Software Safety activities in Sweden

I. L. Bratteby-Ribbing, Defence Materiel Administration, Uppsala, Sweden

Keywords: real-time systems, reliability, requirements, safety culture, safety interface, software safety

Abstract

Software is increasingly being used within different application domains to implement safety-oriented and safety-critical functionality. Software system safety is therefore crucial in the effort to design safety properties into software parts in a way that promote safety on an overall system level. Important in this safety work is to consider the characteristics that distinguish an implementation in software from realisations in other techniques.
Sweden is a country with a strong safety climate. The technical level and industrial maturity is high. Close collaboration between different disciplines and domain experts has long tradition. The synergetic effect of these cross-fertilising circumstances has fostered an open, non-punitive safety culture. This paper gives an overview of the various software safety activities prevalent within the scientific, industrial and defence sectors, with examples from companies and organisations within these fields. In particular, the way the software safety community has adopted this safety culture, is described. Special emphasis is on the defence sector, where the author is active.

Introduction

Security vs. Safety: Early computer applications were primarily automated data-processing systems. For such information systems the main concern is security – immunity from external threats. This, and presumably also the fact that Swedish has a single word "*säkerhet*" covering both safety and security, have contributed to a tendency among Swedes to associate this term solely to the concept of security. With automatic control of systems operating in real world and real time, the system itself started to pose threat to people and environment. Consequently, safety became an urgent issue for such real-time systems. Still, the same safety procedures as those in use for mechanical systems were applied. Not until the end of the 20th century did software safety *per se* gain interest.

Software characteristics: Even if the overall principles for system safety apply to all types of system realisations, software has intrinsic properties, which may require a different approach in order to achieve a safe system. As apposed to a human-controlled hardware-based solution, a software implementation offers simpler and cheaper ways to design logic into a system. The potential to create a system responsive to changes in the operational environment is also enhanced. At the same time the higher degree of automation of tasks for monitoring, control and decision-making leads to a substantial growth of the complexity. Concurrent execution of an integrated system-of-systems, for instance, adds considerably to the number of interaction points between the system parts. The multitude of states, modes or configurations that included systems can enter, also tend to escalate the complexity. Despite the benefits of a flexible and adaptable system, these properties have to be balanced against other qualities necessary for a safe system. One of those is simplicity, devised to fulfil, e.g., determinism and testability. These are features that need to be explicitly addressed during design of the software system. A different aspect is that the enhanced software control over safety-critical functions, will cause more of the operator's responsibilities for safety to be transferred to the software parts. This will place more stringent safety requirements on the system architecture, the people involved in the software development and on the processes they use (cf. ref. 1). This responsibility shift – already experienced by the aerospace industry – is currently facing the automotive sector.

The role of software: Software in safety-critical applications is used to implement various safety-related features. Typical examples are functions for monitor, control and display of safety-critical equipment, activities and information. Another role is to protect against a detected safety threat. Software may also be the realisation technique selected to separate parts of different safety criticalities – a *criticality partitioning*, which not necessarily is equivalent to a separation with respect to requested response times.

Partitioning realisations: The main purpose of the criticality partitioning is to ensure – when competing for shared resources – that a system part of none or lower safety-criticality will not be able to interfere with a part of higher criticality. Logical-temporal separation (as apposed to physical) is one of the issues in Integrated Modular Avionics (refs. 2-3). Many software-based Real-Time Operating Systems (RTOS) have started to provide this support, some of them racing to develop separation kernels allowing each partition to run its own OS. A different approach was taken in the development of the *SafetyChip* framework, a multi-national research project providing hardware run-time kernels (RTK) both in single-processor and multiprocessor variants (ref. 4). Another hardware-based solution independent of the RTOS has been proposed within the Swedish avionics industry (ref. 5).

Requirements for safety: Different requirements need to be placed on the safety-related software in order to maintain safety at an overall system level. For functions/parts that control or in other ways have influence on safety-critical activities, requirements on *safety* have to be specified. Where instead the overall safety is dependent on satisfactory performance of the requested functionality within a specified time frame, requirements on *reliability* are crucial. This applies, for instance, to safety functions – functions responsible for monitoring or protection. These were also the first type of functions to be digitized by the process industry. Reliability has consequently been of prime concern for software in nuclear power stations (ref. 6). In contrast, the main requirement on a criticality partitioning realisation is *independence* between the partitions. Chapter *Industry* will discuss these issues further.

Safety requirements specification: The type of safety requirements provided by standards and guidelines are either *general* (i.e. application independent) or *domain-specific*. In order to capture the *system-specific* safety requirements, various safety analysis techniques will need to be applied to the system in its intended environment and usage. A first step in the analyses, however, is the identification of safety threats (e.g. hazards or other contributing factors) and of safety-critical parts. This is used to define safety requirements and design restrictions for the critical items. The goal is to arrive at a design where unacceptable safety threats are removed or at least reduced to a tolerable level, if possible. Threats still remaining after redesign are disabled or controlled by additional protection systems and safety functions. The final step in this restructuring process is the introduction of warnings and usage restrictions.
Even if most traditional safety analysis techniques also are applicable to software, not all may be efficient enough to compensate for the perceived effort. A limited study on this subject is reported in chapter *Defence*.

Software evaluation: In evaluating a specific software requirement or property, it is crucial to realize that this can not be fully assessed without regard to the software's intended context _ the system in which it is going to be integrated, the operational environment of the system and the way in which this is going to be used. Any change to these aspects _ even a seemingly insignificant difference _ may turn the execution of the software control flow into another path, resulting in a radically different behaviour. For a real-time system it may suffice with a minor shift in the timing conditions, e.g. a replaced processor with a slight drift in its internal clock, or a change in speed of the system under control or observation (refs. 7-8). This discontinuous character of software, where small deviations in the preconditions may result in large variations in the overall system functionality, makes it problematic to reuse software components, especially for safety-critical applications. These are the reasons, why, for instance, the property of software safety _ the software's contribution to the safety of the system _ can not be evaluated in isolation. Moreover, software regarded as "safe" in a specific system and usage, may not be so even in a similar, but slightly different context. Special care is therefore needed in the design of a software component in order to promote safe reuse. An interesting approach in this connection is the research on *Safety interfaces* (see chap. *Academia*).

Software system safety: From the above it is clear that the activities and procedures needed to incorporate, maintain and assess the required software safety properties, can be seen as a combination of Software engineering, Reliability techniques and System safety methodology. To promote software safety, progress in any of these areas is important. In the following, some advancements within the research community, industry and defence are described.

Academia

Background: In Sweden more than 60 universities, university colleges and government-independent institutes provide academic programmes for higher education and research. Many of these cover areas of importance to software safety, e.g. different aspects on safety and risk management, computer science, systems and software engineering, real-time systems, software quality, formal methods. Few of these, however, do explicitly address software safety as such. Several programmes focus safety-related issues such as reliability, fault tolerance, failure awareness. Others examine more basic aspects, for instance, predictability and complexity reduction, both vital to pursue any safety analysis at all. This chapter presents a small selection of research projects combining software engineering with reliability and safety issues. These programmes are typically carried out in collaboration with different research institutes and industrial partners.

Chalmers University of Technology (CTH): Chalmers and Göteborg universities have established the joint department Computer Science and Engineering (CSE). The research within CSE's Division of Computer Engineering includes, but is not limited to, dependable computer engineering, real-time systems, software engineering and computer communications. The studies within the Software Methodologies group cover, for instance, different aspects on development, maintenance and safe usage of large systems. The Applied Software Engineering group explores new processes and technologies for software-based systems, among those quality criteria and methods for identification and reduction of common software deficiencies. Special attention areas are automotive applications and system safety. The work is often performed as member in other research consortia in collaboration with various specialists. One such research constellation is the Software Engineering Laboratory (SEL), which examines novel methods for hazard identification in the early development stages and designs new methods for process improvement. SEL is involved in the research project *Cost Efficient Dependable Electronic Systems* (CEDES), which is part of the national research programme *Intelligent Vehicle Safety Systems* (IVSS) set up by the Program Board for Automotive Research (PFF) in collaboration with Industry and Government. The aim is to stimulate R&D of road safety. CEDES is thus associated with, but independent of CTH, a way to promote cross-disciplinary fertilization between engineers, researchers and domain experts. The project focuses on software technology, especially within fault-handling, formal methods and distributed systems. Three fault-handling mechanisms are studied with the intent to simplify the design of fault-tolerant systems. One approach is based on aspect-oriented programming (AOP) to allow basic functionality scattered throughout the different modules of the system to be captured, encapsulated and reused. These cross-cutting aspects offer another type of abstractions than those enabled by procedural or object-oriented programming. Another approach is to build up a source code library of fault-handling routines. In the third approach the application's inherent properties are used. The project applies formal methods for symbolic execution of source code programs. One aim is to carry out symbolic fault injection during the execution in order to verify dependability in early development stages and reduce extensive testing. Within the area of distributed systems, membership handling on the application level is studied. This is of interest to future integrated systems, possibly with several applications running in each node. Membership handling on node level only, will in such cases not be sufficient. Currently eight PhD students work in the project.

Linköping University (LiU): At Linköping University research in the areas of dependability and resource allocation in distributed systems is conducted at the Real-Time Systems Laboratory (RTSLAB) of the Software and Systems Division within the Department of Computer and Information Science (IDA). Among the topics addressed within the dependability domain are *Formal analysis of safety and fault tolerance* and *Availability and survivability in critical infrastructures*. A number of projects are conducted in collaboration with other universities and industrial partners as parts of different national and European research programs, e.g.

– *System Safety* (NFFP-428), a now concluded study on languages with formal semantics for specification of system components (e.g. reconfigurable components, FPGAs) and techniques for formal verification and automatic code generation investigating how these can be used and combined with existing methods for safety analysis (FTA, FMEA).
– *Incremental Verification of Modular Avionic Architectures* (NFFP4), a continuation of the previous project. In NFFP4 techniques for compositional formal verification of component-based systems are adapted and applied to distributed modular avionics architectures.
– *Adhoc Networks of Unmanned Aerial Vehicles* (UAV-MANET), which investigates suitable mobility models for UAV networks and model services delivered to mobile ground stations in presence of safety-related, security-induced and resource-oriented constraints.

An interesting concept in this research _ and a complement to the functional specification of a component _ is the notion of a *safety interface* (ref. 9-10), which captures the component's behaviour with respect to a given system-level safety property regardless of undesired faults in the intended environment. By generating the safety interface of a component from a specific system-level safety property and a set of pre-defined fault modes, the effects and assumptions needed for the component to be resilient to these faults can be specified. The automatic derivation of these interfaces is supported by formal verification tools. The safety interface approach is useful, for example, in evaluating safety violations of the resulting system at component upgrade or replacement. A safer reuse of components is achieved at the same time as it allows safety analysis results from the component level to be reused for a complete safety assessment at the system level.

Mälardalen University (MdH): The research within the Department of Computer Science and Electronics is oriented around embedded high-integrity real-time systems. Of special interest for software safety has been the *SafetyChip* project, a work started at Uppsala University and continued at MdH, where, in co-operation with MIT, a number of PhD projects have been completed (ref. 4, 11). In this project a formal (and readable) model of the system is generated from source code (e.g Ada, VHDL) and used to formally verify the requirements specified for the system. From the model a so-called *SafetyChip* (supporting the Ada Ravenscar tasking model) can be generated for runtime monitoring of the system. Several variants of these hardware-implemented RTKs have been produced.
Another MdH-project, *Drive/NTCM*, is studying the software safety standards IEC 61508, ISO/DIS 15998 in relation to the computer architectures used in earth-moving machinery. In this collaboration with Volvo CE the

intention is to investigate, how the requirements of these standards are to be interpreted, in order for the technical solutions to comply with requirements for each safety criticality level. Technical solutions on how to mix differently classified functions and how to integrate supplier components are also studied. Another issue is to identify measures to take in order to introduce safety-classified functions on the current architecture.
The newly-founded PROGRESS center at Mdh focuses on predictable development of embedded software systems using a component-based approach. Within PROGRESS, a dependability cluster has been formed to conduct research and provide analysis techniques considering the crosscutting nature of the various dependability attributes such as reliability, availability, safety as well as their trade-offs. The composite attribute, robustness, is also being addressed in order to highlight conflicting requirements on dependability, to allow studies on system overload and to be able to perform sensitivity analysis of systems for predictability guarantees. Among the issues to investigate are:

– Parameterized reliability models of systems built on top of components, taking into consideration their usage profiles and interactions
– Fault tolerant techniques and schedulability analysis of systems under various fault models especially taking safety aspects in account
– Formal verification of liveness and safety properties using model checking
  Liveness is an indispensable property for timely execution of a concurrent application with several processes simultaneously competing for the same shared resource. Typical examples of liveness problems are deadlock, livelock and starvation, situations where, in different ways, access to such a resource is prevented.
– Wrapping to prevent unsafe behaviour of legacy systems parts
– Instrumentation and probing of systems to develop failure-aware systems.
  Failure-awareness, as opposed to fault-tolerance, focuses on impending failures (rather than potential faults) together with suitable counter-measures (e.g. flushing caches/stacks or resetting a node in a distributed system).

Royal Institute of Technology (KTH): The Embedded Control Systems group within the Division of Mechatronics in the Department of Machine Design is conducting research on development methods for safety-critical embedded systems, in particular approaches employing model-based development techniques that facilitate

– systematic and cost-efficient analysis, including formal techniques as well as testing and reviews
– configuration management and traceability, targeting the product life-cycle

The application area in focus is automotive embedded control systems, mainly through European and national projects such as

– ATESST, *Advancing Traffic Efficiency and Safety through Software Technology*. The main contribution from KTH will be requirements on, and assessment of, the new Architecture Description Language, EAST_ADL2.0, which provides means to improve safety, reliability, cost and development efficiency.
– SAVE, *Safety critical vehicular systems* (ref. 12).

Exchange of experiences with other application domains including aerospace, medical equipment and telecommunication will be set up. For publications and current activities, see reference 13.

SAVE is a national research project hosted by MdH with partners from KTH, Uppsala University, and Linköping University. The goal is to define an engineering discipline for systematic development of component-based software in safety-critical embedded systems. One of the challenges in the development of such systems is to combine a bottom-up component approach with the top-down view that system safety requires. At the same time other conflicting requirements must be handled, e.g., safety vs. cost and time-to-market. Important factors in succeeding to achieve safety while time and cost are reduced, is a well-defined, safety-oriented architecture, which allows reusable software assets to be inserted and replaced. A structured use of appropriate design methods, composition/integration support and verification techniques are essential to accomplish this. The general framework, which initially will focus on safety-critical vehicular systems, will include

– Methodology and process for development of systems with components
– Component specification and composition, providing a component model which includes the basic characteristics of safety-critical components and infrastructure supporting component collaboration.
– Techniques for analysis and verification of functional correctness, real-time behaviour, safety, and reliability.
– Run-time and configuration support, including support for assembling components into systems, run-time monitoring, and evaluation of alternative configurations.

SP Technical Research Institute of Sweden: The Software section within the Electronics department of SP is active in dependable systems and functional safety of programmable electronic systems. The research and the commissions for industry are mainly within the fields of vehicle electronics, safety of machinery and safety in process control. Focus is on risks emanating from faults in control systems resulting in personal injury, damage to property or to the environment. The safety efforts are concentrated on proactive risk minimisation (e.g. correct design) and safety evaluations, where aspects on software safety are important ingredients. The research activities cover software-based fault injection, validation of safety-critical software and model-based development of safety-critical software. Among the research projects are

– DECOS (*Dependable Embedded Components and Systems*), a project within the European ICT research programme. The objective of DECOS is to develop a basic enabling technology to move from a federated distributed architecture to an integrated distributed architecture.
– SafeProd (*Functional Safety in Complex Products*), which is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA). The main focus of SafeProd is to explore ways to develop complex products based on hardware, software and mechanics with a high level of safety still being maintained.
– CEDES and AutoVal, both projects within the Swedish national research program IVSS (*Intelligent Vehicle Safety Systems*). CEDES addresses development of cost efficient dependable electronic automotive systems. AutoVal is focusing on safety validation of automotive electronics and software.

Industry

Safety culture: Long before the industrial era and mechanization of the Industry, safety has been of prime concern to the Swedish society. This has fostered a safety culture emphasizing pro-active safety measures. A posteriori investigations based on after-the-fact knowledge have been supported by open incident reporting rather than blame and claim on individuals. Independence between developer, safety assessor/certifier and accident investigator has been a hallmark.

Software safety: System safety activities were introduced early in the Industry, especially within the transportation sector, the energy domain and the medical field. In contrast to this, Software System Safety is a fairly young field, both as an academic and an engineering discipline. Nevertheless, principles for building safety into software products have been applied within the Industry since decades, even if these initially were based on empirical evidence rather than science. Several industries have played a prominent role in the early software safety initiatives.

The aerospace sector: A leading company within the aerospace industry is Saab Aerosystems. Among its more advanced efforts is the design and manufacturing of the Gripen multirole aircraft, the first operational fourth generation fighter in the world. The aircraft is totally reliant on software in order to sustain safe flight. Also as a fourth generation fighter the systems are highly integrated. The safety process used to ensure safety of the aircraft has to be adopted in order to account for a high degree of integration in the aircraft. From a safety perspective, integration poses many challenges since it is more difficult to prove fault containment boundaries in an integrated system. Saab was well aware of this at the start of the Gripen project. Together with the Swedish Defence Materiel Administration (FMV), Saab established a detailed safety process to be used for the Gripen aircraft. This process also provides guidance on how to determine the criticality of different software components and what measures to take in order to manage the risk of software deficiencies in those components. In the continuing work to improve safety and reliability of the Gripen aircraft, Saab has during recent years adopted tailored versions of the standards used by the civil aviation community, such as RTCA/DO-178B and ARP 4754 (refs. 14-15). These guidelines address issues such as assessing risk to third parties (not covered by civilian standards), architectural design patterns for reduced impact of software errors and the use of external mitigation in the software safety process.
Currently Saab is engaged in the development of UAVs (*Unmanned Aerial Vehicles*). A number of patents have been acquired allowing novel solutions to some of the software-related safety problems that face manufacturers of UAVs. Among these is the issue of criticality separation (ref. 5).
Saab is also active within SC-205/WG-71 committee in its revision of RTCA/DO-178B. The new revision will most likely set the stage for the next 15-20 years within the area of software safety in the aerospace domain.

The automotive sector: With the introduction of new technology in vehicles, the automotive industry in general has experienced an intense evolution. The volume of software in cars, for instance, has since the 70ies undergone a rapid growth – from practically nothing to 10^7 measured in lines of source code. In addition, common networked infrastructures supporting dependable data communication have been developed. These advances together with other technical innovations have initiated a change of the development process. A transformation is thus in progress, from a bottom-up design of reliable, loosely coupled electro-mechanical control units towards a top-down, software-controlled approach based on a safety-oriented architecture on the topmost system level promoting safe real-time interaction between distributed subsystems and components. This system-wide scope will demand a safety viewpoint of the architecture (cf. ref. 16) focusing system as well as software safety, not only with respect to vehicle functionality, but also driver behaviour and traffic environment. The architecture will no longer be limited to a design expanded with systems for warning or active protection, such as collision avoidance and air bags. Safety features of the command and control functions will also be addressed. More of the driver's or owner's responsibilities will in that way be transferred to the producer and system designer. Monitoring of the vehicle surroundings, cooperative communications (e.g. for synchronised operations of multiple autonomous vehicles) are among future features. An adoption of the classical design principles for safety can already be seen. Typical examples are separation of parts of different criticalities, prevention from unintentional execution of deactivated code (designated for a specific system configuration or mode etc). Within the car industry these have given rise to a number of design patterns, for instance, exclusive association of a certain key position to a specific operational mode or function – a pattern reused from the aerospace domain.

Sweden is recognized as a world leader in automotive safety with a strong position in telematics and wireless communication. The innovative climate and the collaborative spirit in these areas have fuelled the vehicle industry with Saab, Scania, Volvo Cars and Volvo group as the main manufacturers. The technology shift with several partners responsible for the different subsystems will demand a systematic and uniform development strategy with technical solutions founded on international standards regarding, for instance, communication and control. Swedish vehicle industry participates, for instance, in the international AUTOSAR (*AUTomotive Open System ARchitecure*) partnership, developing and establishing a de-facto open industry standard for automotive electronics architecture in general and software architecture in particular. To ensure that safety aspects are appropriately addressed in AUTOSAR, a dedicated safety team has been established.
The Swedish automotive industry contributes to the development of a new international standard for functional safety of road vehicles through the working group ISO TC22/SC3/WG16 (ref. 17). The current draft includes the part *Product development – Software level*, an adaptation of IEC 61508's software requirements to the automotive domain (ref. 18). Additionally, the Swedish automotive industry is active in many European Union research projects. Several of these are concerned with software safety issues (e.g. EASIS, ATESST, cf. chap. *Academia*).

<u>The naval sector</u>: The Swedish naval industry has long record of shipbuilding. Kockums, who entered naval business in 1870, is still one of the leading shipyards. Today, the company designs, builds and maintains multi-mission surface vessels and submarines. Frontline technology such as composite materiel, stealth features and the Stirling air-independent propulsion technique is used to reduce mass and signature at the same time as mission flexibility, shock resistance and endurance are increased.
The responsibility of the overall ship and the combat management systems covers system design, integration, installation and V&V of subsystems developed in-house and by subcontractors. This means that the main system safety activities involve the early phase of safety requirement specification and the various safety verifications performed on the system in its successive versions after integration and installation. For safety-critical software parts the software safety handbook, reference 1, is used and relevant requirements forwarded to the subcontractors. Even with all subcontracted components and subsystems developed according to prescribed safety standards and procedures, unintentional combination effects may appear after integration. During safety verification of the software, special attention is therefore paid to interacting parts and their exchange of messages and signals over different communication channels.

One of the main providers for the included computer-based systems is Saab Systems. The Naval Division develops advanced communication, command and control systems integrating multi-sensor directors and weapons to support network-based operations. A uniform system architecture is defined for a family of command and control systems adaptable to different customer needs, application domains and operational profiles. The open architecture is founded on COTS and standard interfaces to allow easy update of software and hardware equipment. This is further supported by selecting Ada as the main implementation language, Ethernet for the local area network and Windows XP embedded for the operating system. A consequence of the latter is that additional safety constraints have been imposed on the design. One such restriction placed on software is, for instance, to deny autonomous control over potentially hazardous functions or commands (cf. ref. 19). This means, e.g., that both software control and hardware key activation are required for the final firing engagement of a weapon.
The practiced software safety activities are part of an overall system safety program. An integration of the system safety methodology into the company's software engineering environment has started. As an example, the Division Air & Land has supplemented their software development process – an adaptation of the framework defined by the IBM Rational Unified Process (RUP) – with principles and requirements on software system safety according to *H ProgSäk* (ref. 1). The requirement database has also been extended with safety attributes allowing tools to trace between safety requirements, implementation parts and verification/validation evidences.

The nuclear power sector: The industry within the nuclear power sector is subject to high safety regulations. Research and development within safety and the introduction of software-based safety systems are performed mostly in international cooperation. Swedish experts participate in the work performed by IAEA, EC and OECD Nuclear Energy Agency along with the standard development of IEC. As an example, a task force on safety-critical software with experts from the nuclear regulators in the EC countries was set up in 1994. The latest report (ref. 20) gives common positions and recommended practices on several important safety issues, both those specifically addressing life-cycle phase licensing and those of generic type.
Current issues of concern for the digitized systems are consensus on strategies for verification and validation, and how the evidences for safety shall be structured and performed. Examples of important software safety properties are low complexity, functional diversity and deterministic behaviour. The systems are mostly built on commercial platforms. The provided platform functionality is tested and validated before a subset is selected for implementation of the safety functions. So far, bus technology is only used for information exchange within each of the four redundant subsystems (so called *trains*), but not between the different system parts.
The instrumentation and control system in nuclear power plants has a *defence-in-depth* approach. This means that safety is built up in several layers. The first level consists of the normal control system. This is responsible for the normal operation, for instance, the automatic feed-water control or the control system for the steam to the turbine. If there is a failure in the steam control system, the second safety level is entered. An alarm will then be presented to the operators in the main control room to allow for manual correction. If the operators fail or don't have enough time for action, the automatic reactor protection system belonging to the third safety level will take over and shut down the reactor. Between these different control systems there is a clear hierarchy; the operators can override the normal control system and the reactor protection system can override the other systems. The design strategy is also that failures in a system of low safety category shall not affect systems of a higher safety category.
The instrumentation and control systems of the Swedish nuclear power plants were initially built with conventional analogue technique. A modernisation has now started. The oldest plant in Sweden, Oskarshamn 1, was upgraded in 2002. Today digital technique is used for its instrumentation and control. In order to achieve high reliability of the reactor protection functions, a diverse back-up based on conventional technique is added to the primary, software-based protection system.

Defence

History: The loss of the royal ship Wasa in 1628 made the king Gustav II Adolf realize the value of a safe and stable architecture built on quality materiel. After this disaster, system safety became a recognised discipline within the Swedish Defence. An immediate consequence was the establishment of the Royal War College. Two centuries later, this was reinforced with the Royal Army Materiel Department. After another century the Royal Air Force Materiel Department was set up. These agencies were in the 1960-ies joined into the Defence Supply Administration, a predecessor of FMV, the Swedish Defence Materiel Administration.

System safety: Even though system safety has long history within the Swedish Defence, it was not until the end of the 1990-ies that a more systematic approach gained ground. A proactive system safety programme was defined in a first issue of a system safety manual (ref. 21), an upgrade of the accident-preventing safety procedures in use.

Software System Safety: A similar move was taken by FMV a couple of years later for Software System Safety. The first step was to start a software safety project. The purpose was to implement a software safety handbook, *H ProgSäkE*, (ref. 1), where the overall system safety requirements of the System Safety Manual is detailed for software. Another early initiative was to introduce software safety into FMV´s strategic competences and to create a specialist position within the discipline. With these efforts as a basis a software safety programme was established. Introductory material and support tools were prepared and made available on the web, (refs. 22-23). As an example, cross-reference tables for comparison between *H ProgSäkE* and some other software safety manual are provided to facilitate evaluation of software developed according to this other guideline (ref. 24). A result of such a comparison is a mapping between *H ProgSäkE* and the aviation standard RTCA/DO-178B (refs. 14, 25).
An adaptable template is also available assisting an organization to assess its software safety maturity and to exhibit the safety status of the software project under development. This is achieved through evaluation of the company's compliance to the generic *H ProgSäkE* requirements as regards its competence profile, the processes used to produce safety-critical software and the necessary properties of such software products (ref. 26). A finding from companies that have performed this task is the need of close cooperation between professions of different engineering disciplines. Another effect is an increased appreciation and understanding among traditional system safety engineers, system designers and software developers.

Knowledge transfer: In addition to the above activities, FMV arranges courses and seminars addressing different software safety themes. FMV has also organized, and is heavily involved in, a study group for software safety within SESAM, a non-profit association within the defence sector for software engineering. The working model for this group was taken from a Research & Technology project, FoTA, successful in its knowledge exchange between participants from Defence, Industry and University.
One objective, stated at the inception of the SESAM group, was that software safety within five years should be regarded as an indispensable qualification for every software engineer building safety-critical systems. Now, four years later, the majority of the group members are themselves their companies' experts in this area.

The study group: The SESAM group studies topics relevant to safety-critical computer systems. The work is organized in *themes*, which are subdivided into one or more *micro projects*. Less demanding tasks are categorized as *discussion topics*; each with its own coach introducing an urgent issue and stimulating further discussions. The idea is to get a wider view, especially in cases, where no solution seems possible or, where several or conflicting approaches exist. Among issues discussed are "*How to organize and integrate system and software safety with software development*", "*Different ways to map the software criticality on the system's risk matrix*", "*The effect on risk reduction of a mishap classification based on* the worst possible *versus* the worst reasonable *consequence*".

Study themes: The theme, *Autonomous Vehicle Systems*, was selected for the first study. Public seminars by field experts from government organizations, Industry and research institutes were arranged with the scope limited to safety-critical software in unmanned underwater, ground and air vehicles.
A similar approach was used for the second theme, *Human-Machine Interaction in Safety-Critical Systems*. A series of lectures covering various application areas – from control rooms and operations centers to cockpits and ships bridges – were given by specialists from related branches such as cognitive science, system and software engineering. Among the issues of interest to the group was to which extent there exist techniques to model a multimodal man-machine interface (MMI) – a dynamic flow of visual, auditory and tactile sensory information presented to an operator in critical situations. An associated question was how to evaluate this MMI-design proactively with regard to safety – something hard to achieve with the traditional, static-oriented safety-analysis methods currently at hand.

Micro projects: In parallel with the HMI-seminars, a micro project, *MMI safety*, was conducted. The task appointed to the members was to investigate their in-house collection of design criteria, patterns, development principles, methods, tools etc. in use for safety-critical MMI-scenarios. The result, which could be documented both in internal and open versions, turned out to be of interest not only to the group members, but also to other sections within the own company.

A micro project currently underway is *Safety Analysis Methods for Software*. The purpose of the study is manifold.
One objective is to find the analysis techniques most effective in early identification of the various types of hazards, where software is a contributing factor. Another is to learn during which stages in the system development these methods are applicable. The study's input material and result should also be compiled into self-instruction materials for publication on the SESAM home page. This includes compiled fact sheets summarizing related safety analysis phases (e.g. PHL, PHA, SHA, SSHA), the methods examined (HAZOP, FMECA, FTA) and a number of checklists supporting these methods. A unique example of the latter is a general checklist of software-related hazards.
The software system to which the methods were to be applied was an ejection system for safe escape from an aircraft in emergency situations. This fictitious subsystem was modelled with a high degree of automation in order to make it more challenging from a criticality viewpoint. As for the rest of the system, the design was described on a high level – albeit not without deficiencies. Early software artefacts were provided, sufficiently detailed to allow a worthwhile evaluation without access to source code.
The limited time and effort that could be spent on the task meant that the analyses could not be brought to final completion. Despite these shortcomings, the exercises could be carried out to such a depth that a good understanding of the usability of the tested techniques on software could be gained and some conclusions be drawn. As a consequence of the study, methods previously untried on software were introduced by the team members in their companies. Among the findings of special value to the group was insight about how to enhance the effectiveness of the tested methods. The significance of adding other types of input to those traditionally recommended became apparent during the early brainstorming activities to obtain a Preliminary Hazard List (PHL). In this case the identification of potential hazards was speeded up by publicly available video-recordings of accidents and incidents involving ejection seats.

Another positive experience was the strength of HAZOP to support a systematic search in software for deviations from the intended design and operation, especially in communication parts. Special tables, which probably could be reused, were also developed interpreting HAZOP's general guide words for each parameter type of the analysed interaction (messages, signals, data flow elements etc).
The result from testing FME(C)A, Failure Mode and Effects (and Criticality) Analysis, on the ejection software caused some difficulties. Either the failure modes found were of generic type (deadlock, memory shortage, missing exception handler for triggered exception etc.) or the same failures as those already identified through HAZOP turned up (commission, omission, timing and value failure of requested function/command/message). Moreover, it was felt that the former type of software faults more effectively could be singled out by other, more dedicated analysis techniques available in the traditional software toolbox. Therefore, FMECA's efficiency in finding failure modes for software did not convince, but the technique was believed to be more useful on a higher level.
In applying Fault Tree Analysis (FTA) to software one aim was to avoid an analysis terminating in basic events of type "*Erroneous software*". For more useful results the system-specific software defects had to be found. The analysis was therefore continued into component design, yet without penetrating the code structure. Examples of typical basic events arrived at were "*Navigation data fault*", "*Erroneous navigation algorithm*". Another study reported to the group was an analysis of the node layout in a large, operational system. In this case the potential loss of an entire software functionality (omission failure) was evaluated and used to improve system availability.

Among other experiences from this study group is that technical aspects on software and/or safety issues – for very good reasons – can serve as an excellent vehicle for promoting an open atmosphere of knowledge sharing, underpinned by informal communication and trust: No stakeholder wants a system to cause accidents. Furthermore, by selecting general software safety issues of enough importance and, in addition, by limiting the effort required, it is also possible to engage professional engineers in this type of external studies, despite heavy involvement in the company's own projects.

Summary

This paper has presented some different software safety efforts within the Swedish Industry, Academia and Defence. These sectors have distinct preconditions, driving forces and goals. Yet, in a small country it is imperative that these efforts merge into a successful realization both on the national and the international arena.

References

1. H ProgSäkE, Handbook for Software in Safety-Critical Applications, M7762-000621, Swedish edition 2001, English edition, 2005.
2. ARINC 651. Guidance for Integrated Modular Avionics, Aeronauticcal Radio Inc, Nov 1991.
3. ARINC 653. Avionics Application Software Standard Interface, Aeronauticcal Radio Inc, Jan1997.
4. G. Naeser, A Formal Approach to Embedded High-Integrity Real-Time Systems, Mälardalen University, 2005.
5. R. Johansson, Computer facility and procedure for execution of software of different criticalities (a translation of "*Datoranordning och förfarande för exekvering av programvaror av olika kritikalitet*"), Patent 0004421-4, 2003.
6. IEC 60880. Software for computers in the safety systems of nuclear power stations, Publ. 880, Sep 1986.
7. J.L. Lions, Ariane 5, Flight 501 Failure, Report by the Inquiry Board, 1996.
8. GAO/IMTEC-92-26, Patriot Missile Software Problem, Feb 1992.
9. J. Elmqvist, S. Nadjm-Tehrani, M. Minea, Safety interfaces for Component-based systems, SAFECOMP 2005, Springer-Verlag.
10. J. Elmqvist, S. Nadjm-Tehrani, Safety-Oriented Design of Component Assemblies using Safety Interfaces, Proceedings of the International Workshop on Formal Aspects of Component Software (FAC'06), Elsevier, Sept. 2006.
11. K. Lundqvist, Distributed Computing and Safety Critical Systems in Ada, Uppsala university, 2000.
12. SAVE web site, http://www.mrtc.mdh.se/save.
13. KTH. The Embedded Control Systems research group, http://www.md.kth.se/RTC.
14. RTCA/DO-178B, Software Considerations In Airborne Systems And Equipment Certification, 1992.
15. ARP4754, Certification Considerations for Highly-integrated or Complex Aircraft Systems, Nov 1996.
16. IEEE 1471-2000. Recommended Practice for Architectural Description of Software-Intensive Systems.
17. ISO/WD-26262
18. IEC 61508, Functional safety of E/E/PE Safety-related systems, part 1-7, Dec 1998-May 2000.

19. 882E, Standard Practice for System Safety, MIL_STD-882E, Draft 30 Dec 2006.
20. Common position of European nuclear regulators for the licensing of safety critical software for nuclear reactors, EUR 19265, http://www.avn.be/uk/7_publications/9_2_articles_etudes.asp, May 2000.
21. H SystSäkE, System Safety Manual, M7740-784861, Swedish edition 1996, English edition 1998.
22. FMV web site, http://www.fmv.se: Publikationer: Handböcker: H ProgSäk 2001.
23. SESAM, Software Engineering for the Swedish Defence, http://sesam.smart-lab.se.
24. FMV. Cross reference tables for H ProgSäk E and XXX_STD, KC Ledsyst 14910:41378/04.
25. FMV. Cross reference tables for H ProgSäk E and DO-178B, KC Ledsyst 14910:41371/04.
26. FMV. Organisation/Project compliance with H ProgSäk E, KC Ledsyst 14910:41378/04.

Biography

I. L. Bratteby-Ribbing, Defence Materiel Administration, P.O. Box 228, SE-75104 Uppsala, Sweden, telephone – (+46) 18 120263, (+46) 70 3770263, facsimile – (+46) 18 120272, e-mail – inga-lill.bratteby-ribbing@fmv.se.

Inga-Lill Bratteby-Ribbing is Strategic Specialist in Software Safety at the Swedish Defence Materiel Administration (FMV). She has 40 years' experience in software development and techniques for safety-critical systems from the Swedish Atomic Energy Company (AB Atomenergi), the National Defence Research Agency (FOI), the defence industry (e.g. Saab Naval Systems) and since 1995 from FMV. She has been active on the boards of Ada-Europe and Ada in Sweden, in study and working groups within IEEE, ISO/IEC JTC1/Sc22 and SESAM (Software Engineering for the Swedish Defence Sector). She has presented articles and introductions in software safety within several organisations, such as Encress, ISSC and SESAM. At FMV she has worked as project leader and technical expert in software safety within different projects. One of these has produced a handbook for software in safety-critical applications (*H ProgSäkE*).