Towards Systematic Software Reuse in Certifiable Safety-Critical Systems

Mikael Åkerholm[1,2], Rikard Land[1,2]

[1]Mälardalen University, School of Innovation, Design and Engineering, Västerås, Sweden
[2]CC Systems, Västerås, Sweden
rikard.land@mdh.se, mikael.akerholm@cc-systems.com

Abstract. Safety-critical systems and subsystems are often developed as a new generation of a previous system, or as a variant of a system already developed and put into operation. However, in our experience, even in such cases, where large parts of the systems are actually reused, organizations implement very much the same heavy processes as for new development. This is partly because during a safety assessment the evidence needed to motivate the desired level of system safety calls for coherent documentation of the complete system development project. We believe the reuse process can be adapted to be more efficient, while still being compatible with safety standards, by adopting a state-of-the-art structured component-based reuse approach incorporating the specific safety activities that the standards mandate. This position paper outlines our planned research, which will consist of two parts: the first part is an interview study of industrial cases, in order to identify good practices to employ and pitfalls to avoid. In the second part we will implement the most promising practices in suitable industrial projects for evaluation.

1 Introduction

Certifying safety-critical software systems to be compliant with required levels of safety stipulated by different safety standards, e.g., [2, 4, 5, 6, 7, 8, 9, 10], becomes more and more common in the industry. This is due to increased legislation requirements (e.g., the EU Machinery Directive, 2006/42/EC) and market expectations. The approach to achieve safety for software in the standards is qualitative. Implying that the safety standards are general and cover many practices in the systems entire life-cycle, e.g., how requirements are handled and traced in the development process, how the software is designed and programmed, how the potential safety hazards are analysed and handled, what compilers and development tools that are used, how well the tests covers the code, and how the system is planned to be maintained. The benefit is clearly that the standards raise consciousness of different safety related aspects of the software; we can perhaps also expect a minimum level of safety of a certified software system given that the safety assessment and certification is fair.

The problem from the viewpoint of the software manufacturers is that the standards are extensive and require much documentation to provide evidence how the requirements of the standards are met. At the same time the standards are very restrictive when it comes to usage of the latest development of software engineering tools and approaches, e.g., the most general standard IEC61608 [4], which many other domain specific standards refer to, does not allow any object oriented programming languages yet. Furthermore, reuse is often not explicitly treated and the required processes focus on development of new software. This seems to result in expectations from safety assessors issuing safety certificates that the documentation shall reflect that all the software have been developed from scratch.

The advisory circular AC20-148 [3] from the avionics domain is an exception where reuse and safety is treated together. It provides guidelines how to be able to develop software components that is certifiable according to the avionics safety standard for air-born equipment RTCA DO-178B [10].

However, it also gives good guidance for any development of reusable components where safety is important. The advisory circular takes the viewpoint that all assumptions and requirements on the component's usage must be fully specified, and stresses that a lot of verification efforts are left for the system integrator to prove that the environmental assumptions made by the component developer are valid in the context where the component is used. The inward sense of AC20-148 is clearly to put so high requirements on re-verification activities for integrators that safety is not compromised within the new context. However, the advisory circular gives no guidelines on how to e.g. organize documentation or design the software to achieve this as efficiently as possible.

In this position paper we outline the main challenges we intend to address in our future research. The goal is to define an efficient state-of-the-art structured component-based reuse approach [11] for safety-critical systems. Specified pragmatically, the goal is to specify a high-level process model and identify more detailed practices that enable efficient development (both *of* reusable components and *with* reusable components), while not compromising the strict requirements of the safety standards and the advisory circular AC20-148.

We intend to perform two main types of studies in our work, interviews and case studies. Interviews will mainly be held with practitioners who have been involved in safety-critical projects, in order to collect good practices and lessons learned. As a complement, we intend to also interview safety assessors about their view about past experiences of software reuse. Case studies will consist of us participating in, or closely studying specific industrial projects, where some practices considered having the highest potential in the particular project are implemented. We will then evaluate the effect of these practices.

2 Process Model

Fundamental to all safety standards is that safety of software systems has to be addressed during all life-cycle phases, from early analysis, throughout design, implementation, production, maintenance and decommissioning, also taking into account people aspects such as training of developers as well as of users. All safety standards therefore specify a reference process which must be followed, where certain tasks are performed before others in an integrated project. For example, hazards have to be identified early, and the design has to be designed to tolerate those hazards; safety cannot be assessed on basis of only the delivered artefact. The standards do discuss the usage of components treated as "proven-in-use"; however, the standards are not very clear about the absolute requirements. Proven-in-use tends to be commonly understood as something that is applicable to standard C libraries shipped with certified compilers rather than software developed by companies certifying vehicles and machines. In industrial practice, subcontractors therefore have to work closely together with system integrators. Even if a subcontractor, or an internal department, intends to reuse some software component, it has to be re-fit by, e.g., making all assumptions explicit, reconstruct the trace from requirements down to design, implementation, and tests of the component, in order to fulfil the standard. The consequence is that if a safety standard is followed in the most straight-forward way, it requires a lot of rework which (we believe) should be possible to reduce by applying a state-of-the-art component-based development process.

Clearly, this call for a process model which has the following two properties: 1) It provides a clear interface between system development and component development. "Interface" should here be interpreted in the broadest possible sense, including, e.g., how documents are organized and divided between system development and component development to accommodate different parts of the safety analysis and final motivations that the safety goals are met. 2) It is compatible with the process model of safety standards, which simplified can be described on a high level as a V-model

with many detailed verification requirements. With “compatibility” we intend nothing less than acceptance by safety assessors in real industrial cases. We are aware that the details differ depending on which standard is being addressed, and the result of our work might need to be adjusted to be applicable in a certain domain.

Fig. 1 outlines such an initial process model. It illustrates the essentials of the process models of important safety standards [2, 4, 5, 6, 7, 8, 9, 10], while also incorporating the characteristics of component-based development process [1]. Although it is described at a very high level and only reflects the very initial phase of our research. The main feature we want to point out is that some activities and artifacts are specified twice, and one of the important challenges is to specify which sections should be specified as part of the system development or component development, respectively. For example, how extensive could and should a hazard analysis be performed at system level, in order to interface well with the hazard analysis performed as part of the component development?

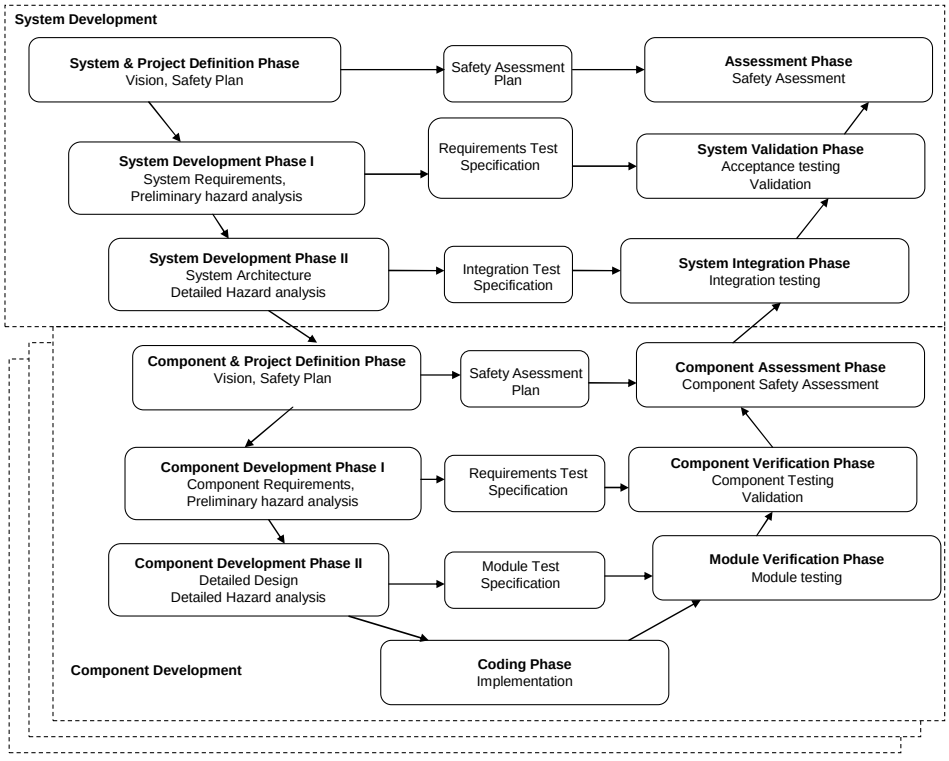

Figure 1. A component oriented process derived from the V-models imposed by safety standard

3 Specific Challenges

The following paragraphs present some specific challenges we find particularly challenging or having a great potential for improvement by considering carefully. The list is by no means intended to be exhaustive, but is a selection of issues discussed in safety standards [2, 4, 5, 6, 7, 8, 9, 10] and in particular the advisory circular AC20-148 [3].

Component interface - Component developers need to specify the component's interface, and the system integrator needs to ensure that the component and the rest of the system interact correctly. "Interface" should be interpreted in a wide sense; the advisory circular AC20-148 specifies e.g. configuration parameters, restrictions on tools, memory and timing requirements, external resources required by the component, communication mechanisms to software and hardware, input and output variables, and access mechanisms. In addition to this, component developers also need to specify all assumptions concerning how integrators will use the components, and system integrators have to validate that all the assumptions the component developer made are met.

Component abstraction – Defining components so that they can be successfully reused with high benefits in many applications is non-trivial. A big software component would imply a higher benefit associated with reuse, however, a big software component is also typically very information rich which makes it hard to reuse in different contexts. For safety-critical systems, functions not used in an application are typically not allowed to be part of the application, thus, some mechanism for deactivating code would be required. This must be considered during the development of the component. Moreover, the deactivation mechanism itself must be target for certification. The developer and users must clearly identify any information about deactivated code and the usage of associated deactivation mechanisms.

Activities left for the integrator - Component developers need to specify activities remaining for the integrator to perform when using the component in a specific system context. In particular this involves (re-)verification of the component each time it is reused in a new system. Examples of verification activities left for the integrator according to AC20-178 are data coupling analysis, control coupling analysis, timing analysis, memory analysis, stack analysis, software integration testing, requirements-based test coverage, hardware-software integration testing, robustness testing of component functions, including safety and protection features, and partitioning and other protection mechanisms for integrity validation. Again, the challenge is to make it easy to redo all this verification, and will likely require solutions including both documentation and verification environments.

System level traceability - The integrator must address and maintain traceability between the component, the system software, and the system. The development of a component does not follow from the top-down system development process, so the design decisions and assumptions made for the component have to be revisited each time it is being reused in a new system context. To be efficient, all these decisions and assumptions must be packaged so that it is easy for the integrator to establish a link/connection/motivation that the system requirements are actually met satisfactory by using the component.

Certified or certifiable - There is a difference if the goal is a component certified (for reuse without modification) or a component developed to be easily certifiable with little effort. In the first case, the component has to be general enough (which in itself to some extent involves the challenge of component abstraction). If aiming for easy modification for integration in a new, certifiable system, the focus is more on making it extensible, including organizing all safety-related documentation, traceability, etc. for easy adoption and integration into a new system context. This choice influences how the component development is organized the first time. Due to the nature of safety and safety assessment, to be successful all development activities and artifacts will have to be considered in an overall reuse approach: requirements, analyses, documentation, design models, verification, safety case, etc. We believe there is no silver bullet (universal, single solution) but that many small steps/practices are needed in order to accomplish efficient reuse.

4 Conclusions

We have presented the main challenges we plan to address in our planned research project. In summary we seek an efficient state-of-the-art structured component-based reuse approach suitable for certifiable safety-critical systems. Important is a high-level process model compatible with the process models mandated by safety standards, and to identify more detailed practices that enable efficient development both *of* reusable components and *with* reusable components. Relating and motivating processes and practices with respect to main safety standards will be a crucial part in order to get impact in industry, and enable opportunities to validate the work in practice and get acceptance of, e.g., safety assessors in real projects. Thus, we hope to propose efficient ways to address at least some of the issues discussed in this position paper, and validate these scientifically.

We also hope that we can contribute to an increased awareness and interest in the research community of these practical aspects of safety certification.

Acknowledgements

This work is supported by the Knowledge Foundation (KKS), and the Swedish Foundation for Strategic Research (SSF) via the strategic research centre PROGRESS.

References

[1] Crnkovic, I., Chaudron, M., Larsson, S.: Component-based Development Process and Component Lifecycle. In : International Conference on Software Engineering Advances (ICSEA'06), 2006

[2] European Committee for Electrotechnical Standardization (CENELEC), standard EN50128, Railway applications. Communications, signaling and processing systems. Software for railway control and protection systems, 2001

[3] Federal Aviation Administration (FAA), advisory circular AC20-148, Reusable Software Components, 2004

[4] International Electrotechnical Commission (IEC), standard IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systems, 2005

[5] International Electrotechnical Commission (IEC), standard IEC 62061, Safety of machinery - Functional safety of safety-related electrical, electronic and programmable electronic control systems, 2005

[6] International Electrotechnical Commission (IEC), standard IEC 61511, Functional safety - Safety instrumented systems for the process industry sector, 2004

[7] International Organization for Standardization (ISO), standard ISO13849, Safety of machinery -- Safety-related parts of control systems, 2006

[8] International Organization for Standardization (ISO), standard ISO15998, Earth-moving machinery -- Machine-control systems (MCS) using electronic components -- Performance criteria and tests for functional safety, 2008

[9] International Organization for standardization (ISO), standard ISO26262, Road vehicles -- Functional safety, under development

[10] Radio Technical Commission for Aeronautics (RTCA), standard RTCA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992

[11] Szyperski, C.: Component Software 2nd edn. Addison-Wesley, 2002