described earlier by defining part of the problem of tool development as being that it is often undertaken by “teams other than those who use the tools to develop software”. This means that DO-330 sets a wider context by for instance describing how the responsibilities during tool qualification should be divided between tool users and tool developers in regard to COTS tools. DO-330 is nevertheless still constrained by only being applicable to scenarios in which any of the prescribed processes are eliminated, reduced or automated by use of a software tool and the output of said tool is not verified according to DO-178C. Within these constraints a considerable number of issues are however discussed, such as how to handle external components (such as operating system primitives), abnormal conditions, supplier conformance to DO-330, COTS tools, etc. A wide number of requirements are made on reviews, analyses, tests, etc. This is especially true for COTS tools for which all anticipated tool operational environments, all input (including its structure), all output, all expected functional behaviour and how to handle all abnormal conditions needs to
be defined. On top of this DO-330 also allows alternative tool qualification methods, as long as these can be sufficiently motivated. Given the focus on exhaustive analysis and testing, it appears that DO-330 depends heavily on the limitations DO-178C puts on which tools that need to be qualified. Modern development environments can consist of several hundreds of tools and TIMs \cite{66}, making it infeasible to perform exhaustive testing on all possible combinations of interactions.

All of the standards mentioned in this section are influential standards that are likely to affect both how industry and academia approach the question of what is required by a development environment to ensure safety.

The approach of DO-178C and IEC 61508:2010 is to draw a line between engineering tools that should be qualified and the rest of the development environment that can be excluded as a primary target for a qualification effort. As seen in the discussion above, this line may not be explicitly spelled out in the guidelines. However, there will be strong economical reasons for disregarding efforts that are implicit or linked to issues that are not properly discussed, thereby ensuring the likelihood that this line will in fact be established. As discussed in (Paper C), this approach will encounter problems with ensuring safety when development environments scale up and become more automated, since important parts for ensuring safety are likely to be ignored during software tool qualification. In fact, even when these standards make it probable that TIMs will be covered during a qualification effort, the focus on faulty behaviour of engineering tools or TIMs will obscure the cases when a risk\footnote{Note that the term risk in this thesis also refers to indirect risks, for instance when actions carried out during development may eventually lead to a hazard in the end product. This is different from the common use of the term risk in safety standards. In these the term risk is often used only to indicate product risks, i.e. the risks that are inherent to end products and direct in nature (an example of a product risk would be the failure of the lock on the baggage door to an aircraft leading to the baggage door opening in mid-flight).} is not caused by any software malfunction\footnote{A risk that is not necessarily caused by software malfunction is for instance that of conflicting commands. An incomplete tool integration may for instance result in two software tools that are both correct in issuing a certain command, while the end result may be that these conflict in a hazardous way.}.

ISO 26262 does not draw this line, but instead tries to establish generic, one size fits all guidelines for how to qualify software tools. The discussion on how to apply this standard has subsequently focused on how to limit the tools (and TIMs) that need to be taken into account. This is not surprising, since qualifying a whole modern development environment by treating all of its parts in the same way is likely to be extremely costly. The most likely outcome of such an approach if applied to a large, modern development environment is rather sweeping generalizations of the danger posed by different parts of the development environment (Paper C).

Engineering tools have become the major focus in both of these approaches, since the increase in size and automation of development environments is seen as simply adding more of something (software) with essentially the same basic properties. The first approach then needs something central to tie the limitation of the qualification effort to, while the second approach is likely to be more influenced
by the “core” tools then the “peripheral” TIMs when the “generic” guidelines are to be established.

This focus on engineering tools has lead to a weak focus on the implications that tool integration has on safety.

2.4 Chapter Summary

This chapter highlighted web-based architectural styles, especially REST and SOA, and MBD as important technological innovations with implications for tool integration that have seen increased use during the last decade. The overall progress of tool integration is, however, found to be slow and mostly focused on creating a common technological environment for reaching and manipulating tools and data. Suggestions for how to create meaningful, and common, interactions on top of this common infrastructure are still to be put forward by the research community.

This chapter also identified industrial safety standards as the primary motivation for research into the relationship between tool integration and safety, since they contain guidelines on software tool qualification. Analysis of a number of widely used safety standards within the domain of functional safety of embedded systems however point at the implications of this relationship as being ignored. These standards instead focus on engineering tools during qualification, either as something of central importance to tie a limitation of the qualification effort to or something of central importance to inspire generic, one size fits all guidelines. Not only do these two approaches have problems with ensuring safety when development environments scale up and become more automated, they have also lead to a weak focus on the implications that tool integration have on safety.
Chapter 3

Systems Thinking, System Safety

Chapter 3 is, like the previous chapter, concerned with the State of the Art. This chapter, however, deals with theory that has not yet been applied to the domain of tool integration. As such, this chapter represents the first step towards a new approach, a systems approach, to analysing the relationship between tool integration and safety. To this effect some of the most important ideas in the school of thought known as Systems Thinking are presented, followed by a causality model and a hazard analysis technique that both build on these ideas to handle the new challenges of modern systems in regard to safety.

3.1 The Origins of Systems Thinking

As Checkland points out in [67], a scientific approach based on reductionism, repeatability and refutation revolutionized the world of science during the 17th century. This approach did not only provide a highly successful way of searching for new knowledge, but also rapidly changed the way humanity viewed the world. The profound impact of this approach on not only the scientific community, but the entire western civilization, has lead to a deeply ingrained, almost universal acceptance of this approach as the method of conducting science.

Yet this approach fails when research encounters phenomena that cannot be reduced, when the object of study is so complex as to make it impossible to dismantle it into parts that can be studied in isolation. A scientific approach for studying complex phenomena, while still requiring repeatability and refutation, would therefore have to provide methods for approaching an object of study as a whole.

The starting point for an organized theory on this issue was the writings of biologists in the early 20th century [67]. While the different parts of organisms can be described using concepts from chemistry and physics, there was still a need to discuss properties of organisms as wholes in a way which could not be done using concepts from these fundamental disciplines. The terms needed would essentially be meaningless when related to chemistry or physics. The theory was eventually
CHAPTER 3. SYSTEMS THINKING, SYSTEM SAFETY

generalized into General Systems Theory (GST) in the 1940s by Ludwig von Bertalanffy. However, like the notion of such a theory is more or less apparent in the writings of many other scientists, this was only one part of a new school of thought emerging throughout the first half of the 20th century. The approach of this school, which is called Systems Thinking, can be best defined as one of approaching problems by considering the whole rather than any specific part, input, output, event, etc.

Within Systems Thinking, a number of ideas have gained widespread traction. Checkland lists four of these as especially important, grouped into two pairs, namely hierarchy and emergence, and communication and control [67].

3.2 Hierarchy and Emergence

Hierarchy theory, as a specific flavour of GST, was introduced by Simon in [68]. Here he puts forward the idea that complex system frequently takes the form of a hierarchy, and that hierarchic systems have some common properties that are independent of their specific content.

This idea has subsequently been refined [3] to contain the following terms:

- A hierarchy consists of a number of hierarchical levels. All levels contain entities whose properties characterize the level in question and the entities of each level depend on the criteria used to link the levels with each other.

- An ordering of levels exists, based on criteria such as context, containment, etc. Of specific importance to this thesis are orderings based on constraints, i.e. in which an upper level constrain the possibilities of a lower level. Checkland uses DNA and base-sequences as an example of an ordering based on constraints [67]. The genetic coding constrains the possible ways that base-sequences may interact chemically with each other, therefore creating an upper level consisting of the genetic coding and a lower level consisting of the base-sequences. The lower level, in contrast to the constraints enforced by the upper level, establish that which is possible at higher levels. In analogue with the previous example, humans have five digits due to the constraints of our genetic code, but if it was (chemically) impossible for base-sequences to order themselves to encode this then we could not have five digits.

- A level of organization is a specific type of level, whose place in a hierarchy is defined solely based on definitions that relate the level to those above and below. An example is the way a population level can be defined as being above an organism level.

Based on the terms described above one may for instance envision a description of a civilization based on four levels of organization, the society, the community, the family and the individual. The entities on each level are easily identifiable as parts when viewed from an upper level and as self-contained wholes when viewed from
COMMUNICATION AND CONTROL

3.3

a lower level (they are holons, simultaneously a whole and a part [4]). Checkland describes several problems associated with analysing such a system [67], such as imprecise generalizations, predictions changing the systems they are made for, the active participation of the individuals under study, etc. It is however not these problems per se that invalidates the idea of reductionism. This is instead done by the acceptance of the notion of emergence.

Bertalanffy gives a simple definition of emergence:

“The meaning of the somewhat mystical expression, “the whole is more than the sum of parts” is simply that constitutive characteristics are not explainable from the characteristics of isolated parts. The characteristics of the complex, therefore, compared to those of the elements, appear as “new” or “emergent.” If, however, we know the total of parts contained in a system and the relations between them, the behavior of the system may be derived from the behavior of the parts.” [69]

The basic definition of an emergent property is therefore that it depends not only on the isolated parts of lower levels, but also on their interactions. Checkland relates emergence to the terminology of hierarchy by stating that in a general model of organized complexity, each level is characterized by emergent properties which are meaningless when using the concepts of lower levels [67]. In the example above, where DNA is viewed as the embodiment of two levels of organization, genetic coding is an emergent property at the upper level (biology). This is due to the fact that genetic coding does not only depend on the isolated nucleobases at the lower level (chemistry), but also on their relationships. Additionally, scientists concerned with the lower level (chemists) have no use for the concept of genetic coding when investigating the particulars of their domain (chemistry). For scientists concerned with the upper level (biologists), however, the concept of genetic coding is critical to handling complexity.

3.3 Communication and Control

If one is concerned with truly static systems or models of systems, the theory described in the two previous sections should be sufficient. This is rarely the case, since most (if not all non-abstract) systems are open systems that maintain their form based on continuous exchanges with their environment [69]. If this exchange is not to break down there must be some measure of control within the system. One example of control has already been discussed in the previous section, i.e. the way upper levels control lower levels by enforcing constraints on the possibilities that can be realized there. Checkland makes one further important observation regarding this hierarchical control in that the emergent properties that appear at the upper levels do so due to this enforcement of constraints [67], since they limit the possibilities of what can be. Like genetic coding (using the example from the
previous section) can exist because the number of possible sequences of nucleobases can be constrained.

*Communication*, the flow of information (feedback for instance), is important, since stable control mechanisms will depend on it. This issue can be discussed at length at an abstract level, but this would be of less importance to this thesis. For the application of Systems Thinking within the area of safety feedback is however critical, as shall be seen in the subsequent sections.

### 3.4 Systems-Theoretic Accident Model and Processes

In [5] Leveson challenges whether some of the underlying assumptions of many *State of the Practice* approaches to ensuring safety are still (if they ever were) valid. Examples include the assumptions that accidents are caused by chains of directly related events and that these events occur simultaneously by chance to produce most accidents. This challenging of common assumptions lead to the conclusion that a systems approach is needed when performing safety-related system analysis.

This idea has been suggested earlier, the nucleus of several of Leveson’s ideas are for instance mentioned by Leplat in [70]. Leveson, however, provides an exhaustive reference model for a system safety approach in the proposition of the *Systems-Theoretic Accident Model and Processes* (STAMP) causality model. STAMP is based on three concepts: safety constraints\(^1\), hierarchical safety control structures\(^2\) and process models\(^3\) [5].

By defining safety constraints as the most important entity when analysing accidents, Leveson moves the focus from isolated events to the question of what allows an accident to occur. When safety is compromised it is not the triggering event that is relevant, usually another random event would eventually have acted as a trigger if that particular one had not occurred, but the incomplete or insufficient control of the occurrences within the system. A hazard is therefore defined as a *system state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object), will lead inevitably to an accident (loss event)*. For instance, instead of disassembling the actions of an individual operator that eventually triggered an accident, one has to define the hierarchical safety control structures in which this operator acted. Did the operators manager (the upper level) provide unsafe instructions to the operators (the lower level)? And what, in turn, allowed the manager to act in this way (perhaps the upper level in that case, i.e. the manager’s manager, allowed safety constraint violations due to budget cuts)? This systems approach also highlights the importance of commu-

---

1A safety constraint is a constraint as defined by hierarchy theory, but concerned with enforcing safety.

2A hierarchical safety control structure is a system described as an ordering of different levels of organization, with all system components relevant to safety and their interactions included.

3The process model concept comes from control theory and denotes the model of the controlled process that every controller has to have to be able to exert any sort of meaningful control.
nication, since the downward control of safety constraints must be matched with upward feedback regarding to which degree the safety constraints are enforced.

The focus on process models in STAMP is tied to the view that safety constraints and hierarchical control structures allows a system to be seen as a number of controlled processes. The control of the processes is performed through the safety constraints enforced downwards, feedback is received by information sent upwards and the control decisions are taken based on the process models of the controllers (which can be automated or manual). This ensures that STAMP incorporates what is a common source of accidents, namely that there is a difference between the behaviour of what is controlled and how the controller believes that it will behave.

Figure 3.1 provides an example of the three central concepts of STAMP by presenting a simplified hierarchical safety control structure for practicing clay pigeon shooting. The different entities of interest belong to three different levels of organization, i.e., (1) a control level consisting of law enforcement, (2) an operator level consisting of the shooter, the bystanders and the safety officer and (3) a physical level consisting of the different tools required to practice clay pigeon shooting. Safety constraints are enforced downwards, both between and inside the different levels, through the control actions performed by the different entities. Through the feedback received from the controlled entities closed control loops are formed throughout the system. Each entity that enforces control also contains a process model, which is used when the control actions to be invoked are determined.

3.5 System-Theoretic Process Analysis

Leveson then continues by proposing the System-Theoretic Process Analysis (STPA) \cite{5}, a hazard analysis technique in two steps that can be iterated to increase the safety of a system step by step. This hazard analysis technique provides a systems approach to accident analysis based on STAMP.

System-Theoretic Process Analysis, Step 1

The intention of step 1 of STPA is to “identify the potential for inadequate control of the system that could lead to a hazardous state” \cite{5}.

To initiate this step one first defines which hazards should be analysed and then defines the control structure\(^4\) that describes the control of these hazards in the system to be analysed. The control structure is a logical structure that should include the components of interest in the system, their process models and the control actions\(^5\) they can enforce on each other. If one later realizes that parts are missing or incomplete, the control structure can be refined throughout the iterations of STPA.

\(^4\)A control structure can consist solely of a hierarchical safety control structure, but it can also include the interaction between components at the same level of organization.

\(^5\)The control actions are those component interactions that are intended to exert control.
The control actions are then analysed to identify basic and additional risks that could lead to hazardous states. The four basic risks that describe the possibilities for inadequate control are that (1) a control action required for safety is not provided or not followed, (2) an unsafe control action is provided, (3) a potentially safe control action is provided too early or too late and (4) a control action required for safety is stopped too soon or applied too long [5]. Additional risks are context-specific risks that can be defined and used when the basic risks are not sufficient. Additional risks for organizational contexts are for instance coordination risks, i.e. that (1) uncertainty regarding which controller should perform a control responsibility leads to a control action not being performed or (2) conflicting control actions from different controllers have unintended side effects [5].

If the system analysed is complex, one may additionally have to translate the relevant basic and additional risks to general types of risk. These are interpretations of the basic and additional risks based on a specific context and a specific hazard. An example, in an organizational context and in regard to the hazard of unsafe decision making, is the translation of the basic risk that “an unsafe control action is provided” to the risk that “unsafe decisions are made or approved”.

The output from this step 1 are the programmatic risks, i.e. the specific unsafe controls (in a specific context and safety control structure) that can lead to the relevant basic, additional or general types of risk.

Figure 3.1: An Example that Visualizes the Three Central Concepts of STAMP.
3.5. **SYSTEM-THEORETIC PROCESS ANALYSIS**

27

Figure 3.2: The Causal Factors to be Considered to Identify Causes in Step 2 [5].

**System-Theoretic Process Analysis, Step 2**

The intention of step 2 of STPA is to “determine how each potentially hazardous control action identified in step 1 could occur” [5].

The safety requirements for the components of interest can be deduced based on the programmatic risks identified in step 1. In step 2 one uses guidance provided by STPA to go further and try to identify the potential causes to these risks by examining the control loops associated with them. This guidance is in the form of **general causal factors** associated with risks in control loops. These causal factors are shown in Figure [3.2](#).

Once all the hazardous control actions in the system have been analysed in regard to the control loops they appear in, the analyst should have an exhaustive list of causes to the risks in the system at hand. The improbable causes can then be removed from the list and the system checked to see whether it handles the remaining causes correctly. Thereafter only one part remains, to analyse the special case that controls may degrade over time. This includes stipulating performance audits, change management, etc. After this the analysis can end or the steps be reiterated to further refine the analysis or ensure that unhandled causes are
mitigated by updates to the safety control structure.

3.6 Chapter Summary

This chapter started with describing the school of thought known as Systems Thinking, a scientific approach that is useful when dealing with complex systems that cannot be reduced into isolated parts without losing information necessary for analysis. Four important ideas within this theory were then discussed:

- **Hierarchy:** The idea that complex systems can be partitioned into a number of hierarchical levels, based on the relationship between entities at the different levels in regard to context, containment, etc. The most important levels in regard to the discussion in this thesis are those whose place in a hierarchy is defined solely based on definitions (i.e. levels of organization).

- **Emergence:** The idea that some properties at different hierarchical levels cannot be decomposed into isolated parts at lower levels, since they depend on the interaction between the parts at those lower levels. In regard to hierarchy theory, each hierarchical level will be characterized by emergent properties which are meaningless when using the concepts of lower levels.

- **Control:** Open systems need some measure of control within the system to avoid breaking down. In regard to hierarchy and emergence, the emergent properties at each level will depend on enforcement of constraints (control) from upper levels to limit the possibilities given by the lower levels.

- **Communication:** The observation that stable control mechanisms will depend on the flow of information (communication).

Then the STAMP causality model, whose base it built on top of these four ideas, was introduced. This model introduces the notion of safety constraints, hierarchical safety control structures and process models to allow a system to be viewed as a number of controlled processes. The control of the processes is performed through the safety constraints enforced downwards, feedback is received by information sent upwards and the control decisions are taken based on the process models of the controllers.

Finally a hazard analysis technique which is based on the STAMP causality model was introduced. This technique, called STPA, is divided into two steps:

1. The first step of STPA is meant to identify the potential for inadequate control of the system at hand that could lead to a hazardous state. This is done by constructing a safety control structure of the system and then analysing this structure for risks by using predefined risks as guidance (either generic risks or risks defined for the specific context of the system at hand).
2. The second step of STPA is meant to determine how each potentially hazardous control action identified in step 1 could occur. This is achieved by analysing the safety control structure, viewed as a number of control loops, for causes to the identified risks by using general causal factors as guidance. In addition, the special case that controls may degrade over time is considered in this step.

After these two steps the findings of the analysis can be used to update the control structure, after which the steps can be reiterated to ensure safety to an even higher degree.
Chapter 4

Tool Integration Concepts and Models

Chapter 4 presents three concepts within the domain of tool integration, which later on in the thesis will be important to describe the relationship between tool integration and safety. The first is a new conceptual model of a development effort. The second is a new reference model for tool integration developed for analysis of highly heterogeneous environments, i.e. for analysis of for instance embedded systems development efforts. The third is a mapping of the integration aspects of the new reference model to a generic control loop.

These concepts are to a large extent constructed using the theory and terminology described in the previous chapter.

4.1 A New Conceptual Model of an Embedded Systems Development Effort

The conceptual model presented in this section was first published in (Paper C). It was based on Systems Thinking because a model that reduces an embedded systems development effort into all the involved parts and their direct relationships would be both difficult to comprehend and to handle. By using the concepts from Systems Thinking described in chapter 3 the construction of such a model can however be greatly simplified. To construct a model using these concepts one can start by taking a wide perspective on the context at hand and then try to identify different constraints between entities while narrowing the perspective. One way of doing this in the context of a development effort is to (1) start with the entities which constrain the basic conditions and actions of the involved operators, (2) identify that these operators through their processes constrain the development and therefore the tool chains used, (3) note that the tool chains define the interactions between different parts of the development environment and (4) end with the development environment as a collection of different tools. Based on this chain of entities that
in turn constrain each other a model with four levels, namely the *management*, *operator*, *tool chain* and *tool levels* (see Figure 4.1), was constructed. This conceptual model is constructed using the theory described in the previous chapter and as such describes the organization of the context at hand as a hierarchical system. However, the model will also be used in this thesis to reason about the run-time actions and feedback taking place within the context at hand.

![Figure 4.1: Conceptual Model of a Embedded Systems Development Effort.](image)

At this point it is important to note that the construction of this conceptual model was entirely dependent on the use to which it was going to be put. When analysing tool integration at this level of abstraction there was (yet) no need to for instance discuss differences between organizational roles (such as differences between mechanical, electrical and software engineers) or which process is a supporting one (as opposed to the main development process). There are several examples of similar conceptual models that due to the purpose behind their construction are for instance more detailed. An example of this is the model of the socio-technical system involved in risk management defined by Rasmussen in [71], which further divides the management and operator levels and does not contain any tool chain or tool levels.

**The Management Level**

The top level of the conceptual model consists of the management of the development effort. To belong to this level a part of a development effort must exert some type of control (legal, social, etc.) over the processes needed during the product
life-cycle. The constraints on lower levels would be enforced through oversight or insight, while the feedback could come in the form of reports. Parts found at this level include certification agencies, industrial standards, high level management, advisers, trade unions, governments, etc.

The Operator Level

The second level consists of the separate operators that implement any of the processes needed during the product life-cycle phases. This is not limited only to the development of the final end product, but also to those operators that act within earlier or later phases (such as operators constructing tools or performing maintenance). To belong to this level an operator must exert some type of control over which processes are actually implemented (as opposed to planned), since this will define which tool chains are used. The constraints on lower levels would be enforced through the actual process implemented, while the feedback would come in the form of the resulting process state. Tool developers and tool chain developers belong to this level, as well as different types of engineers, managers involved in the day-to-day operations, etc.

The Tool Chain Level

The tool chain level consists of the tool chains used during the product life-cycle phases. It imposes constraints on the next level, the tool level, by specifying the order of the engineering tools, stipulating which tools or TIMs that may communicate with each other and how this communication can be carried out. The feedback are the actual islands of automation \[7] that are created by the different tool chains.

The Tool Level

The tool level consists of the separate engineering tools and TIMs.

4.2 A New Reference Model for Tool Integration

To augment this conceptual model of an embedded systems development effort a description of tool integration per se is needed. Such a description can be found in the tool integration reference model presented in (Paper A). Below, however, a modified and more elaborate description of this model is presented, based on the critique received from internal\(^1\) and external\(^2\) reviewers. Before going into each integration aspect in the reference model in detail, an overview of them and their relationships are presented in Figure 4.2.

\(^{1}\)Supervisors and fellow PhD Candidates.

\(^{2}\)Workshop and Conference reviewers.
CHAPTER 4. TOOL INTEGRATION CONCEPTS AND MODELS

Figure 4.2: A New Reference Model for Tool Integration.

Platform Integration

*Platform integration* is the degree to which tools share a common environment. In a sense that means that all levels of the conceptual model in section 4.1 are concerned with platform integration, since operators can be this common denominator. In this sense the notion of platform integration is also almost meaningless, albeit one could consider the cases when there is a point in only allowing tool integration through operators due to for instance security reasons.

However, if one limits oneself to the automation of tool integration (as this thesis does) then the tool level becomes the most important, since platform integration at that level will define the length of the possible automated tool chain parts. A metric for platform integration in relation to automated tool integration is then easy to define, it is the number of instances when parts of tool chains *need* to be joined by operators. This means that the metric for platform integration is the *de facto* number of separate platforms that are utilized in the system. An operating system can be viewed as a single platform even if many separate parts of a tool chain execute on top of it, if no additional common denominators need to be added to allow these tool chain parts to be joined into a single automated tool chain (part). In this way the metric for platform integration is an absolute\(^3\) metric. A higher number will always signal a weaker platform integration, since it will indicate more gaps between tools that cannot be bridged by software.

Operating systems have already been mentioned above as an example of entities related to platform integration. These are important to platform integration, since they usually define borders between which there are no common denominators in software. Operators are therefore forced to handle the interaction between

---

\(^3\)The choice of an absolute metric is preferred, since there is no pressing need to compare different development environments. The possibility to analyse the impact of changes to the tool integration of a particular development environment is however of direct value to several stakeholders within a development effort. This also means, however, that the metrics defined in this chapter are of limited value when comparing different development environments, since the effort to reach the same number can be very different for different development environments.
4.2. A NEW REFERENCE MODEL FOR TOOL INTEGRATION

tools executing on different operating systems. Today, with the new architectures mentioned in section 2.1 this may however be an oversimplification.

Control Integration

*Control integration* is the degree to which tools can issue commands and notifications correctly to one another. This means that control integration is dependent on platform integration for tools to be able to reach one another.

If only automated tool integration is considered important, then the metric for control integration can be defined as the number of tool integration services that have to be invoked by operators in their processes. By considering this number, instead of all services provided by entities at the tool level as in (Paper A), this metric also becomes an absolute one. A higher number will always signal a weaker control integration, since it will indicate more tool integration activities that cannot be initiated by software.

An example of an entity important to control integration is for instance a TIM that is invoked by an engineer to retrieve a development artifact.

Data Integration

*Data integration* is the degree to which tools can, in a meaningful way, access the complete data set available within the system of interest. Data integration is therefore dependent on both platform and control integration for tools to be able to reach one another’s data.

Data integration is the act of defining an unambiguous relationship between two semantically or syntactically different data sets. This can be performed manually or by automation, but in this thesis the focus is on when this definition is automated. The metric of data integration is the number of data interoperability sets in a system, i.e. the number of groups of syntactic and semantic pairs not related by links or transformations that can be found in the complete data set. A higher number will always signal a weaker data integration, since it will indicate more arbitrariness in the de facto way of relating different groups of syntactic and semantic pairs to each other.

An example of an entity important to data integration is a data transformation engine that can translate between data interoperability sets understandable by engineers from different engineering disciplines (for instance between VHDL and C Code).

Process Integration

*Process integration* is the degree to which interaction with tools can be supervised. This means that process integration is dependent on both control and data integration, since process control is achieved through commands, process awareness is propagated by notifications and process state is defined by data. This is important
to note to fully understand the difference between control integration and process integration, i.e. it is important to note that the former can occur without a change in process state.

If process integration is automated, it means that the interpretation of process state will have to move to an entity at the tool level. This means that there must exist an entity at that level that includes a process model. The metric of process integration can therefore be made clearer than the definition which was provided in (Paper A) if limited to the automation of tool integration. In that case this metric can be defined as the number of process states that need to be handled by operators. In this way the metric becomes absolute. A higher number will always signal a weaker process integration, since it will indicate a larger need for operators to monitor and control the development process.

An example of an entity important to process integration is a process engine that triggers certain scripts when the input data signal that a certain process state has been reached.

**Presentation Integration**

*Presentation integration* is the degree to which tools, in different contexts, can set up user interaction so that the tool input/output is perceived correctly. Presentation integration is dependent on the platform, control and data integration aspects mentioned earlier, since they limit the ways of presenting input/output to users.

Presentation integration is only meaningful when the subjective views of operators are considered and is therefore difficult to define. The initial metric was defined as “the level to which users are forced to view or manipulate data in views they are not used to interact with” (Paper A). This is subjective, but it points out the fact that presentation integration is not meaningful if defined as how uniform different graphical interfaces are. Instead it should define how well integrated the tools are in regard to the context they are used in (including operators). If only the automation of tool integration is considered, then the case when part of the interaction between operator and tool is automated is what needs to be analysed (since if it is all automated, one ends up with process integration). In this case the metric of presentation integration can be defined as the number of states of automated process models that can be invoked by an operator without that operator sharing related states in his own process model. For instance, if an operator invokes a data transformation to provide data to another operator without realizing that this invocation will also trigger a notification to a project manager, then this should metric should be increased by one. This metric is still subjective and will require analysis, but it defines what “not used to interact with” is and it is absolute. A higher number will always signal a weaker presentation integration, since it will indicate a greater uncertainty in regard to what the tools actually do.

An example of an entity important to presentation integration is a graphical toolbox, since it would limit the ways a user can perceive the engineering tools and TIMs.
4.3  A Mapping of the Integration Aspects

The last concept within the domain of tool integration that is required for the discussion in later chapters is a mapping of the integration aspects presented in the previous section to a generic control loop. This is required, since the approach in subsequent chapters is based on viewing an embedded systems development effort as a number of controlled processes. The motivation for allocating the integration aspects to the different parts of a generic control loop (see Figure 4.3 for the final allocation) is as follows:

- **Platform Integration:** Platform integration is concerned with common denominators between tools. This integration aspect is therefore mapped to all parts of the generic control loop, since they will all be related to common denominators.

- **Control Integration:** Control integration is concerned with the issuing of commands and reception of notifications. This integration aspect is therefore mapped to the links between the different entities in the generic control loop, since these links symbolize the actual activities of control and feedback.

- **Data Integration:** Data integration is concerned with the data, the information, passed around the system. This integration aspect is therefore mapped to the actuators and sensors, since the information passed around an embedded development effort is the actual instrument to achieve control and feedback. To make something happen or inform someone about something, information needs to be put in design documents, reports, configuration files, etc.

- **Process and Presentation Integration:** Process and presentation integration is concerned with supervising tools. The former when this is done in an automated fashion and the latter when it is done manually. To achieve this supervision there must be a controller, control input to this controller and a process model of the controller. These parts of a general control loop are therefore related to both of these integration aspects.

4.4  Chapter Summary

This chapter presented three concepts within the domain of tool integration, which later on in the thesis will be important to describe the relationship between tool integration and safety. These are:

- A new conceptual model of a development effort. This model, based on Systems Thinking, consists of four levels of organization. From top to bottom they are:
The management level. This level consists of the entities concerned with the management of the development effort. These entities constrain the basic conditions and actions of the involved operators.

The operator level. This level consists of the separate operators that implement any of the processes needed during the product life-cycle. These operators constrain the development activities through the processes they work with and provide feedback to the management level through for instance reports.

The tool chain level. This level consists of the tool chains used during the product life-cycle phases. These tool chains define (and therefore constrain) the interactions between the different parts of the development environment and provide feedback to the operator level through process states.

The tool level. This level consists of the separate engineering tools and TIMs. These provide feedback to the tool chain level through the formation of islands of automation based on the properties of the entities included in the tool chains.

A new reference model for tool integration developed for analysis of highly heterogeneous environments, i.e. for analysis of for instance embedded systems development efforts. This reference model defines five tool integration aspects, namely platform, control, data, process and presentation integration. The reference model also provides metrics for and defines the relationships between these tool integration aspects.
– **Platform Integration** is the degree to which tools share a common environment.

– **Control Integration** is the degree to which tools can issue commands and notifications correctly to one another.

– **Data Integration** is the degree to which tools can, in a meaningful way, access the complete data set available within the system of interest.

– **Process Integration** is the degree to which the interaction with tools can be supervised.

– **Presentation Integration** is the degree to which tools, in different contexts, can set up user interaction so that the tool input/output is perceived correctly.

• A mapping of the integration aspects of the new reference model to a generic control loop.

  – Platform integration is mapped to all parts of the generic control loop.
  
  – Control integration is mapped to the links between the different entities in the generic control loop.
  
  – Data integration is mapped to the actuators and sensors.
  
  – Process and presentation integration is mapped to the controller, control input to this controller and the process model of the controller.
Chapter 5

Tool Integration and Safety

Chapter 5 presents the results of a case study that investigated the risks that tool integration introduces to an embedded systems development effort, risks that may lead to safety violations in the end products associated with these development efforts. What is considered is the larger system that includes the development effort, as well as the end product that is the output of this development effort. In this way the hazards that are considered are system states of this greater system (not only system states of the end product) and the risks that are considered may be indirectly related to the actual triggering event of an accident. An example of such a risk is when erroneous data is provided to a Go/No Go launch decision by a tool developed under the wrong assumptions, thereby triggering an accident by causing a premature launch. The hypothesis is that the implications of the risks introduced by tool integration in regard to safety will define the relationship between tool integration and safety.

5.1 Analysing an Industrial Tool Chain using STPA

The case study that (Paper C) and (Paper D) is based on is presented in [24]. This case study was an analysis of an industrial tool chain using STPA. To conduct this case study STPA had to be extended, or elaborated with information, to enable an analysis in the context of tool integration. These extensions (or elaborations) are what is mostly described below, since they provide a background to the results and required the largest effort to finish during the case study. However, in the context of the discussion of this thesis these extensions are of less importance and are therefore not discussed in detail. For the other parts of the case study, related to simply “going through the motions” of applying STPA to the industrial tool chain at hand, the reader is also referred to [24] due to space limitations.

The analysis was conducted by:

1. Defining the general types of risk related to the context of tool integration.
2. Constructing a safety control structure and identifying the programmatic risks.

3. Identifying the types of control loops relevant to tool integration and identifying the causes to the programmatic risks related to these control loops.

4. Introducing changes to the control structure, reiterating the STPA analysis and analysing the results.

These steps are described in the subsequent subsections.

**Step 1 - Identify the General Types of Risk Associated with Tool integration**

The first step was to interpret the basic coordination risks mentioned in section 3.5 into general types of risk for each of the tool integration aspects and their relationships that are part of the reference model described in section 4.2. This is exemplified in Table 5.1 by the use of a tool integration aspect, two of its relationships to other tool integration aspects and two basic risks.

This interpretation identified 46 general types of risk, since not all combinations yielded an actual risk. However, many of these general types of risk were in fact the same kind of risk, only attributed to a different part of software development environments. For instance, errors in or missing parts of software could create the same kind of risk of data corruption, no matter if the missing part was concerned with platform, control or data integration (see the last column in Table 5.1). Therefore, these 46 general types of risk could be merged into 13 equivalent general types of risk. Based on these 13 general types of risk it is possible to make two observations:

1. The relationships between the different tool integration aspects imply that support by several tool integration aspects is needed to avoid certain general types of risk.

2. 8 of these general types of risk are relevant during the construction of a tool chain, 4 are relevant when a tool chain is used and 1 of them is relevant in both of these cases.

**Step 2 - Construction and Analysis of a Control Structure**

The next step was to define a control structure for the industrial tool chain at hand. The control structure for the original version of the industrial tool chain is shown in Figure 5.1. This picture is shown in a more readable form in [24], while it is used here to highlight the two parts that the control structure was made up of. These two parts were a process description in the form of a SysML Activity Diagram [73] and a tool integration aspect description based on the graphical format described in (Paper A). By going through each element of the original control structure for each
### Table 5.1: Examples of General Types of Risk for the New Reference Model.

<table>
<thead>
<tr>
<th>A control action required for safety is not provided or not followed.</th>
<th>An unsafe control action is provided.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Integration</strong></td>
<td>Data provided by a tool cannot be interpreted or is interpreted erroneously by another tool, since there is no possible, available or correct data transformation.</td>
</tr>
</tbody>
</table>

| **Data to Control Integration Relationship** | No service to request/update relevant data is provided by a tool, and data is corrupted during manual transfer. | Corrupt data is provided/created, due to an unqualified service, erroneous Control Integration Software (CIS) or missing CIS (manual transfer). |

| **Data to Platform Integration Relationship** | No common link exists to request/update data, and data is corrupted during manual transfer. | Corrupt data is provided/created, due to a unqualified platform service, erroneous Platform Integration Software (PIS) or missing PIS (manual transfer). |

of the 13 general types of risk it was possible to identify 85 programmatic risks in the original version of the industrial tool chain.

For instance, using the general type of risk used as an example in the previous subsection (i.e. that safety-critical data could be corrupted due to errors in or missing parts of the software concerned with platform, control or data integration), it was possible to identify 12 programmatic risks associated with different development artifacts used in the tool chain at hand.

### Step 3 - Identify the Causes to the Programmatic Risks

The third step was to identify the cause to the programmatic risks, using the guidance provided by STPA in the form of general causal factors. To achieve this identification the different control loops applicable to the context of tool integration in an embedded systems development effort needed to be defined.
The first two types of control loops that were of interest were implicitly identified, without much thought. The first type was concerned with the interaction between operators at the operator level, i.e. the interactions found in the process description part of the control structure. An example of this first type is the interaction in the original control structure where a FPGA Application Engineer reviews the artifact known as the “FPGA IO List” that is created by a Matlab/Simulink Application Designer. Both the Controller and the Controlled Process are operators as shown in Figure 5.2. The second type was not present in the original control structure, but concerns control loops in which an engineering tool or TIM controls (an)other engineering tool(s) or TIM(s). In relation to the conceptual model described in section 4.2 these two types are both contained within one level of organization and are therefore trivial to define.

The types of control loops associated with the constraints between the levels of organization found in the conceptual model was harder to define. The type of control loops associated with the constraints between the operator and tool chain level was however defined and identified as relevant. This type of control loops could be defined as one in which the operators (the controllers) can introduce hazards in the end product (the end result of the controlled process) through processes that update development artifacts (the actuators), hazards which can be identified through the process state signaled by verification and validation artifacts (the sensors). This control loop is visualized in Figure 5.3.
5.1. ANALYSING AN INDUSTRIAL TOOL CHAIN USING STPA

To summarize, three different types of control loops were identified as being of interest. (1) The manual or semi-manual control loop at the operator level, (2) the manual or semi-manual control loop between the operator and tool chain levels and (3) the fully automated control loop at the tool level. These control loops are visualized together in Figure 5.4. This list of types of control loops may not be exhaustive, but that is a question for future research.

Based on these types of control loops and the general causal factors, a number of causes to each of the programmatic risks identified in the previous step could be
Table 5.2: Programmatic Risks and Causes.

<table>
<thead>
<tr>
<th>Risk</th>
<th>Causes</th>
</tr>
</thead>
</table>
| (13) System Design Specification (SDS) and/or Matlab System Simulation Model (MSSM) not correctly reproduced. | (a) Matlab/Simulink Application Designer does not understand SDS and/or MSSM (lack of training, tight time plan, etc.).  
(b) Matlab/Simulink Application Designer enters erroneous information in the Matlab/Simulink Application Description (M/SAD) (large amount of information in unstructured text, etc.).  
(c) Matlab/Simulink Application Designer enters erroneously structured information in the M/SAD (wrong format, wrong table style, etc.).  
(d) M/SAD is corrupted during storage.  
(e) Matlab/Simulink Application Designer does not understand end product environment completely (unknown EMI, etc.).  
(f) No feedback on M/SAD from later development phases. |
| (80) To create the FPGA Design Specification (FDS) takes too long time, due to manual transformation of data. An incomplete version of the FDS is therefore used. | (a) No clear directive on when the FDS is acceptable.  
(b) FPGA Application Engineer does not understand the implications of the FDS (lack of training, not present in earlier projects, tight time plan, etc.).  
(c) FDS is not structured efficiently and/or appropriately.  
(d) Management Team enforces the acceptance of the FDS prematurely due to project timing constraints.  
(e) FPGA Requirements Specification (FRS) and/or Control Hardware Specification (CHS) are not structured efficiently and/or appropriately.  
(f) Late changes in the FRS and/or CHS require extensive rework of the FDS.  
(g) Late feedback on the FDS from the Matlab/Simulink Application Designer requires extensive rework of the FDS. |

identified. Table 5.2 lists two examples of programmatic risks from [24]. (13) is from the general type of risk that was mentioned as an example in an earlier subsection, i.e. when errors in or missing parts of software create the risk of data corruption. (80) is related to the operator level control loop mentioned in the beginning of this subsection. Of the causes, all are related to the manual or semi-manual control loop between the operator and tool chain levels, except causes (d) and (g) of risk (80) which relate to control loops at the operator level.
5.1. ANALYSING AN INDUSTRIAL TOOL CHAIN USING STPA

Step 4 - Analysis and Reiterations

After the causes had been identified two iterations of changes were made to the control structure of the industrial tool chain. The first iteration was made to make the development process more efficient. The second iteration was made to mitigate the safety issues observed throughout the first two iterations. Analyses were performed between iterations and after the final iteration. Based on the risks and causes identified throughout these three iterations it was possible to make two important deductions.

First, the move from manual to automated tool integration can essentially be defined as “moving between risks”, i.e. while some risks may be solved by the automation there will be others that arise due to it. The important thing to note is that the causes to the new risks will be qualitatively different from those that cause the old risks. While the same result is achieved both through manual and automated tool integration, the way of achieving this result includes a critical difference in that the TIMs used are most likely developed by other operators than those who use the TIMs for developing end products. DO-300 makes a similar point in regard to engineering tools, as mentioned in section 2.3. As an example, consider a data transformation which is set up to replace the manual move of data from a source artifact to a target artifact. The risk that is meant to be solved by this is perhaps that “operator A” may by mistake enter the wrong value into the target artifact, which may be caused by for instance tiredness or insufficient training. However, a risk associated with the data transformation is that it may be erroneous in some cases. This may be caused by “operator B”, i.e. the developer of the data transformation, not understanding the context of the transformation. This cause is far more complex than tiredness or insufficient training, since it can be caused by the insufficient transfer of knowledge from operator A (or other expert) to operator B. To make matters worse, it may even be that the introduction of the data transformation makes it less likely that an erroneous entry is discovered, since operator A is no longer directly involved in overseeing the data transfer and operator B may not be directly involved in the development at all. The choice regarding whether to introduce automated tool integration is therefore strongly linked to the relationship between the developers of the TIMs and the users of them. It might be that some development efforts are not capable of mitigating risks created by developers of third-party software and therefore should refrain from introducing automated tool integration.

Secondly, if the tool integration aspects were true dimensions (i.e. independent of each other) as suggested by [6], then the risks they create would be straightforward to mitigate (albeit the difficulty could depend on such things as the size of the development environment, etc.). A TIM to for instance increase the data ambiguity, the risk of data corruption, etc. However, as seen in the description of step 1 of this case study, risks associated with tool integration can be caused by many different tool integration aspects or even by the relationships between them.
The causes are no less complex, relating to such diverse factors as artifact structure, operator training, unrealistic time plans, corruption of data by software, lack of feedback from verification and validation activities, etc. Similarities were, however, found between the causes of many programmatic risks [24]. These similarities make it possible to identify a set of properties of tool chains that could mitigate at least part of the risks of tool integration no matter which tool integration aspect that was the ultimate cause:

1. *Data Integrity.* That the data used reflects the current state of the development, so that there for instance is no data corruption or use of wrong versions of development artifacts.

2. *Data Mining.* That all data necessary to handle all safety goals correctly during development is possible to extract and present in a human-understandable form, so that decision making is supported by the necessary information.

3. *Traceability for Completeness and Consistency.* That it is possible to trace between development artifacts, so that operators can check if the development artifacts are complete and consistent in comparison to each other.

4. *Well Defined Data Semantics.* That the development artifacts created by the tool chain use unambiguous semantics, so that for instance operators that belong to different engineering disciplines can understand each other correctly.

5. *Process Notifications and Process Control.* That (1) the state and results of the different activities are signaled to the operators who need to know these results and states and (2) erroneous process states are not entered or that entering them is discovered promptly.

6. *Create Customized GUIs.* That it is possible to create GUIs that are either simplified or adapted to operators from a different engineering domain. An example is for instance a customized GUI for a requirement tool that simplifies the interaction for use by operators other than requirement engineers.

7. *Coherent Time Information.* That development artifacts are marked with for instance a production time synchronized to a global clock, so that obsolete information is not used.

8. *Possibility to Automate Tool Usage.* That the tool chain can provide automated tool usage when the task of manually invoking engineering tools or TIMs would be unsuited to operators, so that operators for instance do not have to handle repetitive tasks.

9. *Automated Transformations of Data.* That the tool chain can provide automated transformations of data when the task of manually transferring data between different semantics or syntax would be unsuited to operators, so that operators for instance do not have to manually transfer large amounts of data between development artifacts.
5.2. Chapter Summary

This chapter discussed a case study that investigated the risks that tool integration introduces to an embedded systems development effort. The case study consisted of an extended STPA hazard analysis of an industrial tool chain in four steps:

1. The definition of 13 general types of risk in the context of tool integration. The observation was made that the relationships between the different tool integration aspects lead to a complex situation where several of the general types of risk could be caused by problems in any of several tool integration aspects.

2. The construction of a control structure to identify the programmatic risks of the tool chain at hand. The control structure included both the development process and the different tool integration aspects at hand.

3. The identification of the relevant types of control loops and the subsequent deduction of the causes to the programmatic risks. The relevant types of control loops included (1) the manual or semi-manual control loops at the operator level, (2) the manual or semi-manual control loop between the operator and tool chain levels and (3) the fully automated control loops at the tool level.
Table 5.3: Mitigation of Causes.

<table>
<thead>
<tr>
<th>Cause</th>
<th>Mitigating Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Matlab/Simulink Application Designer does not understand SDS and/or MSSM (lack of training, tight time plan, etc.).</td>
<td>Automated Transformations of Data (Note, however, that this automation creates new risks as discussed earlier in this subsection.)</td>
</tr>
<tr>
<td>(b) Matlab/Simulink Application Designer enters erroneous information in the M/SAD (large amount of information in unstructured text, etc.).</td>
<td>Automated Transformations of Data (Note, however, that this automation creates new risks as discussed earlier in this subsection.)</td>
</tr>
<tr>
<td>(c) Matlab/Simulink Application Designer enters erroneously structured information in the Matlab/Simulink Application Description (M/SAD) (wrong format, wrong table style, etc.).</td>
<td>Automated Transformations of Data (Note, however, that this automation creates new risks as discussed earlier in this subsection.)</td>
</tr>
<tr>
<td>(d) M/SAD is corrupted during storage.</td>
<td>Data Integrity</td>
</tr>
<tr>
<td>(e) Matlab/Simulink Application Designer does not understand end product environment completely (unknown EMI, etc.).</td>
<td>Traceability for Completeness and Consistency</td>
</tr>
<tr>
<td>(f) No feedback on M/SAD from later development phases.</td>
<td>Traceability for Completeness and Consistency, Process Notifications</td>
</tr>
</tbody>
</table>

4. Analysis of the results.

Step 2-4 were reiterated twice, with changes being done to the control structure to make the development process more efficient and mitigate safety issues that were observed. The final analysis lead to two important deductions:

1. First, the move from manual to automated tool integration can essentially be defined as “moving between risks”. While some risks may be solved by automation, there will be others that arise due to it. The important thing to note is that the causes to the new risks will be qualitatively different from those that cause the old risks. While the same result is achieved, the way of achieving this result includes a critical difference in that the TIMs used are most likely developed by other operators than those who use the TIMs for developing end products.
2. Similarities between the causes of many programmatic risks allow for the identification of nine properties of tool chains that could mitigate at least part of the risks of tool integration. These properties can be used to define the relationship between tool integration and safety, i.e. the introduction of tool integration to a tool chain will increase the risk of safety violations in the end product if the tool chain at hand does not support these properties at critical sections. Further analysis of these properties suggest that supporting them are likely to be complex and cumbersome, at least if an insufficient strategy is employed.
Chapter 6

Towards New Methods for Software Tool Qualification

Chapter 6 discusses the results of the case study presented in chapter 5. This discussion has two goals. The first goal is to identify how the findings could be used to enhance the industrial safety standards mentioned in section 2.3. The second goal is to explore some of the possible future venues of research, which can be based on the theory presented in this thesis.

First, the discussion focuses on the possibility to identify which parts of a development environment that need to be analysed in regard to safety when the tool integration metrics presented in section 4.2 indicate a high level of automation. Secondly, the discussion focuses on a proposed new software tool qualification process that (1) deals with the shortcomings of the current software tool qualification approaches described in section 2.3 (2) can be combined with the qualification efforts stipulated by modern safety standards and (3) minimizes the complexity of the required safety analysis.

6.1 The Implications of the Tool Integration Metrics

In 6 the different TIMs are allocated to different tool integration aspects to define a multidimensional space. An implication of this allocation is then that TIMs can be graded on a scale from “lesser” to “advanced” (a metric that supposedly changes over time for each TIM as they become outdated). The alternative metrics presented in section 4.2 aim to provide the same guidance on how to improve tool integration, but at an abstract level where non-functional properties not directly related to isolated parts of an development effort can also be taken into account.

The question then becomes, can the metrics found in section 4.2 guide a safety analysis based on the level of automation that they indicate?

The metrics are intentionally generic enough to allow reasoning at a higher level, but this also means that there are two possible interpretations of a high level of the
tool integration aspects. The first possibility is that automated tool integration is widely utilized in an otherwise highly heterogeneous development environment to join many of the potential islands of automation. The second possibility is that the development environment is simply small and homogeneous, i.e. that there are not many activities performed during development and that all tools make use of the same platform, data types, etc. In the discussion in this section, the first interpretation is adopted. This means that the assumption is that small, homogeneous development environments are not considered. This is not a critical or even a new limitation, since this type of development environments should be easily recognizable and the reference model was in fact developed for analysis of highly heterogeneous environments.

A high level of a specific tool integration aspect usually implies that a high degree of automation has been introduced at some point. This increase in automation implies a strong move from the types of control loops at the operator and tool chain levels to the types of control loops at the tool level (this is visualized in Figure 6.1). The exception is a high level of presentation integration, which does not imply a high degree of automation in itself. This exception is discussed separately in the subsection below dedicated to presentation integration.

In the subsequent subsections the implications of a high level of each of the different tool integration aspects are discussed. This discussion is based on (1) the application of the mapping in section 4.3 to the three different types of control loops identified in section 5.1 and (2) the general causal factors from STPA as related to the mapping in section 4.3 (see Figure 6.2). In other words, for each tool integration aspect, the relevant parts of the three different types of control loops are (1) identified, (2) highlighted and (3) related to specific general causal factors. This allows specific risks implied by the relevant parts and the causal factors to be pointed out as important in regard to a high level of the tool integration aspect at hand. Any additional factors that are required to indicate a strong move from manual to automated tool integration are then discussed in regard to the tool integration aspect at hand.

The overall hypothesis is that a high level of a specific tool integration aspect and a move from manual to automated tool integration will indicate a stronger reliance on automation linked to specific relevant risks and that these risks therefore should be considered more carefully. This increased consideration could be achieved in many ways, for example by treating risks as more likely to occur or by protecting against them to a larger degree. The best way of achieving this will depend on the development environment at hand.

### Platform Integration

The mapping in section 4.3 indicates that all parts of a control loop are dependent on platform integration, since platform integration is concerned with common denominators between tools. Figure 6.1 therefore highlights all the parts of the three different types of control loops identified in section 5.1 in red to indicate their rele-
6.1. THE IMPLICATIONS OF THE TOOL INTEGRATION METRICS

Figure 6.1: Parts Relevant to Platform Integration.

A high level of platform integration indicates a small number of separate platforms in the system at hand. If the other tool integration aspects indicate a strong move from manual to automated tool integration (i.e. they all show a high level of tool integration), then this means that the few platforms that are present are critical (a platform failure could have large repercussions). These platforms will therefore be important to consider in relation to all of the causal factors found in Figure 6.2.

A high level of platform integration together with an overall strong move from manual to automated tool integration will therefore, perhaps ironically, mean that all risks identified at the tool level need to be considered more carefully in the light of for instance failures of common denominators.

Control Integration

Figure 6.3 highlights (also in this figure in red) the parts of the three different types of control loops identified in section 5.1 relevant for control integration.
A high level of control integration indicates a small number of tool integration services that have to be invoked by operators in their processes in the system at hand, i.e. that potentially many services are automated. The causal factors (see Figure 6.2) relevant for control integration then indicate that causes such as delayed, inappropriate, ineffective or missing control or feedback could have large repercussions at the tool level.

A high level of control integration will therefore mean that risks identified at the tool level which can be caused by delayed, inappropriate, ineffective or missing control or feedback should be more carefully considered.

**Data Integration**

The parts of the three different types of control loops identified in section 5.1 relevant for data integration should now be easily deduced and are therefore not visualized here.

A high level of data integration indicates a small number of data interoperability sets to be found in the complete data set in the system at hand, i.e. that few of the groups of syntactic and semantic pairs in the system at hand are not related to each other. The causal factors (see Figure 6.2) relevant for data integration then indicate that causes related to inadequate operation could have large repercussions at the tool level. In regard to data integration, inadequate operation can be understood as related to both semantics (for instance ambiguous data) and syntax (corrupt data).
6.1. **THE IMPLICATIONS OF THE TOOL INTEGRATION METRICS**

A high level of data integration will therefore mean that risks identified at the tool level which can be caused by issues linked to the semantics and syntax of data should be more carefully considered.

**Process Integration**

The parts of the three different types of control loops identified in section 5.1 relevant for process integration should now be easily deduced and are therefore not visualized here.

A high level of process integration indicates that only a few process states have to be handled by operators, i.e. that potentially many process states are handled by automation. The causal factors (see Figure 6.2) relevant for process integration then indicate that causes such as erroneous software adapters, erroneous process logic and unnoticed data deterioration\(^1\) could have large repercussions at the tool level.

A high level of process integration will therefore mean that risks identified at the tool level which can be caused by erroneous software adapters, erroneous process logic and unnoticed data deterioration should be more carefully considered.

---

\(^1\)During a process the data can deteriorate so that it is no longer possible to correctly identify which the current state is or which state should be the next one.
CHAPTER 6. TOWARDS NEW METHODS FOR SOFTWARE TOOL QUALIFICATION

Presentation Integration

The parts of the three different types of control loops identified in section 5.1 relevant for presentation integration should now be easily deduced and are therefore not visualized here.

A high level of presentation integration indicates that few states of automated process models can be invoked by an operator without that operator sharing related states in his own process model. It is likely that operators can only achieve this accurate view of automation in a highly heterogeneous and automated development environment if considerable effort has been spent on constructing correct and diverse presentation integration support. Therefore, if there is a high level of presentation integration and the other tool integration aspects indicate a strong move from manual to automated tool integration (i.e. they all show a high level of tool integration), then the causal factors (see Figure 6.2) relevant for presentation integration indicate that causes such as insufficient communication between operators, inadequately trained operators and confusion of operators handling tools from other engineering disciplines could have large repercussions at the tool level. In other words, in a highly automated context where operators are usually well supported in understanding the behaviour and actions of tools, the anomaly when there is a mismatch between operators and this support is likely to be difficult to handle, undetected for a long time, etc.

A high level of presentation integration together with an overall strong move from manual to automated tool integration will therefore mean that risks identified at the tool level which can be caused by insufficient communication between operators, inadequately trained operators and confusion of operators handling tools from other engineering disciplines should be more carefully considered.

6.2 A New Software Tool Qualification Process

As mentioned in section 2.3 the approaches of the current set of influential, modern industrial safety standards for embedded systems are less sufficient in ensuring safety when development environments scale up and become more automated. Either these standards (1) limit the required software tool qualification and risk leaving out parts of the development environment critical to safety or (2) provide guidelines for software tool qualification that are so generic that they risk leading to sweeping generalizations to avoid the huge costs associated with analysing a whole development environment.

A new software tool qualification process, that attempts to solve the problems of these approaches, is proposed in (Paper C). The following subsections describe and discuss the main points of this proposition.
The Proposition for a New Software Tool Qualification Process

The first step towards a proposition on how to solve the problems mentioned in section 2.3 is to translate each of the properties described in section 5.1 into safety goals, i.e. to state these properties as goal descriptions for which assurance can be proven. Note that these safety goals are defined for the greater system that includes both the development effort and the end product, not only the end product itself. All of the properties are translated in (Paper C), with the translation of two properties given as examples in this thesis:

1. **Data Mining, Safety Goal Description:** A tool chain shall ensure that it is possible to (1) extract all the data necessary to handle all safety goals correctly during development and (2) present this data in a human-understandable form.

2. **Automated Transformations of Data, Safety Goal Description:** Relevant parts of a tool chain shall support the automated transfer of data between tools.

A new software tool qualification process in four steps, inspired\(^2\) by the tool chain approach in [74] and the Safety Element out of Context (SEooC) concept from ISO 26262 [75], can then be based on these safety goals:

1. **Pre-qualification of engineering tools** is performed by tool vendors based on representative tool use cases and a relevant safety standard.

2. **Pre-qualification at the tool chain level** by tool vendors or certification agencies is made possible by the deduction of requirements regarding which safety goals need to be supported at which points of tool chains to avoid unacceptable risks. The elicitation of these requirement is (at least partially) based on the information defined in step 1 and one or several reference workflows. These requirements, the points in the tool chains where they apply and mitigating efforts required (for instance qualification means, qualification of tool integration mechanisms, guidelines for processes and requirements for operator training) are documented and included in reference tool chain descriptions. In effect, one is decomposing the relevant safety goals at the tool chain level to the relevant subgoals at the tool level.

3. **Qualification of the tool chain** identifies the differences between the assumptions of step 2 and the actual use cases, workflow and development environment used when deploying the tool chain.

\(^2\)This inspiration is on a very general level. [74] provided the idea that qualification can be divided into two parts, a first part in which a typical reference workflow is qualified and a second part in which the differences between this reference workflow and the tool chain actually deployed is taken into account. [75] provided the idea that one can stipulate the requirements that a certain entity (in this case a software tool) puts on its context (in this case a tool chain) before it is deployed and then qualify the system by checking that all assumptions are correct in the system actually deployed.
4. **Qualification at the tool level** is based on the actual use cases, workflow and development environment used when deploying the tool chain and performed according to a relevant safety standard. Three issues could therefore require further efforts at this step:

a) An engineering tool has not been qualified by the tool vendor.

b) The relevant safety standard differs between tool vendors and tool users and requires additional efforts in regard to engineering tools if the vendor and user are different (an example of such a standard is DO-178C [64]).

c) The actual use cases, workflow or development environment are different from those assumed during pre-qualification, which means that tools and/or tool integration mechanisms will have to be (re)qualified by the tool user.

**Dealing with the Shortcomings**

Step 2 in the proposed qualification process means that not all parts of a development environment need to be considered during software tool qualification. Instead information available from tool vendors can be used to pinpoint the parts associated with risk due to tool integration. The use of safety goals then leads to a focus on what to ensure (data mining, automated transformations of data, etc.) rather than on what might go wrong (which is a lot in a modern development environment). This should allow the proposed qualification process to avoid the shortcomings of the current approaches towards software tool qualification mentioned in section 2.3.

**Combining with Modern Safety Standards**

Steps 1, 4.a and 4.b in the proposed qualification process are essentially what is required by safety standards such as ISO 26262 [16], IEC 61508:2010[62] and DO-178C [64]. The use of safety goals is then the key also in regard to combining the proposed qualification process with modern safety standards. By stating what an engineering tool requires from its environment steps 2 and 3 are decoupled from the other steps. Engineering tools can then, akin to how safety-related elements can be developed for an unspecified environment according to the SEooC concept, be qualified separately from TIMs as long as the deployed development environment fulfills the assumptions used during pre-qualification (i.e. the safety goals). This should mean that the proposed qualification process does not interfere with the set of influential, modern industrial safety standards for embedded systems that are analysed in section 2.3.

**Dealing with the Complexity**

Nevertheless, several issues with implications for both tool integration in general and the approach taken in this thesis are stated in section 5.1. A valid question is
6.2. A NEW SOFTWARE TOOL QUALIFICATION PROCESS

how the proposed qualification process deals with those issues.

1. The “move between risks” when moving from manual to automated tool integration is especially important, since a software tool qualification effort will deal with the automated part of tool integration. The proposed qualification process deals with this issue by breaking up the qualification process into separate steps for tool vendors and tool users.

2. The fact that any single risk may require support from several properties is not problematic, since the safety goals can be combined if they happen to be relevant for the same part(s).

3. The proposed qualification process favours early planning in regard to the development environment. This means that the additional parts required to ensure safety can be taken into account early, which should minimize their impact.

4. The fact that some of the properties are emergent at the tool chain level also means that some of the safety goals will be emergent at this level. The decomposition required in step 2 of the proposed qualification process is therefore impossible for these safety goals. However, even if a safety goal is emergent, it may still be possible to isolate some portion of the goal behaviour that is partially composable [76]. In other words, it might be possible to identify (a) part(s) of the safety goal that can be decomposed to the entities at lower levels of organization. When the target is to ensure sufficient safety (which is the aim of safety standards) it should be sufficient to decompose the most important part(s) of the safety goal to the tool level. In (Paper C) this most important part of safety goals was defined as the relation to the way increased automation of tool integration hides what occurs to the operators, rendering efforts at high levels of organization less effective. The data mining safety goal mentioned above could then be reformulated into “a tool chain shall ensure that data regarding safety-related development artifacts that is not handled by operators directly is automatically gathered, analysed and presented to operators”. The data mining safety goal could then be partially decomposed into the requirement for qualified tool integration mechanisms for data mining and analysis (for instance a qualified project dashboard). What can be considered as the most important part of the emergent safety goals will be context-specific and therefore needs to be decided during each pre-qualification effort.

Additional Benefits

The proposed qualification process should also bring the following benefits:

- Framing the requirements on ensuring safety for tool integration as safety goals means that:
Ensuring safety can be realized in many different ways. Several solutions can be compared already during step 2, providing an early estimate of how much effort must go into procuring tools, training operators, etc. Alternatively, the different solutions can be used to tailor tool chains for different development efforts.

Those TIMs that indirectly contribute to a risk are also considered. Consider for instance a bug tracking tool, which is not likely to directly influence the engineering activities performed in the main development process. However, if the data of this tool is used for Go/No Go launch decisions the requirement that this data should always be up-to-date is critical to ensuring safety. This is unlikely to be considered if the choice of which software should be qualified is for instance done based on the bottom-up approach of considering the flow of data in the main development process, as opposed to the top-down approach based on considering safety goals.

- The favouring of early planning could support tool chain developers in avoiding *islands of automation* [72].

### 6.3 Chapter Summary

This chapter discussed the results of the case study presented in chapter 5 to propose ways in which they can be used to support future versions of industrial safety standards for embedded systems. Two propositions were made:

- The first proposition was that the metrics for the tool integration aspects found in section 4.2 could provide guidance in regard to safety when moving from manual to automated tool integration. This guidance was then defined by applying the mapping found in section 4.3 to the three different types of control loops identified in section 5.1:

  - A high level of platform integration together with an overall strong move from manual to automated tool integration indicates that all risks identified at the tool level need to be considered more carefully in the light of for instance failures of common denominators.

  - A high level of control integration indicates that risks identified at the tool level which can be caused by delayed, inappropriate, ineffective or missing control or feedback should be more carefully considered.

  - A high level of data integration indicates that risks identified at the tool level which can be caused by issues linked to semantics and syntax of data should be more carefully considered.

  - A high level of process integration indicates that risks identified at the tool level which can be caused by erroneous software adapters, erroneous
process logic and unnoticed data deterioration data should be more carefully considered.

- A high level of presentation integration together with an overall strong move from manual to automated tool integration indicates that risks identified at the tool level which can be caused by insufficient communication between operators, inadequately trained operators and confusion of operators handling tools from other engineering disciplines should be more carefully considered.

- The second proposition was a new software tool qualification process based on translating the properties found in section 5.1 into safety goals. The qualification process includes four steps, namely (1) pre-qualification of engineering tools, (2) pre-qualification at the tool chain level, (3) qualification of the tool chain and (4) qualification at the tool level, and should bring the following benefits:

  - It should deal with the shortcomings of the software tool qualification processes of the safety standards mentioned in section 2.3.
  - It should be possible to combine with the safety standards mentioned in section 2.3.
  - It should allow for dealing with the complexity associated with the results from the case study described in chapter 5.
  - It should bring additional benefits associated with early planning and the identification of TIMs that indirectly influence the safety of a development effort.
Chapter 7

Summary, Contributions, Future Work and Conclusions

7.1 Summary

Tool integration is a research field that has received an increased amount of attention during the last few years, evident in research projects such as Cesar [55], iFEST [22] and MBAT [56]. This is not surprising, given that the increasing complexity of embedded systems development is becoming difficult to handle with development environments based on disjoint engineering tools. A solution to this problem based on increased tool integration, and especially of automated tool integration, is emerging as a likely candidate. This development will probably lead to an increased impact of tool integration on non-functional properties of both development environments and end products. The research during the last two decades has, however, not focused on non-functional properties, but rather on the technical details of implementing tool integration.

This thesis has discussed the relationship between tool integration and the safety of end products (focusing on changes in tool integration, embedded systems and a limited part of the product life-cycle). To enable this discussion the following theory was first introduced:

- The school of thought known as Systems Thinking.
- A causality model (STAMP) and a hazard analysis technique (STPA) based on systems theory.

This theory was then used to build new concepts to help describe the relationship between tool integration and safety:

- A new conceptual model of a development effort based on systems theory.
- A new reference model for tool integration developed for analysis of highly heterogeneous environments.
All of these State of the Art concepts were used in a case study that investigated the risks that tool integration introduces to a development effort. This case study provided the basis for the discussion in the thesis. This basis was primarily created through the identification of nine properties of tool chains that could mitigate at least part of the risks of tool integration, but also through the identification of three control loops important to tool integration and safety in the new conceptual model.

Based on these findings this thesis could provide two propositions. The first proposition concerns how the metrics for tool integration aspects described in the new reference model can be used to provide guidance in regard to safety when moving from manual to automated tool integration. The second proposition is a new software tool qualification process that promises to deal with the shortcomings of current safety standards in regard to tool integration.

7.2 Contributions

To emphasize the original contributions of this thesis they are summarized in this section.

A New Conceptual Model of a Development Effort Emphasizing Tool Integration

A new conceptual model of a development effort that emphasizes tool integration was introduced to enable the analysis of the risks related to tool integration. This model, based on Systems Thinking, consists of four levels of organization. From top to bottom they are:

- **The management level.** This level consists of the entities concerned with the management of the development effort. These entities constrain the basic conditions and actions of the involved operators.

- **The operator level.** This level consists of the separate operators that implement any of the processes needed during the product life-cycle phases. These operators constrain the development activities through the processes they work with and provide feedback to the management level through for instance reports.

- **The tool chain level.** This level consists of the tool chains used during the product life-cycle phases. These tool chains define (and therefore constrain) the interactions between the different parts of the development environment and provide feedback to the operator level through process states.

- **The tool level.** This level consists of the separate engineering tools and TIMs. These provide feedback to the tool chain level through the formation of islands of automation based on the properties of the entities included in the tool chains.
7.2. CONTRIBUTIONS

Based on this model three parts of a development effort important to the relationship between tool integration and safety can be discerned and defined. The first part consists of the manual or semi-manual control loops at the operator level, i.e. when different operators interact with each other to for instance review each others work. The second part consists of the manual or semi-manual control loops between the operator and tool chain level, i.e. when the operators execute some part of a development process to bring the development of the end product forward. The third part consists of the fully automated control loops at the tool level, i.e. when an engineering tool or TIM controls (an)other engineering tool(s) or TIM(s).

A New Reference Model for Tool Integration

A new reference model for tool integration developed for analysis of highly heterogeneous environments originally presented in (Paper A) was revisited and updated in this thesis. This reference model redefines the five tool integration aspects from [6], i.e. platform, control, data, process and presentation integration. The new reference model also provides metrics for and defines the relationships between these tool integration aspects. While the relationships are the same as those presented in (Paper A), in this thesis the metrics were updated based on critique from supervisors, fellow PhD Candidates, workshop reviewers and conference reviewers and then related to the automation of tool integration.

- **Platform Integration** is the degree to which tools share a common environment. The metric of platform integration is the number of instances when parts of tool chains need to be joined by operators.

- **Control Integration** is the degree to which tools can issue commands and notifications correctly to one another. The metric of control integration is the number of tool integration services that have to be invoked by operators in their processes. Control integration is dependent on platform integration for tools to be able to reach one another.

- **Data Integration** is the degree to which tools can, in a meaningful way, access the complete data set available within the system of interest. The metric of data integration is the number of data interoperability sets in a system. Data integration is dependent on platform and control integration for tools to be able to reach one another’s data.

- **Process Integration** is the degree to which the interaction with tools can be supervised. The metric of process integration is the number of process states that need to be handled by operators. Process integration is dependent on control and data integration, since process awareness is propagated by notifications and process state is defined by data.

- **Presentation Integration** is the degree to which tools, in different contexts, can set up user interaction so that the tool input/output is perceived correctly.
The metric of presentation integration is the number of states of automated process models that can be invoked by an operator without that operator sharing related states in his own process model. Presentation integration is dependent on platform, control and data integration, since they limit the ways of presenting input/output to users.

In addition to the new reference model the possibility to use a new graphical format for visualizing a control structure was mentioned, but not elaborated on. This graphical format highlights tool integration in a development effort by combining a SysML Activity Diagram (for the development process) with a tool integration aspect description based on the graphical format described in (Paper A) (for the different tool integration aspects of a development environment).

Safety-related Tool Chain Properties

Similarities between the causes of many risks created by tool integration made it possible to identify nine properties of tool chains that could mitigate at least part of said risks. These properties are:

1. **Data Integrity.** That the data used reflects the current state of the development, so that there for instance is no data corruption or use of wrong versions of development artifacts.

2. **Data Mining.** That all data necessary to handle all safety goals correctly during development is possible to extract and present in a human-understandable form, so that decision making is supported by the necessary information.

3. **Traceability for Completeness and Consistency.** That it is possible to trace between development artifacts, so that operators can check if the development artifacts are complete and consistent in comparison to each other.

4. **Well Defined Data Semantics.** That the development artifacts created by the tool chain use unambiguous semantics, so that for instance operators that belong to different engineering disciplines can understand each other correctly.

5. **Process Notifications and Process Control.** That (1) the state and results of the different activities are signaled to the operators who need to know these results and states and (2) erroneous process states are not entered or that entering them is discovered promptly.

6. **Create Customized GUIs.** That it is possible to create GUIs that are either simplified or adapted to operators from a different engineering domain. An example is for instance a customized GUI for a requirement tool that simplifies the interaction for use by operators other than requirement engineers.

7. **Coherent Time Information.** That development artifacts are marked with for instance a production time synchronized to a global clock, so that obsolete information is not used.
8. **Possibility to Automate Tool Usage.** That the tool chain can provide automated tool usage when the task of manually invoking engineering tools or TIMs would be unsuited to operators, so that operators for instance do not have to handle repetitive tasks.

9. **Automated Transformations of Data.** That the tool chain can provide automated transformations of data when the task of manually transferring data between different semantics or syntax would be unsuited to operators, so that operators for instance do not have to manually transfer large amounts of data between development artifacts.

This list may not be complete, but the identification of these properties opens up the possibility to reduce complexity when dealing with risks associated with tool integration.

**The Implications for Safety due to a High Level of Automation of Tool Integration**

A proposition for how to identify the implications for safety due to a high level of automation of tool integration was presented. This proposition was made possible by (1) mapping the different tool integration aspects from the new reference model to the different parts of a generic control loop and then (2) applying this mapping to the three different parts of a development effort identified as related to tool integration by the new conceptual model of a development effort. The implications were proposed to be that:

- A high level of platform integration together with an overall strong move from manual to automated tool integration indicates that all risks identified at the tool level need to be considered more carefully in the light of for instance failures of common denominators.

- A high level of control integration indicates that risks identified at the tool level which can be caused by delayed, inappropriate, ineffective or missing control or feedback should be more carefully considered.

- A high level of data integration indicates that risks identified at the tool level which can be caused by issues linked to semantics and syntax of data should be more carefully considered.

- A high level of process integration indicates that risks identified at the tool level which can be caused by erroneous software adapters, erroneous process logic and unnoticed data deterioration data should be more carefully considered.

- A high level of presentation integration together with an overall strong move from manual to automated tool integration indicates that risks identified at
CHAPTER 7. SUMMARY, CONTRIBUTIONS, FUTURE WORK AND CONCLUSIONS

the tool level which can be caused by insufficient communication between operators, inadequately trained operators and confusion of operators handling tools from other engineering disciplines should be more carefully considered.

A New Software Tool Qualification Process

A proposition for a new software tool qualification process was presented. This proposition was made possible by translating the safety-related tool chain properties into safety goals. This allows a qualification process to be partitioned into four steps, namely:

1. **Pre-qualification of engineering tools** based on representative tool use cases and a relevant safety standard.

2. **Pre-qualification at the tool chain level** by the deduction of requirements regarding which safety goals need to be supported at which points of tool chains to avoid unacceptable risks.

3. **Qualification of the tool chain** to identify the differences between the assumptions of step 2 and the actual use cases, workflow and development environment used when deploying the tool chain.

4. **Qualification at the tool level** based on the actual use cases, workflow and development environment used when deploying the tool chain and performed according to a relevant safety standard.

Adhering to this new qualification process should bring the following benefits:

- It should deal with the problem during software tool qualification that either the danger posed by different parts of a development environment is subject to sweeping generalizations or that important parts for ensuring safety are likely to be ignored (shortcomings of the software tool qualification processes of modern safety standards for embedded systems development).

- It should be possible to combine with modern safety standards for embedded systems development.

- It should allow for dealing with the complexity associated with the nine safety-related tool chain properties.

- It should bring additional benefits associated with early planning and the identification of TIMs that indirectly influence the safety of a development effort.
7.3 Future Work

This section discusses future research that, based on what has been presented in this thesis, is deemed of special interest. This interest primarily stem from a number of concerns in regard to the validity of the approach and the results presented in this thesis. The most important concerns are therefore first described in a separate subsection, before a second subsection discusses future research efforts of special interest in detail.

Concerns

The approach and results of this thesis can be criticized based on (1) the limited extent to which relevant domains were analyzed, (2) the limited number of aspects of the relevant domains that were analyzed, (3) the lack of diversity of the research methods that were used and (4) the limited verification of the results. Based on the critique from supervisors, fellow PhD Candidates, workshop reviewers and conference reviewers five concerns can be highlighted as especially important to mitigate during future research:

- The first concern is in regard to the completeness of the list of safety-related tool chain properties, or even that it may be impossible to render such a list complete. This list is currently only based on a single case study and there is a high possibility that further case studies may yield additional properties. These additional properties may complicate further analysis, for instance by contradicting the properties identified so far or by being dependent on which domain the development is taking place in.

- The second concern is in regard to the analysis in this thesis being based solely on qualitative data. Further quantitative analyses may show that the actual risks associated with tool integration are usually so few as to almost always be acceptable.

- The third concern is that the implications due to a high level of automation of tool integration and the proposition for a new qualification process have not been verified to any large degree, even if the logic is sound. Further case studies may show that (1) other, commonly existing properties of a development environment impact the preconditions for risks as to always make the proposed implications useless or that (2) other, commonly existing requirements of industrial safety standards conflict with the proposed qualification process.

- The fourth concern is that the different organizational and psychological properties of the operators belonging to for instance different engineering disciplines have not been sufficiently taken into account, since the analyses so far have treated all operators (almost) as equals. Further analyses may show
that different types of operators exhibit differences so vast as to make the approach taken in this thesis futile.

- The fifth concern is that not all of the product life-cycle phases have been taken into account. A decision that improves safety in the short term may very well be unacceptable if the whole product life-cycle is considered. Consider for instance the choice to deploy an optimal, but very specialized and not readily available, engineering tool in a development environment. Short-term this may increase safety, but if the vendor of this tool can no longer support it then maintaining the integration of this tool may have serious implications on its reliability and safe operation. The implications of thinking long-term may supersede all the implications of the results identified so far.

**Future Venues of Research**

This subsection presents a more detailed description of possible future venues of research. Most of the described research efforts are aimed at mitigating the concerns described in the previous subsection, but included below are also some research efforts which are simply judged as having a high potential to yield useful results in the regard to how tool integration relates to non-functional properties. Special emphasis is placed on matters related to safety, since this thesis has focused on this non-functional property.

**A Complete List of Safety-related Tool Chain Properties**

The list of safety-related tool chain properties identified in chapter 5 is most likely not complete. Additional case studies could support the uncovering of additional safety-related tool chain properties. These case studies should preferably be conducted in many different industrial domains and target development environments featuring different mixes of manual and automated tool integration.

**Verify the Implications for Safety due to a High Level of Automation of Tool Integration**

As shown throughout this thesis, automated tool integration can be the source of many risks in a development environment. It would therefore be useful to be able to quantify the risks of moving from manual to automated tool integration. In regard to the contents of this thesis it would, first and foremost, be valuable as a verification of the proposition on the implications for safety due to a high level of automation of tool integration. However, this quantification would also be beneficial when planning a move from manual to automated tool integration and for those studying the certification process to ensure safety afterwards.

Unfortunately there are at least two difficulties with utilizing quantitative methods to quantify the risks of moving from manual to automated tool integration.
First, while modern development environments are utilizing COTS tools to an increasing degree, the State of the Practice is still limited to a high degree of manual tool integration. Secondly, development environments will be at least one step removed from any chain of events leading up to an accident, meaning that accident investigations (lacking a systems perspective) will most likely fail to investigate whether any part of the development environment contributed to the accident. Both of these difficulties imply that there will be little data to find on the risks of interest in conventional sources (such as publicly available accident reports).

To yield useful results access to more detailed information, closer to the development of a particular end product, is required. The approach most likely to succeed is probably one where deviation reports internal to a manufacturer of embedded systems can be obtained and analysed. By tracing the deviations from normal operation by end products to causes in a development environment the risks of interest could be quantified. Several iterations of analyses, interspersed with interviews and preferably conducted in cooperation with several manufacturers from different industrial domains, would most likely have to be performed to quantify the risks with sufficient certainty.

If access to this type of internal data cannot be obtained, then an alternative could be a qualitative study based on interviews to gather expert knowledge on current problems with automated tool integration.

**Verify the New Software Tool Qualification Process**

Section 6.2 discusses the qualification of software related to tool integration by the use of safety goals that can be deduced from the results of the case study presented in chapter 5. However, the possible gain of qualifying software related to tool integration in this way has not been quantified. This would be valuable both to verify the method as such and to compare it to other methods in regard to ensuring safety. However, to collect enough data to quantify this is likely to be very time-consuming, since analysing only a single software tool qualification effort according to a modern safety standard is usually a substantial task. A suggestion is therefore to instead perform a negative case study to try to disprove that there is a gain. Such a case study, if performed in a stringent manner, could at least strengthen the case for the new software tool qualification process if it does not disprove it outright. If a tool chain with associated deviation reports could be found, then a negative case study could be performed by:

1. Qualifying the tool chain by using the method described in section 6.2.

2. Qualifying the tool chain according to all the different methods for certifying tool chains described in the safety standards mentioned in section 2.3, regardless of the domain the tool chain was initially meant for. If there are domain-specific issues that need to be handled this can be noted, but the methods all seem generic enough to allow for a straight-forward comparison.
3. Tracing the deviations from normal operation by end products to causes in the tool chain and noting which of the qualification methods would have ensured that the deviation would not have occurred. By favouring the established qualification methods when in doubt, the case study could reasonably be seen as negative.

Furthermore, while performing this type of negative case study additional issues related to the proposed software tool qualification process can be studied:

- Many safety standards differ between different levels of criticality, i.e. acknowledging that the task of ensuring safety for some systems needs to be more carefully performed than for other systems. This can probably be argued also in regard to which or to which detail safety goals need to be considered for specific types of systems.

- The different practices of engineers belonging to different engineering disciplines have been discussed as a potential source of risk. However, there are other groupings that operators can be partitioned into that may lead to risk, for instance the different practices adopted by operators belonging to different organizations.

Increase the Foundation for Analysing Tool Integration

As mentioned in section 1.3, the aim of this thesis is to create a foundation for analysing tool integration in relation to non-functional properties. Therefore the scope of the thesis has been limited with the intention that the findings based on this restricted scope will later support a more universal analysis.

The limitations of the scope can be removed by additional case studies similar to the one presented in [24]:

- Manual and automated tool integration can be studied in isolation, to provide insights into how safety relates to development environments that rely heavily on one of these types of tool integration.

- Development environments pertaining to the development of other types of systems than embedded systems (information technology systems, cloud computing systems, software applications, etc.) can be studied to establish more generic findings on how safety impacts development environments.

- Additional phases of the product life-cycle, such as the operations and maintenance phases, can be studied to ensure that optimizing the relationship between safety and tool integration in the beginning of the product life-cycle does not bring negative effects in the end of it.

The limitation in focusing only on safety would require other approaches to overcome:
• Safety is only one of a multitude of non-functional properties relevant to software development. Cost, security, scalability, interoperability, portability and reliability are just a few other non-functional properties that would be of interest for managing a development effort. Each would require a thorough analysis, using the tools of that particular domain, to describe and relate to both tool integration and safety. If one considers cost as an example, this is a non-functional property of development efforts that has been studied since before Software Engineering was recognized as a separate field of study (Boehm gives examples from the 1950s in [77]). The proposed methods and tools for handling cost are therefore many, including parametric models, expert judgment, analogies, top-down and bottom-up calculations, case-based reasoning, regression trees, artificial neural networks and genetic algorithms to name a few. The COCOMO software cost model (latest version published in [78]) can be mentioned as being of special interest, since it has been widely used, evaluated in a range of organizations and explicitly mentions the use of software tools as being a factor in the final cost of a development effort. The last of these benefits has been further developed in proposed elaborations of the software tool parameter to consider tool integration [79]. Additionally one can find model extensions to COCOMO (i.e. COCOTS [80]) which handles the special considerations to be taken into account when working with COTS software components (of interest since a tool chain is often made up of COTS software components).

Support Systems

Expert systems (i.e. software programs providing decision support) can be useful for risk analysis in relation to tool integration. The amount of software support for this activity is however minimal in relation to STPA, even though several parts of this method would benefit from this type of support (Paper D).

It would therefore be interesting to analyse the possibilities and problems of such expert systems for STPA associated with for instance:

1. The search for weak and strong parts of the system analysed.

2. The identification of faults that occur due to the inadequate, simultaneous control exerted by several controllers or constraints.

3. Disseminating knowledge from experts (on engineering disciplines, organizations, etc.) to risk analysts.

4. Simulation (for instance using the information gathered in expert systems to analyse the impact of inadequacies).

5. System boundary support to identify incomplete system descriptions.
Recent research at KTH (the Royal Institute of Technology) has targeted the creation of a *Domain Specific Modeling Language* for tool chains named *Tool Integration Language* (TIL). TIL could be extended with relevant concepts to allow a TIL description to be analysed in regard to the findings of this thesis [81]. If this proof of concept is successful, TIL could then be extended in steps to allow expert systems designed to support the use of STPA to use TIL descriptions to explore the possibilities mentioned in the bullet list above.

### 7.4 Conclusions

In the introduction of this thesis three research questions were stated. To conclude this thesis these questions are answered based on the research presented throughout the thesis.

*What is the relationship between tool integration in development environments and the safety of embedded systems developed by use of said environments?*

The hypothesis in regard to this main research question was that constraining a development effort could stop unsafe behaviour from being introduced into end products due to tool integration. These constraints could then be used to define the relationship between tool integration and the safety of embedded systems.

As shown in this thesis the mechanisms and risks associated with tool integration form complex relationships that are difficult to define, making the mechanisms and risks less useful when trying to describe the relationship between tool integration and safety. Even the new reference model for tool integration, developed for analysis of highly heterogeneous environments, yielded tool integration aspects with relationships so complex as to make them difficult to apply when reasoning about safety.

However, analysing the causes to the risks introduced by tool integration enabled the deduction of safety-related properties of tool chains. Properties that then could be translated into safety goals which stipulate what needs to be ensured to mitigate safety concerns due to tool integration, i.e. how the development effort needs to be constrained to avoid risks. This thesis has not presented a complete set of constraints to define the relationship between tool integration and safety, but these safety goals allow for defining part of this relationship in a way that deals with complexity successfully.

*Safety standards based on Systems and Software Engineering principles define the boundaries for development efforts targeting safety-critical, embedded systems. Do these safety standards provide guidance on tool integration? If they do, what form does this guidance take and is it different in standards targeting different industries or between different versions of the same standards?*
The hypothesis in regard to this derivative research question was that the safety standards were likely to define activities to ensure safety when transitioning between different phases of the product life-cycle, but at the same time they were likely to strive towards being technology agnostic.

As shown in this thesis the safety standards do indeed try to be technology agnostic by defining processes for how to qualify software to ensure the mitigation of any risks this software may introduce into the development effort. Unfortunately these qualification processes focus on engineering tools and therefore fail to take all risks associated with tool integration into account. It is therefore possible to conclude that the guidance provided for all identified industrial domains is not sufficient in regard to tool integration.

What are the differences between manual and automated tool integration in relation to the safety of embedded systems? Is one to prefer over the other in the context of the safety of end products?

The hypothesis was that these different solutions would be interrelated in some way, rather than possible to define as better or worse.

As shown in this thesis the introduction of automated tool integration qualitatively changes a development effort. However, this qualitative difference will be in the causes to the risks found within the development effort, causes that will be one step removed from the development of the end product. This is due to the fact that the automated tool integration is likely to be developed by other operators than those who use the automated tool integration for developing end products. Which type of tool integration that is to prefer in regard to ensuring safety is therefore dependent on the resources of the development effort at hand. If these qualitatively different causes can be handled, then automated tool integration can bring benefits related to other non-functional properties (efficiency, cost, etc.). If these causes cannot be handled, then introducing automated tool integration is in itself a risk.

The decision regarding whether to automate tool integration is therefore essentially a trade-off decision. The introduction of automated tool integration makes sense, if the benefits and costs of introducing automated tool integration outweigh the difficulties and costs associated with this introduction. In this thesis we have reasoned around the risks introduced by an increase in tool integration, providing a start on how to weigh tool integration against safety. Further research may provide a foundation that allows a faster, clearer and more exact way of weighing tool integration against even more non-functional properties of development efforts, development environments and end products.
Bibliography


