Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 238 (2009) 71–77
www.elsevier.com/locate/entcs

Model-Based Safety-Cases for Software-Intensive Systems

Peter Braun[1] Jan Philipps[2]

Validas AG
München, Germany

Bernhard Schätz[3] Stefan Wagner[4]

Institut für Informatik
Technische Universität München
Garching b. München, Germany

Abstract

Safety cases become increasingly important for software certification. Models play a crucial role in building and combining information for the safety case. This position paper sketches an ideal model-based safety case with defect hypotheses and failure characterisations. From this, open research issues are derived.

Keywords: Safety case, model-based, structured argument, defect hypothesis, failure characterisation

1 Introduction

The proliferation of software-intensive technical systems has resulted in a growing need for methods to demonstrate their safety and reliability. The goal of such methods is to develop a *safety case* for a system – a line of argument that establishes safety and reliability properties from known properties of the components of the system.

Various approaches exist: Leveson et al. [7] describe an FTA-like approach to examine Ada programs, while Giese et al. [3] show how HAZOP-like safety analyses can be based on component and deployment diagrams of the UML. Within the ISAAC project [2], models of functional, geometrical and human aspects are

[1] Email: peter.braun@validas.de
[2] Email: jan.philipps@validas.de
[3] Email: schaetz@in.tum.de
[4] Email: wagnerst@in.tum.de

doi:10.1016/j.entcs.2009.09.007

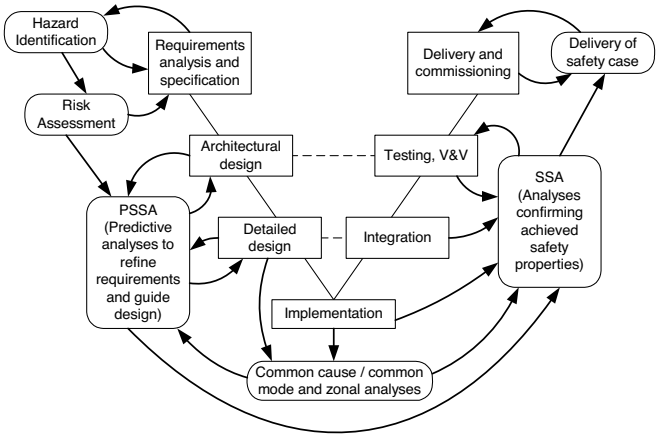

Fig. 1. Safety activities in a development process (Source: [9, p. 29])

integrated for safety analyses. Pumfrey [9] gathers a list of nine factors for success of safety analysis methods and goes on to develop two methods for dealing with mixed hw/sw systems.

In all these approaches the use of models plays a central role in the construction of a safety case. While earlier approaches are based on structured reviews of models, recently formal verification techniques have been applied for model analysis [1,3,5,2]. However, a systematic approach to the definition of those models is still uncommon. It is also an open issue how to justify the appropriateness of the underlying models for the safety case: Is it possible to derive all relevant hazards, system failures and component faults, and is it possible to reason about the causal chains that link them? In other words, we believe that the major open issue is how to reason about the *choice* of models, and not so much how to reason about the *properties* of the models.

In addition to this principal issue of the appropriateness of the models, we believe that there are a number of core success factors for building model-based safety cases:

- *Seamless integration into development processes.* It is not sufficient to merely perform a single safety analysis for certification of the final system – analyses with different focus play their role throughout system development, in order to clarify requirements, designs, and in general to improve both product and process (see Fig. 1).
- *Consideration of system, platform and environment.* It is not sufficient to examine models for the functional behaviour (even if they are augmented with fault models, as in [5]) of a system by verification or tests. Since hazards manifest themselves in the system's environment, the environment must also be modelled and included in the analysis. Since faults often are caused by the underlying computing platform, the platform must also be included; possibly, abstract user models may be needed to reason about operator errors and ways to avoid them or to deal with them. Note that in the development process these different models may well be constructed and analysed at different times: For instance, a prelimi-

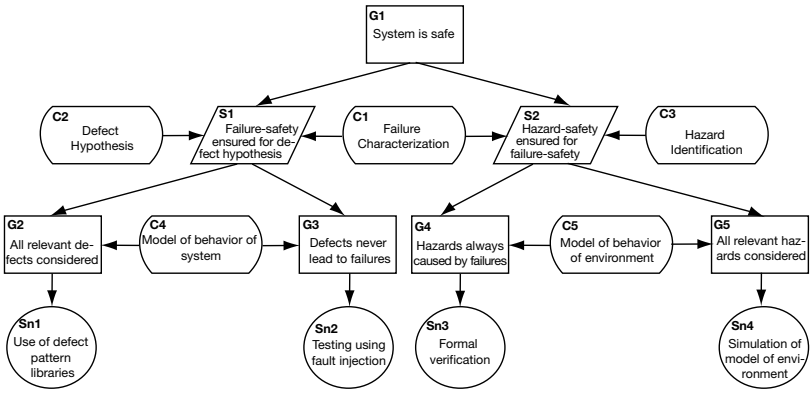

Fig. 2. A simplified example of a safety argument using GSN and context information from models

nary hazard analysis (corresponding to the top left activity in Figure 1) may need only rather abstract environment and system models.

- *Heterogeneous reasoning.* Finally, in order to cope with the realities of systems development practice, where different components – some newly developed, some legacy, some off-the-shelf – of different sources are combined, a compositional approach to the development of safety cases with a mixture of quantitative and qualitative reasoning is needed. While some properties may be demonstrated through rigorous verification or testing, others may be based on statistical reasoning (as in some applications of FTA) and empirical data.

In this paper we outline a research agenda for model-based safety cases that tries to give answers for some of these issues. In Section 2, we give a short overview over the argumentation behind a safety case and in Section 3 we look at the requirements on models and modelling languages that can support the building of safety cases. Section 4 lists some of the open research issues, and Section 5 concludes.

2 Safety Case

In general, a safety case is a structured line of arguments that shows that the system under consideration is safe. One of the difficulties is that a large variety of information needs to be combined to form this argument. A single safety assurance method is never able to show the complex issue of safety completely. Hence, formal verification, statistical testing, process conformance and other information must be integrated for a convincing argument.

A description technique that has proven to be useful for constructing safety arguments is the *Goal Structuring Notation* (GSN) [6]. It reduces some problems, such as ambiguity, of purely textual descriptions. An example safety argument using GSN is shown in Fig. 2. In this example the overall goal **G1** is that the system is safe. This is intended to be achieved by two strategies: **S1** is to ensure there are no failures in terms of deviations from the intended safety-critial functionality, and **S2** arguments by showing that there are also no hazards in the absence of failures. Hence, defect hypotheses as well as possible hazards must be identified. This is

depicted by the two constraints **C2** and **C3**. From the strategies, four refined goals are derived. **G2** expresses that all relevant defects need to be considered, which is shown by using defect pattern libraries in solution **Sn1**. **G3**, **G4** and **G5** describe further goals for failures and hazards, which are met by the exemplary solutions **Sn2**, **Sn3** and **Sn4**. Important for the following is also that there are several context informations like **C1**, the characterisations of failures, and models of the behaviour of the system (**C4**) and the environment (**C5**).

In summary, the example shows that a variety of information needs to be considered when constructing a safety case. From a methodological point of view, safety requirements and hazards need to be identified. However, this is not sufficient. It is also necessary to build models that form the context for more detailed arguments, for example about specific components. Moreover, the models must be able to handle deterministic as well as probabilistic information and need not only to include the system itself but also its environment.

3 Model-Based Safety Assurance

Model-based development is becoming a common-place approach to embedded (control) software construction; models, which describe the nominal functionality, are found in form of plant models, e.g., for in-the-loop testing, as well as controller models, e.g., for production-code generation. In some areas, the generation of production code from models is already state-of-the-art.

However, without specifically addressing the issue of safety-cases, models of the *nominal functionality* describing the system under development are not sufficient for the analysis of fail-safe behaviour. To apply the models used in a model-based development approach to the construction of a safety-case,

- a model of the system must be derived to *describe the effective functionality, including nominal as well as defect-affected behaviour.*
- a model of the environment must be constructed to *explicitly model the assumptions about the behaviour of the environment.*
- explicit hypotheses of defects – expressed solely in terms of the system – must be provided to *avoid mistaking fail-safe behaviour of the model for fail-safe behaviour of the system.*
- explicit characterisation of failures – expressed solely in terms of the interface between system and environment – must be provided to *allow identifying deviations from the intended behaviour in the interaction between the system and its environment.*
- explicit identification of hazardous situations – expressed solely in terms of the environment – must be provided to *allow describing hazards in terms of the controlled environment rather than the controlling system.*

Therefore, when extending the construction of the nominal functionality of a system to the construction of a safety-case, the issues of *defect hypotheses*, *failure characterisations*, as well as a *hazard identifications* must be addressed in the safety case,

making implicit assumptions explicit.

A failure characterisation describes unintended functionality of the system, which may potentially lead to hazardous situation. In the safety-case it is used to link the line of argument between the environment model and the system model.

A defect hypothesis describes how the nominal functionality of a system is altered to reflect the possible occurrences of faults. As defects may be introduced in different parts of a system, different defect hypotheses are needed, e.g, to reflect defects caused by a deviation from the intended functionality of the system platform (i.e., hardware defects), or faults caused by a deviation from the intended control functionality (i.e., design defects). For practical application, defect hypotheses are often described in form of fault patterns, e.g., in form of intermittent occurrences of value changes to reflect influences of alpha radiation.

A hazard identification describes states of the controlled environment to be avoided by the nominal functionality of the system. However, as noted by [4] and [8], especially in the context of embedded or software-intensive systems, the functionality of the system is often only adequately expressed over the part of the environment, which is only indirectly controlled or monitored by the system under development. Therefore, to describe the achievement and failure of functionality, a hazard identification independent of the model of the system is needed.

Separating these three properties as well as using separate models of system and environment reduces the complexity of the overall argument and minimises the risk of constructing inadequate lines of arguments.

As shown in Figure 2, based on these five models, now a formalised and standardized line of argument can be constructed to *show the absence of hazardous situations based on the defect hypotheses*. This line of argument is constructed in six steps:

(i) The defect hypothesis is validated to ensure that all relevant defects are included.

(ii) The hazard identification is validated to ensure that all relevant hazards are included.

(iii) The model of the system is verified with respect to conformance of the defect hypothesis, i.e., the *system does not constrain the possibility of such defects*.

(iv) The model of the environment is verified with respect to conformance of the hazard identification, i.e. the *environment does not constrain the possibility of such hazards*.

(v) The model of the system is verified with respect to the avoidance of failure situations under the given defect hypothesis.

(vi) The model of the environment is verified with respect to the avoidance of hazardous situations under the given absence of failures.

For each of these steps, different solution techniques can be applied. E.g., applying defect patterns for the identified defects can be used in step iii, while a simulation of the environment model can be used in step iv. Furthermore, steps v and vi

themselves are complex verification goals, requiring appropriate sub-goals and sub-solutions.

4 Open Research Issues

Based on the ideas of the previous section, we briefly state the most important open issues that need to be addressed.

Safety Case. To start with, how can safety requirements and possible hazards be effectively elicited? When is the safety case complete? A strong structure needs to be provided for the safety engineering in the form of safety patterns and reusing existing parts of safety cases. Moreover, the approach has to be mapped to existing (an potentially new) standards to allow certification.

Models. In general, the question is how can be assured that the built models are suitable for safety analysis. The quality of these models is decisive for the whole safety case. There is a plethora of questions: On what level of granularity are the models built? What are suitable interfaces between components so that complex causal chains can be analysed? What are proper defect hypotheses and failure characterisations? Too much detail is not manageable, not enough detail leads to omissions.

Supporting Methods. Finally, various and diverse methods need to be used for arguing in the safety case. Hence, quantitative and qualitative as well as deterministic and probabilistic methods need to be used and integrated in the argument. When is formal verification needed and feasible, when are other arguments necessary and sufficient?

5 Conclusions

Safety cases become increasingly important for software-intensive systems. The current state-of-practice, FTA and FMEA, are not sufficient for the enormously complex and interconnected modern systems. Hence, we need suitable models, not only of the system but also of its environment, especially its users. These models are needed as basis for formal verification. Moreover, they feed into the complete and structured argument in a safety case. We described how model-based safety analysis and safety case development should ideally look like and derived a set of open research issues that need to be addressed. We are currently in the process of discussing with industrial partners first steps to tackle these issues.

References

[1] M. Bozzano et. al. ESACS: an integrated methodology for design and safety analysis of complex systems. In *Proc. ESREL 2003*, pages 237–245, 2003.

[2] O. Akerlund et. al. ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects. In *Proc. ERTS 2006*, 2006.

[3] Holger Giese, Matthias Tichy, and Daniela Schilling. Compositional Hazard Analysis of UML Components and Deployment Models. In *Proc. 23rd International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, volume 3219 of *LNCS*. Springer Verlag, 2004.

[4] Michael Jackson. *Software Requirements and Specifications*. Addison-Wesley and ACM Press, 1996.

[5] Anjali Joshi, Steven P. Miller, Michael Whalen, and Mats P.E. Heimdahl. A proposal for model-based safety analysis. In *Proc. 24th Digital Avionics Systems Conference*, Oct 2005.

[6] Tim Kelly and Rob Weaver. The goal structuring notation – a safety argument notation. In *Proc. DSN 2004 Workshop on Assurance Cases*, 2004.

[7] Nancy G. Leveson, Stephen S. Cha, and Timothy J. Shimeall. Safety verification of ada programs using software fault trees. *IEEE Softw.*, 8(4):48–59, 1991.

[8] D. Parnas and J. Madey. Functional Documents for Computer Systems. *Science of Computer Programming*, 1(25):41–61, October 1995.

[9] David John Pumfrey. *The Principled Design of Computer System Safety Analyses*. PhD thesis, Department of Computer Science, University of York, 1999.