Research on software systems dependability at the OECD Halden Reactor Project

SIVERTSEN Terje[1], and ØWRE Fridtjov[2]

1. Institute for Energy Technology, OECD Halden Reactor Project, Post Box 173, NO-1751 Halden, Norway (Terje.Sivertsen@hrp.no)
2. Institute for Energy Technology, OECD Halden Reactor Project, Post Box 173, NO-1751 Halden, Norway (Fridtjov.Owre@hrp.no)

Abstract: Two central issues related to software systems dependability are those of safety integrity and safety demonstration. A proper understanding of these two issues are important for the selection of processes, methods, techniques and tools to be used in the different life cycle phases of the software. Following a brief discussion on the concept of software safety integrity and its relationship to software systems dependability, this paper gives an introduction to research problems addressed by the OECD Halden Reactor Project within this area. The paper concludes with a discussion on the important role of safety demonstration in this context.
Keywords: software systems dependability; safety integrity; safety demonstration

1. Introduction

The OECD Halden Reactor Project (HRP) is a joint undertaking by national nuclear safety organizations in 18 different countries, sponsoring a jointly financed nuclear directed research programme under the auspices of the OECD - Nuclear Energy Agency.

The organizations participating in the HRP represent a comprehensive cross section of the nuclear community, including licensing and regulatory bodies, vendors, utilities, industry companies and research organizations.

The two main research programs generate key information for safety and licensing assessments. The *Fuels and Materials Research programs* aim at providing: i) Basic data on how the fuel performs in commercial reactors, both at normal operation and transient conditions, with emphasis on extended fuel utilization, and ii) Knowledge of plant materials behavior under the combined deteriorating effects of water chemistry and nuclear environment.

The *Man-Technology-Organization* (*MTO) research programme* aims at providing: Advances in human performance, human factors, human-machine interfaces and interaction, visualization technologies, computerized surveillance systems and *software dependability issues*, in support of development and licensing of new and upgraded control rooms.

The overall objective of the HRP's research programme on software systems dependability is to contribute to successful development, assurance and deployment of high integrity software within the nuclear sector. The Halden Project activities within this area concentrate on processes, methods, techniques and tools for the different life cycle phases of software important to safety. The activities aim at providing lessons learned and recommendations on the selection and use of means that are effective with respect to providing software which meets the demands for safety integrity placed upon it. In this sense, the programme is addressed to all parties involved in the development, assessment, approval, operation and maintenance, including software vendors, safety authorities and utilities.

The Halden Project work is based upon requirements and recommendations in relevant international standards and guidelines, as well as needs identified by stakeholders. The research programme aims at improving the knowledge on how to best implement these requirements in real projects. This involves the use of principles such as top-down design methods, modularity, verification, validation, assessment, configuration management and change control, and the appropriate consideration of organisational and personnel competency issues. The programme involves both in-depth and in-breadth research related to these principles, with the aim or providing lessons learned and recommendations that help the different parties develop, operate and maintain software that is

Received date: May 23, 2011
(Revised date: June 6, 2011)

safe to put into use and that preserves its level of safety integrity and dependability throughout its lifetime.

In order to extend the basis for recommendations and lessons learned within this area, the programme benefits from organised knowledge transfer from the use of software in control and protection systems in other industry sectors such as petroleum, air traffic control and railway signalling.

Following a brief discussion on the concept of software safety integrity and its relationship to software systems dependability, this paper gives an introduction to research problems addressed by the Halden Project within this area. The paper concludes with a discussion on the important role of safety demonstration in this context.

2. Software safety integrity

A safe system is a system that is free of undesired behaviors that can lead to system hazards. The system must be able to perform its function under an acceptable level of risk. Safety of a software based system does not only concern the behavior of the software and hardware, but also human factors related to the use of and interaction with the system. One example is the external interfaces of a system that supervises a critical process. These interfaces must be designed in such a way that the operator gets a correct and adequate picture of the supervised process so that it always can be operated in a safe manner.

Due to the nature of software, many current standards and guidelines for development of software important to safety focus on the methods to be used to avoid systematic errors and thereby provide software which meets the demands for safety integrity. This is commonly done by adopting some notion of *safety integrity level* (SIL), and giving different sets of requirements relative to these levels.

The concept of SIL, or some variant of this, is adopted for software based systems in several industrial sectors and is supported by standards developed for these sectors. Important standards in this context are IEC (International Electrotechnical Commission) 61508 [1] (process industries, including petroleum) and EN (European Norm) 50126 [2], EN 50128 [3], EN 50129 [4] (railway applications). These standards give requirements to the processes, methods, techniques and tools to be used to implement the system functions allocated to software: Higher safety integrity levels means more rigid requirements. By adopting safety integrity levels as a key concept, these standards tend to move the focus from the possible (or impossible) quantification of software dependability to a more qualitative approach based on the use of accepted combinations of methods in the design and implementation of the software.

A concept similar to safety integrity levels, called *design assurance levels*, has for a long time been adopted for airborne systems and been supported by the standard DO-178B [5] (the prefix "DO" designates documents from RTCA, Radio Technical Committee on Aeronautics). The nuclear industry has so far not adopted the SIL concept directly, but follows in practice the principle of differentiating the requirements depending on the class of application. A notable example is the standard IEC 60880 [6], which gives requirements to systems performing category A functions (*i.e.* functions that play a principal role in the achievement or maintenance of nuclear power plant safety to prevent design basis events from leading to unacceptable consequences, or whose failure could directly lead to accident conditions which may cause unacceptable consequences if not mitigated by other category A functions [7]). Whether the nuclear industry will arrive at a consensus regarding a common approach to the use of safety integrity levels remains to be seen. Currently, the practices in the different countries seem to vary more than what is the case *e.g.* within European railway. An important factor here is to which extent the standards are designed to facilitate cross acceptance on the basis of the safety approval and acceptance made by another safety authority. Within the European Union, such a harmonization of the safety approval processes in the different member countries is considered an essential part of the development of the common market.

The use of safety integrity levels reflects a perspective where functions and associated safety requirements are allocated as part of a risk-based approach to the establishment of the overall system requirements. According to this perspective, the demands for

software safety integrity are an outcome of the processes employed at the system level. It follows that the software requirements are based on the functions and associated safety requirements allocated to the software as part of this process. As a consequence, the demands for software safety integrity constitute an input to the software development process. Since this input comes from a process at the system level, it focuses on the functions to be implemented in software, and not on the software itself. It is however not enough to analyze the primary system functions: The establishment of safety requirements needs to investigate the possible sequences of state transitions and identify the states the system should never be able to enter.

This way of integrating the software development process in the overall system development process has the important benefit of giving a clean interface between these two processes. One of the practical consequences is that the software process should not determine the safety integrity of the functions implemented – this need to be determined at the system level.

The demands on software safety integrity should reflect the risk (consequences and their probabilities) associated to possible software errors: Higher risk means higher demands on software safety integrity. This can be concretized by defining a relationship between tolerable hazards rates and software safety integrity levels, as is done *e.g.* in the process industry standard IEC 61508 and the railway specific standards EN 50126, EN 50128 and EN 50129. By using the safety integrity levels as a guide to the methods to be used, this approach connects overall system risk to the concrete methods, techniques, processes and tools used for developing the software involved. The actual process of determining the safety integrity levels on the basis of the system risk needs however to be defined specifically for the relevant industrial sector or application area.

In general, the following can be considered as activities at the system level, before the requirements to the software are established: Risk analysis, including the identification and assessment of hazards, leading to the risk acceptance criteria, followed by identification of the necessary risk reduction, the establishment of the system safety requirements, the selection of a suitable system architecture, and the allocation of safety integrity levels to the different subsystems and components. Among the outcomes of this process at the system level are the required software safety integrity levels, which constitute inputs to the activities at the software level.

A consequence of this procedure is that the safety functions, and their safety integrity levels, are determined and allocated to software at the system level. In this sense, this also determines the earliest point of departure for the software development, at least as far as safety is concerned.

3. The research problems

In spite of the fact that many software engineering issues are still subject to research and development, the field has reached a general consensus on much of what characterizes an adequate engineering process for software important to safety, such as:

- The software should be specified in terms of a software requirements specification which is traceable back to the system requirements and forwards to its implementation.
- The software should be designed through the use of methods which facilitate modular software architecture, built up by the composition of well-defined components (with well-defined interfaces).
- The software should be developed through a sequence of verifiable steps and phases.
- The software and its development process should be documented in a way that facilitates validation, safety demonstration and independent safety assessment.

Many more examples could have been mentioned as additional evidence to the emergence of a common understanding between different industrial sectors about the existence of some principles that govern the production of software important to safety. This is partly a result of, and partly a motivation for, technology transfer between the different sectors.

These principles are important because they provide important guidance to the recommendation and selection of the processes, methods, techniques and

tools to be used in the software process. Due to their large number and variation however, it is practical to divide these into the types of activities they are meant to support. Traditionally, this has typically been done by defining some life cycle model (normally a variant of the waterfall model), and associating the different means to the phases of this model. Notwithstanding the intention of allowing also other life cycle models, standards and guidelines employing this way of presenting their requirements and recommendations have contributed to the widespread misconception that the software development needs to follow this model. Instead of relating their requirements to development phases, standards should therefore relate their requirements to the types of activities performed (testing, verification, *etc.*). In this way, it becomes easier to use the same standard for development processes based on different life cycle models.

In the following subsections, some of the means adequate for the development, assurance, approval and deployment of software important to safety are presented. For each of these groups of activities, a discussion is given on some of the questions being subject to research at the OECD Halden Reactor Project.

3.1 Software development

Based on the specification of system and safety requirements, the development of software starts by establishing the software requirements and culminates with the final acceptance of the software. Important issues related to software development include how to describe a complete set of requirements for the software, meeting all system and safety requirements, and how to develop a software architecture that achieves these requirements, to identify and evaluate the significance of hardware/software interactions for safety, to achieve software which is analysable, testable, verifiable and maintainable, and to demonstrate that the software and the hardware interact correctly to perform their intended functions.

An important activity in the early phases of the software development is the elicitation of the software requirements from the different inputs to the requirements management process. Standards and guidelines typically put more emphasis on the structure and quality of the requirements specification document than in the actual elicitation of the requirements. It is commonly agreed which types of requirements should be covered (functionality, maintainability, *etc.*), how this should be expressed (complete, verifiable, *etc.*), and so on, but little is typically required of how this is achieved, in particular how to elicitate a complete set of requirements.

Naturally, the question of completeness is in most cases difficult to answer: When to stop searching for more requirements? There is no general answer to this question, and any general criterion for the "completeness" of the requirements would probably lack a scientific basis. The recommendation and selection of methods to facilitate the requirements elicitation process would therefore have to be based on some evidence on how well the use of these methods can reduce the risk of loosing essential requirements. In any case, the software should be "designed for change", in the sense that possible new or modified requirements emerging later in the software lifecycle can be implemented without compromising the software integrity. This should however not be used as an excuse for relaxing the importance of having a "complete" set of requirements from the beginning.

Literature on requirements elicitation typically mentions methods such as interviews, scenarios, requirements reuse, *etc*. All of these are important and useful means for elicitating requirements. An important question is however how to better utilize the intended relationship between the inputs to the requirements elicitation process and the actual requirements. Concretely, the fact that many of the requirements typically can be considered some kind of traceable "implementation" of requirements at the system level indicates that there is some potential in utilizing this relationship to improve, or possibly even automate, parts of the requirements elicitation process. While there certainly are cases where this is possible, an important research question is whether it is possible to formulate such an approach in terms of a well-defined method, including the criteria which need to be fulfilled in order to make the method applicable and of course the pragmatics related to the use of the method. An adequate research method

would be to demonstrate the approach on some relevant examples, extract some essential, case-independent characteristics of the examples, formulate a method on basis of these characteristics, and validate the method on other, relevant examples.

Another research problem related to development is the question as to which processes, methods, techniques and tools are the most effective for generating design solutions appropriate for the required safety and dependability. In particular, there is a need for guidance on the safe use of advanced technologies like adaptive control, multi-core processors and field programmable gate arrays (FPGAs). Such a research problem would go beyond the requirements and recommendations typically given in standards and guidelines, which need to be generic for all software within their scope. The research problem is twofold:

Firstly, there is a need for guidance on the selection of appropriate design solutions. That is, given the requirements to safety and dependability, which design solutions provide the best possibilities with respect to satisfying these requirements, demonstrating that they indeed have been satisfied, and supporting the assessment needed for the approval of the software for its intended use.

Secondly, there is a need for guidance on the use of given technologies. In many cases, the supplier prefers some particular type of technology to implement their software. There could be many different reasons for such a preference, such as the need for adaptivity, low price, high performance, etc. The reasons may vary, but the situations can be compared: Instead of selecting the most appropriate design solution, the question is how the required safety and dependability can be achieved with a given technology. Guidance is needed on how to design safe and dependable software systems with this technology, how to demonstrate that it is fit for its purpose and safe to put into use, and how to facilitate the assessment.

3.2 Software assurance

For software important to safety, important concerns are how to assure that the software fulfils the requirements, is safe to put into use, and otherwise is fit for its purpose. Important issues related to software assurance include how to ascertain the behaviour or performance of software, to ensure that output items of a specific development phase fulfil the requirements and plans with respect to completeness, correctness and consistency, to demonstrate that the processes and their outputs are such that the software fulfils its requirements and is fit for its intended application, to ensure that the software performs as required, preserving the software safety integrity and dependability when modifying the software, and to ensure that potential failures of tools do not undetected adversely affect their output in a safety related manner.

The different assurance activities often involve some kind of analysis. They have in common the need to ascertain the behavior or performance of the software through detailed examination of its architecture and components. An analysis of software important to safety typically takes the form of careful examination of the software or software component, and its associated documentation, with the aim of reaching a conclusion on its dependability and safety on basis of its design. Of special importance is analysis concerning the ability of the software to meet its safety and dependability requirements in the event of systematic failure. When the object of analysis also involves hardware (*i.e.* an electronic system), the analysis has to cover both random and systematic failures. Obviously, analyzing the effects of systematic faults is restricted by the limited possibilities in actually identifying the systematic faults, which indeed is some of the nature of these faults. Of this reason, the analysis usually considers the effects of failed software functions (typically at the component level), without any differentiation between the possible systematic faults.

A research problem related to failure analysis is how to effectively address both product and process aspects in the analysis, covering faults introduced in the technical design as well as errors made in the life cycle activities. Of particular concern are faults that have a potential for common cause failures. There is a need for recommendations on how the failure analysis can be improved through optimal combinations of description and analysis techniques reflecting all

relevant viewpoints. In particular, the analysis of common cause failures needs to analyze not only the software or the electronic system as a product, but also the processes followed in its development. An important source of errors with a potential for common cause failures is the lack of sufficient diversity in the early phases of the development: If the error is introduced already in some common specification, then a possible diversified design, with diverse design teams, methods, languages and tools is not necessarily enough to discover the potential for a common cause failure.

An important aspect of the software assurance activities is how they best can contribute to the approval and acceptance of the software for its intended use. Basic to the approval processes in many countries and industries is the provision of a safety case, *i.e.* the documented demonstration that the product complies with the specified safety requirements. The safety case forms part of the overall documentary evidence to be submitted to the relevant safety authority in order to obtain safety approval of a product.

A research problem related to safety cases is how to support the development and assessment of a safety case and its supporting documentation for software based systems, including the assessment of tool automated processes, making optimal use of probabilistic and analytical assessment methods. Such a support can be secured in terms of methods and tools designed to support the developer and the assessor in respectively documenting and assessing the necessary evidence, providing assistance for checking that relevant questions have been covered, for checking that the argumentation is complete, correct and consistent, and for following up identified deviations and defects in the safety argumentation and documentation.

3.3 Software approval and deployment

In order for a safety critical system to be put into use, the safety demonstration needs to be accepted by the relevant safety authority through a formal approval process. A successful deployment of the software requires furthermore that the final software behaves as expected when executed in the target system, and that it continues to perform at the same level throughout its life time. Important issues related to software approval and deployment include how to ensure an effective approval process and that the software preserves its safety integrity and dependability when it is deployed in the final environment of application and when making corrections, enhancements or adaptations to the software.

A research problem related to approval is what characterizes effective approval processes. Successful introduction of software important to safety requires effective processes providing the necessary documented evidence that the software is safe to put into use. This puts great demands on all life cycle phases, and needs to be reflected in the processes employed for the development and approval of the software.

The identification of the most important criteria behind successful safety approvals can be approached partly by surveying the approval processes in different countries and industrial sectors. Such a survey will probably show that these criteria affect both the authorities' approval activities and how the suppliers can support these activities.

A related research problem is the role of safety qualification tests as part of the approval and acceptance of a software based system. In many cases, the safety case prescribes a number of safety qualification tests to be carried out under operational conditions before the concerned system is given full responsibility for safety. These tests do not replace the safety argumentation in the safety case, but are designed to provide increased confidence in the system.

The problem on what is the proper use of safety qualification tests can partly be approached by surveying the role of such tests in concrete projects. Such a survey will provide an important basis for recommendations on how safety qualification tests can be designed and carried out to support the acceptance and deployment of software important to safety. This includes recommendations on how to provide the necessary documented demonstration of the sufficiency of the tests.

4. Safety demonstration

Common to the approval processes for software based systems important to safety is the need to demonstrate that the system is fit for its intended purpose and safe to put into use. A common approach in many countries and industries is the presentation of safety argumentation in terms of a documented safety case.

The essence of the safety case is the safety argumentation it gives. The quality of this argumentation is an important factor in achieving the necessary approval and acceptance to put the system into use. Compared to other types of documentation, there is one aspect of quality that stands out as particularly important, viz. the quality of the argumentation as a logical, valid, comprehensible argumentation. A good safety case needs not contain much documentation, use many words, represent "good literature" or use elegant language. The safety case needs however to present the safety argumentation in such a way that the independent assessor and the relevant regulatory authority are convinced that the system can be accepted as adequately safe for the intended application.

As explained in this paper, several important research problems within software systems dependability relates to the need to give such an argumentation. The need to provide a convincing argumentation gives direction to the different activities throughout the development of the software. This observation gives valuable insight into the intimate relationship between the life cycle activities on the one side, in particular those involved in the development and assurance of the software, and the provision of the satisfactory safety demonstration on the other side. In practice, such a demonstration requires that all the evidence needed to carry out the necessary argumentation is produced at the relevant steps in the development, *i.e.* as part of the different development and assurance activities.

The safety demonstration needs evidence both on the quality and safety management employed in the development of the system, and on the functional and technical safety of the system. While evidence of proper quality and safety management clearly contributes to the confidence one can have in a system, the actual demonstration that the system is adequately safe for its intended application is first of all supported by the evidence on the functional and technical safety of the system. This consists of the technical evidence for the safety of the design, comprising first of all the demonstration of the appropriateness of the technical principles adopted to assure the safety, and all supporting evidence. Examples of supporting evidence are the reported results from testing, verification and validation. The requirements to the planning, performance and reporting of the activities producing this evidence are usually given in the international or national standards adopted by the supplier or required by the relevant safety authority.

The most critical issue regarding the safety case is its ability to make a convincing argument that the risk involved with putting the system into use has been reduced to an acceptable level. This means that the argumentation needs to be supported by documented facts, first of all that all relevant safety requirements have been established and are fulfilled by the system. The supporting facts need to be documented in such a way that they are auditable to a third party. Of particular importance is the use of a configuration management system that ensures traceability and change control. This should be applied both to the different documents and units produced in the project, as well as to each single requirement to be implemented. Both are essential for ensuring control of development, review and change of the products delivered. The evidence of such a control, and the documentation supporting it, are essential to successful assessment, approval and deployment for all software important to safety.

5. Conclusions

Starting with a brief discussion on the concept of software safety integrity and its relationship to software systems dependability, this paper has given an introduction to research problems addressed by the OECD Halden Reactor Project within this area, followed by a discussion on the important role of safety demonstration in this context. The paper has shown how a proper understanding of safety integrity and safety demonstration influences on the selection of processes, methods, techniques and tools to be used

in the different life cycle phases of the software. This perspective on software systems dependability is also important for understanding the Halden Project's research programme within this area.

As has been emphasized in the paper, the need to provide a convincing safety argumentation gives direction to the different activities to be performed throughout the development of the software. This observation gives valuable insight into the intimate relationship between the life cycle activities on the one side, in particular those involved in the development and assurance of the software, and the provision of the satisfactory safety demonstration on the other side. Such a demonstration requires that all the evidence needed to carry out the necessary argumentation is produced at the relevant steps in the development, i.e. as part of the different development and assurance activities. In this sense, safety demonstration should be a matter of concern throughout the whole software life cycle, and not something to be postponed to a late stage where it might be too late or too difficult to provide the necessary argumentation and supporting evidence.

References

[1] International Electrotechnical Commission (IEC): IEC 61508 ed2.0, Functional safety of electrical/electronic/ programmable electronic safety-related systems, 2010.

[2] European Committee for Electrotechnical Standardization (CENELEC): EN 50126, Railway applications – The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS), 1999.

[3] CENELEC: EN 50128, Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems, 2001.

[4] CENELEC: EN 50129, Railway applications – Communications, signalling and processing systems – Safety related electronic systems for signalling, 2003.

[5] Radio Technical Committee on Aeronautics (RTCA): DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992 (errata 1999).

[6] IEC: IEC 60880 ed2.0, Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A Functions, 2006.

[7] IEC: IEC 61226 ed3.0, Nuclear power plants – Instrumentation and control important to safety – Classification of instrumentation and control functions, 2009.