Chapter 1

Introduction

> *The primary safety problem in software-intensive systems is not software "failure" but the lack of appropriate constraints on software behavior.*
>
> — *N. Leveson*

Given the rapid innovations in software and technology, many complex systems are becoming software intensive. Software-intensive systems are systems in which software interacts with other software, systems, devices, sensors and with people [Wir+08]. Software has become an indispensable part of many modern systems and often performs the main safety-critical functions. Software safety as stated in [Alb+99] is practically concerned with the software causal factors that are linked to individual hazards and ensured that the mitigation of each causal factor is traced from software requirements to design, implementation, and test. An unexpected behavior of software may lead to catastrophic results such as injury or loss of human life, damaged property or environmental disturbances. Therefore, it becomes essential to test the software components for unexpected behavior before using them in practice [Min91]. The Toyota Prius, the General Motors airbag and the loss of the Mars Polar Lander (MPL) mission [JPL00] are well-known software problems in which the software played an important role in the loss, although the software had been successfully verified against all functional requirements. Recently, Google's self-driving car and Tesla autopilot are the two latest software-related accidents in the automotive domain.

Many different safety analysis approaches exist. The most widely practiced safety analysis approaches are Fault Tree Analysis (FTA) [Ves+81], Failure Mode and Effect Criticality Analysis (FMECA) [FME67] and Hazard and Operability Analysis (HAZOP) [Tro68] which are developed over 50 years ago, before computers were common in engineered systems. As a result, these safety analysis

methods do not completely ensure safety in complex systems. In such systems, the accidents are resulting when component failures, external disturbances, and/or dysfunctional interactions among system components are not adequately handled [Lev11]. A new trend is to advance safety analysis techniques using the system and control theory rather than the reliability theory. STAMP (System-Theoretic Accident Model and Processes) [Lev11] is a modern approach for safety engineering that promises to overcome the problems of the traditional safety analysis techniques. STPA (System-Theoretic Process Analysis) is designed for safety analysis in the system development and operating stages; the goal is to identify hazards existing in the system and provide safety constraints to mitigate those hazards.

1.1. Problem Statement

Safety is a system level property and, hence, needs to be analysed on the system level. Therefore, the software must fully satisfy the corresponding safety requirements which constrain the software from these behaviors that violate the safety of the whole system. Ensuring the safe operations of software involves that software must deal with hazardous behaviors which are identified by safety analysis at an early stage. STPA has been developed to derive detailed safety requirements for complex systems. However, STPA has not yet been placed into the software development process of safety-critical systems, and the current software engineering methods do not explicitly incorporate STPA safety activities. STPA safety analysis is often handled separately by the safety engineers, while software developers are usually not familiar with system safety analysis processes. Therefore, there is a gap between the software and safety engineering processes.

Moreover, the complexity of safety-critical software makes exhaustive software testing impossible. Therefore, we need to make sure that safety is sufficiently considered. Yet, many existing testing approaches and tools do not incorporate information from safety analysis. In case they do, they rely on traditional safety analysis techniques such as FTA and FMECA which focus on component failures instead of component interaction failures. A software safety testing approach integrated with alternative systems-theoretic safety analysis approaches such as STPA has been missing.

1.2. Research Objectives

This dissertation aims to fill the aforementioned gap to place STPA in the software engineering process to help software and safety engineers in deriving the appropriate software safety requirements, formally verifying them, generating safety-based test cases to recognize the associated software risks, and reduce them to a low level. Therefore, this dissertation has three main objectives. The first objective is to develop a comprehensive safety engineering approach which integrates the STPA safety analysis activities with the software verification activities in a software engineering development process such as the V-Model [FM91] to offer seamless safety analysis and verification. The second objective is to develop algorithms to automate the safety-based formal verification and testing activities of the proposed approach. Finally, this dissertation aims also at developing an open source tool to support the application of STAMP methodologies as well as the software verification activities based on the information derived during an STPA safety analysis.

1.3. Contributions

The thesis provides four contributions:

- **Developing a comprehensive safety engineering approach based on STPA to derive software safety requirements at the system level** [AWL15; AW14b; AW15a], formally verify them at the design and implementation levels, and generate safety-based test cases from the STPA results. The proposed approach has the following contributions: (1) We develop an algorithm based on STPA to derive unsafe software scenarios and automatically translate them into software safety constraints and specified them into a formal specification in LTL (Linear Temporal Logic) [Pnu77]. (2) We explore how to build a Safe Behavioral Model (SBM) based on the STPA control structure diagram and the process model. (3) We develop an algorithm to automatically extract the Safe Test Model (STM) from SBM model and check its correctness by automatically transforming it into a formal model such as an SMV (Symbolic Model Verifier) [McM93] and

verify it against the STPA-generated safety requirements using the NuSMV model checker [Cim+00].

- **Providing formal definitions and algorithms for automation support** to the proposed approach (STPA SwISs), especially the document of STPA safety analysis results, automatically generate unsafe software scenarios based on the process model variables as well as automatically verify STPA software safety requirements and generate safety-based test cases.
- **Developing an open-source tool called XSTAMPP** to support the application of the STAMP methodologies as well as the STPA SwISs approach and enable the software and safety engineers to derive software safety requirements, automatically verify their software implementation and design against the STPA results and automatically generate safety-based test cases directly from the STPA results.
- **Evaluating the application of the STPA SwISs approach**. We conducted three case studies. The first case study is a pilot case study, which was conducted at our institute during developing a safe software simulator of the Adaptive Cruise Control (ACC) system with stop-and-go function to explore the application of the STPA SwISs approach during the development process of a safety-critical software. The second case study was conducted as an industrial case study at the German company BMW Group. We applied the proposed approach to BMW active cruise control system with stop-and-go function of the new car model G11. The case study was performed in the headquarter of BMW Group in Munich, Germany. The third case study was conducted as an industrial case study at the German company Continental. We first applied the STPA approach to the current project of the fully automated vehicle to evaluate and assess the architecture design of the fully automated vehicle. Then, we explored the application of the STPA SwISs approach at the software level of the fully automated vehicle to derive software safety requirements, automatically generate the unsafe software scenarios, translate them into software safety requirements and formalize the software safety requirements into formal specification in LTL to evaluate the architecture of the fully automated vehicles.

1.4. List of Publications

The following is a list of the publications which have been done to finish this work:

- Abdulkhaleq, A., Wagner, S. (2016) A Systematic and Semi-Automatic Safety-based Test Case Generation Approach Based on System-Theoretic Process Analysis. submitted to ACM Transactions on Software Engineering and Methodology.
- Abdulkhaleq, A., Sebastian, V., Wagner, S., Thomas, J. (2016) An Industrial Case Study on the Evaluation of a Safety Engineering Approach for Software-Intensive Systems in The Automotive Domain. Preprint. University of Stuttgart.
- Abdulkhaleq, A., Wagner, S., Lammering, D., Röder, J., Balbierer, N., Ramsauer, L., Raste, T., Boehmert, H. (2016) A Systematic Approach Based on STPA for Developing a Dependable Architecture for Fully Automated Driving Vehicles, in Proceeding of the Procedia Engineering Journal.
- Abdulkhaleq, A., Wagner, S. (2016) XSTAMPP 2.0: New Improvements to XSTAMPP Including CAST *Accident* Analysis and an Extended Approach to STPA. 2016 STAMP Conference at Massachusetts Institute of Technology (MIT), 21 March 2016, Boston, USA.
- Abdulkhaleq, A., Wagner, S., Leveson, N. (2015) A Comprehensive Safety Engineering Approach for Software-Intensive Systems Based on STPA, Procedia Engineering, Volume 128, 2015, Pages 2-11, ISSN 1877-7058.
- Abdulkhaleq, A., Wagner, S. (2015) Integrated Safety Analysis Using System-Theoretic Process Analysis and Software Model Checking. In Computer Safety, Reliability, and Security (Safecomp2015), Lecture Notes in Computer Science, Springer International Publishing, Delft, Netherlands (22-25 September 2015)
- Abdulkhaleq, A., Wagner, S. (2015) XSTAMPP: An eXtensible STAMP Platform As Tool Support for Safety Engineering. 2015 STAMP Conference at Massachusetts Institute of Technology (MIT), 26 March 2015, Boston, USA.

- Abdulkhaleq, A., Wagner, S. (2014) A Software Safety Verification Method Based on System-Theoretic Process Analysis. In Computer Safety, Reliability, and Security (Safecomp2014), Lecture Notes in Computer Science, Springer International Publishing, Vol. 8696, pp. 401-412.
- Abdulkhaleq, A., Wagner, S. (2014) A-STPA: An Open Tool Support for System-Theoretic Process Analysis. 2014 STAMP Conference at Massachusetts Institute of Technology (MIT), 27 March 2014, Boston, USA.
- Abdulkhaleq, A., Wagner, S. (2013) Experiences with Applying STPA to Software-Intensive Systems in the Automotive Domain". 2013 STAMP Conference at MIT, Boston, USA.
- Abdulkhaleq, A., Wagner, S. (2013) Integrating State Machine Analysis with System-Theoretic Process Analysis (STPA). Multikonferenz Software Engineering 2013 (SE 2013), ZeMoSS - Zertifizierung und modellgetriebene Entwicklung sicherer Software, Aachen, Germany pp. 501-514

1.5. Outline

The remainder of this dissertation is organised as follows: In Chapter 2, we present the background of this dissertation, including software safety challenges, traditional and modern safety analysis techniques and software verification and testing approaches. The state of the art is presented in Chapter 3. In Chapter 4, we presented our proposed approach for software safety engineering based on STPA including software testing and verification activities. Chapter 5 presents the automation support to the proposed approach. The tool support of the proposed approach is presented in Chapter 6. Chapter 7 presents the empirical validation, which introduces three cases studies on evaluating the application of the proposed approach: pilot case study and the two industrial case studies. In Chapter 8, we conclude the dissertation and present the future work.

Chapter 2

Background

> *It's hard enough to find an error in your code when you're looking for it; it's even harder when you've assumed your code is error-free.*
>
> — *Steve McConnell*

2.1. Software Safety Challenges

Software is an integral and increasingly complex part of modern safety critical systems (as shown in Figure 2.1). Therefore, it is essential to analyse software safety in a system context to gain a comprehensive understanding of the roles of software and to identify the software-related risks that can cause hazards in the system. Leveson [Lev91] noted that software by itself is not hazardous and cannot directly cause damage to human life or the environment; it can only contribute to hazards in a system context. Software can create hazardous system states through erroneous control of the system or by misleading the system operators when taking actions [Lev11]. Software has no random failures and it does not wear out like hardware components [Lev91; Jaf+91]. Flaws in software are systematic failures which stem from flawed requirements, design errors or implementation flaws [Lev91; Jaf+91]. System hazards related to software are caused by software flaws, software specification errors and uncontrolled interactions among different components forming the system, rather than failures of single components [Har10].

Ensuring the safe operation of systems requires that the potential risks associated with increased reliance on software be well understood, so that they can be adequately controlled. To develop safe software, therefore, we first need

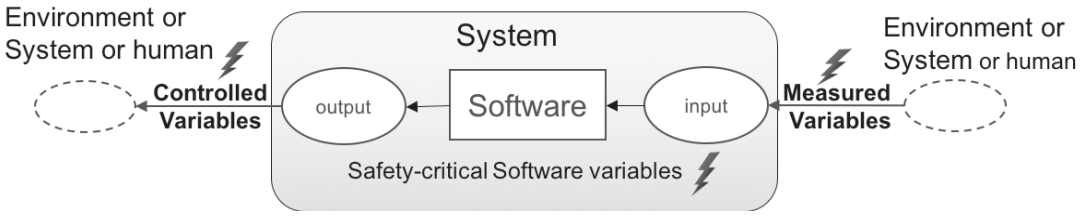

Figure 2.1.: Software is an integral part of system

to identify and analyse software-related hazards and the unsafe scenarios and develop the corresponding software safety requirements at the system level. To assure that these software-related unsafe scenarios cannot occur in a system, safety verification activities are required which include a demonstration of whether the software design and implementation meet those software safety requirements [NAS10]. However, the software safety requirements are written in natural languages. Therefore, to enable the software verification activities (e.g. testing and formal analysis), these requirements should be specified into a formal specification in Linear Temporal Logic (LTL).

2.2. Safety Analysis Techniques

Over the past seventy years, the most basic models of accident causation are the sequential models. The Domino model (sequential accident model) by Heinrich [Hei31] is one of the earliest accident causation models, proposed in 1931. The Domino model describes an accident as a chain of discrete events which occur in a particular temporal order [Fer88]. There are five safety factors which are addressed by the Domino model: 1) social environment, 2) fault of the human, 3) unsafe acts or conditions, 4) accident and 5) injury [Qur08]. Epidemiological accident models [Isk62] are meant to explain the accident causation in complex systems. These models are valuable because they provide a basis for discussing the complexity of accidents that overcomes the limitations of sequential models [Hol04]. The epidemiological models consider the events leading to accidents as analogous to the spreading of a disease. The epidemiological accident model can also be used to study causal relationships between environmental factors and accidents or diseases. An accident is conceived as the outcome of a combination

of factors in this model. Some of the factors manifest and some are latent, but they happen to exist together in time and space [Rau13]. A famous example of epidemiological models is the Swiss Cheese model: It was proposed by Reason in the 1990's [Rea90] and emphasises the concept of organisational safety and how protection barriers may fail. The Swiss cheese model [Rea97; RHP06] views accidents much like the spreading of disease and describes the combination of latent conditions present in the system for some time and their role in unsafe acts made by operators.

2.2.1. Traditional Safety Analysis Techniques

There are over 100 different hazard analysis approaches in existence [Eri05]. Many of these approaches, however, are not widely practiced. Fault Tree Analysis (FTA), Failure Mode and Effect Analysis (FMEA) and Hazard and Operability Analysis (HAZOP) are most commonly used by system safety analysts. These approaches are known as traditional hazard analysis techniques in the academic literature, which rely on accident causation models which are sequential or epidemiological.

2.2.1.1. FTA

The Fault Tree Analysis Approach (FTA) [Ves+81] was developed at Bell Laboratories in the early 1960's under a U.S. Air Force contract to analyse the Minuteman missile system. FTA is a top-down approach to identify critical failure combinations. FTA is based on the chain of event accidents model. It is widely used to discover design defects during the development of a system and to investigate the causes of accidents or problems that occur during system operations [LN05; Lev82; LH83b]. The input of FTA is a known hazard, failure or accident, and a design description of the system under analysis. The FTA process can be divided into four main steps: 1) identify the root node (hazard or accident or failure); 2) identify the combination of events or conditions that caused the root node and combine them by using Boolean logic operators; 3) decompose the sub-nodes until events determined are basic (leaf nodes); and 4) identify minimum cut sets which are the smallest sets of basic events that cause the root node to occur.

2.2.1.2. FMEA and FMECA

The Failure Mode, Effects and Analysis Approach (FMEA) [Mil49] was first introduced by the U.S. military for weapons systems in 1949 as a systematic, procedure for evaluating and discovering the potential failures, their potential cause mechanisms and the risks designed into a product or a process. By the early 1970s FMEA was used in civil aviation and the automotive industry [FME67]. FMEA helps to identify where and how the component might fail and to assess the relative impact of different failures. FMEA is, similar to FTA, based on the chain of events accidents model. FMEA is a bottom-up, structured, table-based process for discovering and documenting the ways in which a component can fail and the consequences of those failures. The input to FMEA is a design description of the system and component. The FMEA process can be divided into four sub-tasks: 1) establish the scope of the analysis, 2) identify the failure modes of each block; 3) determine the effect of each potential failure mode and its potential causes; and 4) evaluate each failure mode in terms of the worst potential consequences and assign the relative values for the assumed severity, occurrence and chance of detection to calculate the risk priority number. Ultimately, the analyst has to develop the recommended action required to reduce the risk associated with potential causes of a failure mode [LN05].

An extension of FMEA called FMECA (Failure Mode, Effects and Criticality Analysis) [FME67] was developed by reliability engineers to evaluate the effect of single component failures. FMECA is adopted and used as a hazard analysis in different domains such as space, nuclear and automotive industries.

2.2.1.3. HAZOP

The Hazard and Operability Study (HAZOP) [Tro68] was initially developed by imperial chemical industries in 1964 and published in 1974. HAZOP is a structured hazard analysis technique to identify risks and operability problems in a given system and develop appropriate safeguards to prevent accidents. HAZOP was originally developed to be used in the chemical industry to identify the potential deviations in chemical processes which can lead to accidents, however, it has been used to identify hazards in different systems in the different domains (e.g. computer systems, software systems) [McD+95]. HAZOP can be applied

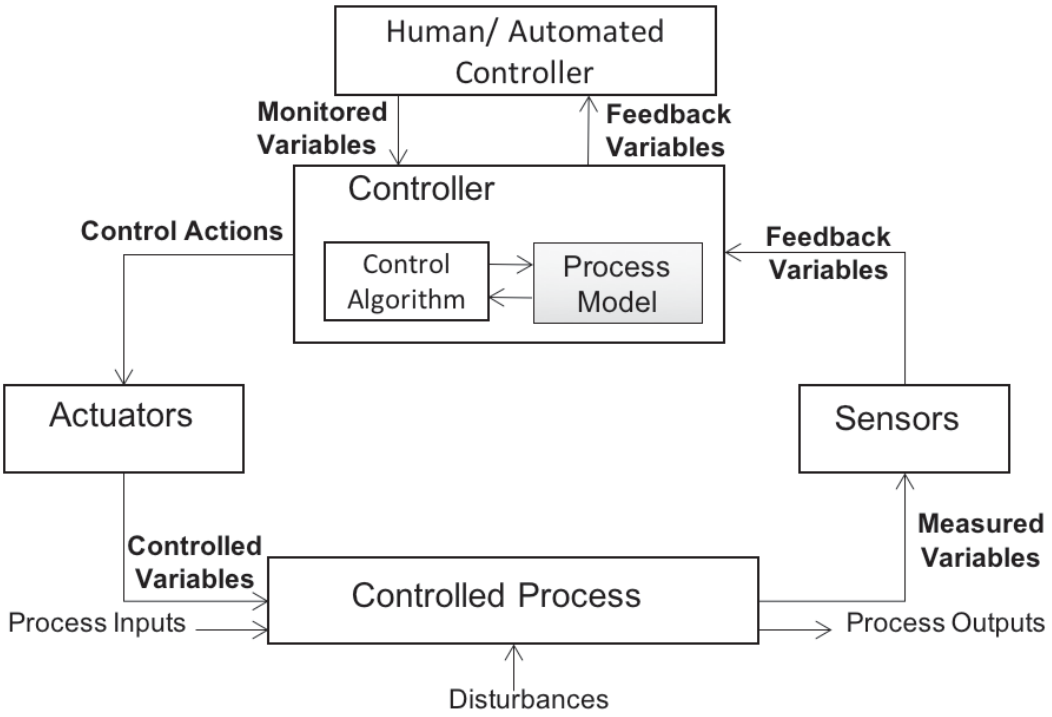

Figure 2.2.: A general feedback control structure

on existing system or during the system design phase before the system has been implemented. The HAZOP process provides guide-words combined with process parameters (e.g. flow, pressure, time, etc.) to help the analysts in identifying possible hazards in a system. These guide-words are used to systematically consider the possible deviations from normal operations of systems like “No", or “more" or “less" or “as well as " or “part of" or “Reverse" or “other than".

2.2.2. System-Theoretic Safety Analysis

The nature of accident causation has, however, become more complex over time. Twenty years ago, accidents causation theory was developed further to capture this increased complexity and a new class of models emerged based on a holistic and systematic approach [Lev04a]. Furthermore, the prevailing chain-of-failure-events models provide the basis for almost all of today’s hazard analysis techniques and the probabilistic risk assessment based on them. All of these analysis and design techniques focus on hardware component failures

and thus reliability theory [Lev11]. These methods assume that accidents are caused by component failures. However, they are not enough to explain accident causation in the more complex systems.

The development of accident causation models and safety analysis from formerly sequential models to systemic models shows the evolution of safety analysis for complex systems. We must emphasis that traditional analysis types, like FMEA or FTA have been designed for simpler systems than nowadays being created in the industry. The integration of technological, software system components stretches the limits of safety analysis. Therefore, new methods are needed which can actually cope with today's complex systems.

To overcome the limitations of the traditional hazard analysis, a recent countermeasure is to advance safety analysis techniques by system and control theory rather than reliability theory. The STAMP (System-Theoretic Accident Model and Processes) [Lev04a] accident model developed by Leveson, which uses system theory and treats safety as a control problem. Hence, it describes the system as a whole as opposed to linear cause effect relationships or epidemiological factors within the system. STAMP also continues corresponding hazard and accident analysis methods. Within this method, accidents are considered as results from inadequate enforcement of safety constraints in system design, development and operations. STAMP treats safety as a control problem rather than component failures. STAMP is based on system theory, which was designed to understand the structure and behavior of any type of system. In a system-theoretic approach, the system is seen as a set of control components which interact with each other (shown in Figure 2.2). This helps to create models of systems which cover human, technology, software, and environmental factors [Lev04a]. Therefore, STAMP considers accidents not only arising from component failures, but also from the interaction among system components. In other words, accidents occur when component failures, external disturbances and/or dysfunctional interactions among system components are not adequately handled by the safety control system [Lev04a].

The STAMP approach can be divided into two different analysis methodologies. While STAMP acts as an underlying theory, the methods STPA (System-Theoretic Process Analysis) and CAST (Causal Accident Analysis based on STAMP) are to be practically used for safety analysis. STPA is designed for safety analysis in the