# *Extending System Safety to Software – Some Software Safety Considerations and Challenges*

I.L. Bratteby-Ribbing
Defence Materiel Administration
inga-lill.bratteby-ribbing@fmv.se

Keywords: System safety, Software safety, safety-critical systems

## Abstract

Software is an excellent instrument for realization of flexible systems adaptable to different situations, environments and missions. For safety-critical systems this versatility may conflict with the requirement to fulfil safety on an overall system level. A central issue of this contribution is to analyse the means and techniques needed to balance such contradictory albeit requested features. This paper will highlight – with examples from the defence sector – some desirable properties of software as a product, the processes used in software production-operation-maintenance and the commitment needed from the people involved in these processes. It will also discuss in which way the described features might compromise or support safety. Finally, an attempt is made, from a software engineering perspective, to present some of the challenges to be resolved for the future, in order to meet the contradicting requirements.

## Introduction

The discipline of software safety, *software system safety*, has become of increasing importance to a society that depends more and more on software and computing systems. This tendency was accelerated by the transition from a manufacturing society to a service-oriented, information society. The significance of software safety, however, is of special concern to those systems operating under real conditions in a real time context which could adversely impact people, property, or the environment. Today such safety-critical real-time systems are ubiquitous in almost every field of the industrial society. Computer-controlled equipment, x-by-wire features, autonomous vehicles, dynamic sensor-actuator systems, and support systems for information exchange and decision-making all enlarge the scope for a variety of safety-critical situations, where severe losses cannot be tolerated. This is equally true for defence systems, in particular when used in humanitarian and peacekeeping operations, where the tolerance towards losses is lower than in a pure military mission.

As a *discipline*, software safety can be regarded as an integration of the system safety and reliability methodologies into software engineering. In fact, reliability has been an integral part of software engineering since its inception. This is illustrated by the focusing on fault, error and failures in the search for software defects and in the choice of analysis and verification techniques. Besides, the influence of the reliability approach can be found among the design strategies, e.g., in the incorporation of fault tolerant mechanisms etc. However, with the introduction of system safety to computer systems in the late 20th century (ref. 1), the necessity to interpret the principles of system safety for the software parts also arose (ref. 2). As a result, focus shifted from defects in general, to those representing unsafe constructs, i.e. those with a potential of being exposed to hazards, unsafe states and accidents. Also recognized was that *software safety*, as a *property*, has to be designed into the individual component from the outset and, even more importantly, into the system architecture. Accordingly, in order to promote the overall safety of the resulting system, the safety activities needed to address the processes used, as well as the people involved in building, operating and maintaining the safety-critical software system. Thus, the main effort in integrating the three disciplines remains in the incorporation of safety considerations into the established software engineering framework, i.e. the processes, the design principles, the development environment and the tools included. In this work, a seamless integration is important; not only of the safety procedures into the traditional software processes, but also of the safety analysis methods into the design artefacts. This allows the various safety analysis techniques to be applied directly to the different design models and views defined for the system. In wait for tools to scale up to industrial strength, some software companies have started this integration into their company-specific software engineering support (ref. 3).

In order to "design-in" *software safety properties* to a software product, several attributes, on which safety depends, must be fulfilled. Examples on such derived properties are determinism and simplicity. Both are essential for concurrent real-time systems, especially for the safety-critical parts. *Determinism* allows the behaviour of a system to be decided from its prerequisites. This is necessary in order to be able to perform safety analyses prior to deployment of a system. *Simplicity* is indispensable to make it practically achievable to perform a correct and complete safety analysis on more than the smallest system. These properties, fundamental to safety, may be opposed to other requested system features. In the following sections the defence sector will be used to illustrate conflicting requirements and ways to address these without compromising safety. The conclusions, however, are equally applicable to other application areas.

## Future defence systems

The systems under development for command and control of task forces can be characterized as network-based information and real-time systems for Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance ($C^4$ISR). A decentralized decision model is depicted, where information is made available over the Net for the authorised user (the specialist as well as the generalist). The information is collected from a variety of sources and evaluated with respect to its trustworthiness, in order to be fused and structured with the different user categories in mind. The service and information provided allow for central and local distribution through different types of networks according to the various real-time needs. The architectural model is founded upon *open*, well-defined interfaces to enable component- and COTS-based development and also to facilitate replacement of system parts over time. The investment in an open structuring of the system architecture is attractive to the client, as it will reduce the implementation and upgrading efforts for developers and maintainers, thereby lowering the system's life-cycle costs. Other client-requested system features include *adaptability* to different situations and *flexibility* to allow immediate build of new configurations upon demand. While adaptability meets the end-user's need of getting a system sensible to changing requests during operation, flexibility gives the system operator the opportunity of fast system reconfigurations. These three features are covered in more detail in the last section, 'Challenges in the near future'.

Pro-active systems: The need to process information from thousands of computers will require a different type of man-machine interface, where much of the human interaction with the system has been automated. A consequence is that the operator will take a more supervising role, resulting in a shift from an interactive to a proactive system. Such a system would need to be super-fast to allow simulation, analysis and timely evaluation of the effect that a change to the system, its environment or user profile (e.g. input data) might involve. This super-fast system would also enable a dynamically-driven architecture, rather than a statically designed model. The main effect would be a transfer from a deterministic to a probabilistic model with implications on the safety strategies and the fault tolerance techniques as applied today. The prospects for such a transition and the underlying technology needed are discussed in the next section, 'Future computer technologies'.

Open architectures: The tasks that future defence systems have to solve impose a number of requirements on the system architecture and its realization. An open architecture based on well-defined interfaces and communication standards is a prerequisite for succeeding in integrating components, subsystems and systems from different vendors into a comprehensive whole. This is also central for achieving interoperability between the software products and to facilitate later upgrades and updates. For this type of super-systems, a hierarchy of architectures will have to be defined. These will describe the different system levels. On the top level, for instance, the overall structure and the system-wide functionality, together with the overall principles that included parts have to fulfil to arrive at a consistent, logically sound behaviour at this level. Furthermore, each architecture determines the underlying structure, e.g. included systems and subsystems, the relations and interactions between these, their communication with the environment etc. In addition to these traditional architectural descriptions and orthogonal to these, more comprehensive views, are necessary. These need to address security and safety, in order to assure that these properties can be satisfied for the system as a whole. Additional aspects on openness in the light of safety are provided in the last section.

Dependability: The computer systems developed for the defence will handle sensitive information over the network. Accordingly, much of the functionality provided and the information transferred are vulnerable targets that need to be classified. Besides, these may also be used more or less directly in safety-critical operations. A vital system property is therefore *dependability*, an expression for the trustworthiness "that reliance can justifiably be placed on the service it delivers". Essential characteristics are, in other words, the system's readiness for usage, its continuity of service, its ability to avoid adverse effects and to prevent unauthorized access to – and handling of – information (ref. 4). The dependability aspect will therefore result in requirements on the software with respect to

- Availability – e.g., to inform when sensors have ceased to function (to avoid interpreting missing information as there is nothing to report),
- Reliability – e.g., to satisfy the functionality and performance requested for the system to fulfil its tasks,

- Security – e.g., to prevent information leakage, distortion and erasure,
- Safety – e.g., to prevent unintentional damage to people, property or environment.

Ideally, the four aspects above should be addressed already during architectural modelling of the topmost system and pursued for underlying levels. As part of this work, any potential violation of these properties has to be identified, accompanied by suitable measures to mitigate them – a difficult task, at least for large system-of-systems. Apparently, this is the case for safety, and yet safety considerations are often neglected or postponed to later stages. Such an approach, however, is both risky and intolerable, especially for systems used in the civilian society, where the acceptance towards losses is lower than in a strict military setting.

Contradictory properties: Properties like adaptability and flexibility may easily conflict with determinism and simplicity, which are fundamental to safety (as mentioned in the end of section 'Introduction'). The question is how these inconsistent properties can be reconciled or at least balanced. Will determinism be of less importance for the super-fast, proactive systems in the future? Is it possible to create proactive systems, rather than reactive systems, in order to meet the request for quick deployment of the system?
Before answering these questions, some of the emerging computing models and technologies will be described.

## Future computer technologies

To make a forecast of the development within any field 10-20 years ahead is both audacious and futile. One of the difficulties, as for any prediction extending beyond a five-year period, is to avoid thinking along predefined paths. An additional, and almost impossible challenge, is to single out those untrodden research paths with a potential impact on the technology development. What might be easier, however, is to predict which advances within ongoing research that will be most promising for the future.
In the following subsections, an attempt is made to look at some emerging technologies that may enhance the computational capacity needed to fulfil the flexibility, adaptability and openness requested from the systems of tomorrow.

Nanotechnology – Nanoelectronics: Among the basic research areas with an expected impact on computing is *nanotechnology*, a comprehensive field comprising a multitude of disciplines addressing objects within the nanometer scale ($10^{-9}$ m). This is a field that might yield immense opportunities to increased performance and miniaturization without requiring a revolutionary way of thinking concerning how to build electronic-based systems. The same type of architectures, solutions patterns, abstractions and programming languages can be applied. Bit-setting, today requiring a current of $10^4$-$10^7$ electrons, may, with advanced isolation, be achieved with _-1 electrons in the micro-voltage range. The need for cooling is thus eliminated. Also, techniques to pile nanometer particles can be used to avoid wiring and consequently leakage currents between adjacent wires. This type of wireless communication, where the particles are lined up by an electrical field, may, together with particle growth (an alternative to etching) and new combinations of material, isolation and manufacturing techniques, push the miniaturization beyond the 45 nm size reached today. At the 10-20 nm size, however, the physical limitations of the silicon-based component technology, for which Moore's law was formulated, will be reached (a doubling of the number of transistors on an integrated circuit and the computing power of these circuits every two years). Other approaches will be needed to increase the computational capacity further, (ref. 5 ).

One such achievement is the creation of single-electron transistors based on *graphene*, a one-atom-thick 2D-crystal of carbon. Graphene has the unique features of being stable and highly conductive at room temperature even in sheets only a few nanometers wide. This allows so-called *quantum dots* to precisely control movements of single electrons, interconnects and logic gates (ref. 6). This breakthrough shows promise of succeeding in substituting silicon in our electronics within the next 20 years, turning them into graphene-based *nanoelectronics*.

Molecular nanotechnology – molecular computing: While nanoelectronics mainly applies a top-down view in building material and devices, *molecular nanotechnology,* MNT, uses a strict bottom-up approach by starting the construction from the molecular level to create successively larger and more complex products. Here, the desired structure and functionality are built by directing, with high precision, the individual atoms in their specific locations. The vision is to enable manufacturing of numerous inexpensive, but yet powerful, molecular machines and robots with dedicated tasks within medicine, environmental engineering, computer technology etc., (ref. 7). In this way smart material with different types of monitoring, reactive and self-repairing capabilities can be constructed and used in devices integrated not only into the environment but even into living bodies. Despite its merits, MNT has raised considerable concern regarding its potential for self-replication and variant-building through random mutation. These are features that might be destructive, and which, unfortunately, are not accompanied by the possibilities for selective elimination that the biological evolution has provided. Ethical design principles, built-in safety restrictions and corrective measures are among the suggested remedies. In case the technique will be mature enough for practical applications, cooperation between authorized engineers from different disciplines will be needed, supported by automated processes, a new type of tools and building blocks at the molecular scale. To make reliable predictions about the outcome of this research and the time range for this is difficult, but the typical guess is 15-30 years.

*Molecular computing* is a disciplinary area based on biochemistry and molecular biology using a huge number of DNA and enzymes to store and process information. A single DNA molecule can be used for input as well as fuel for the process. Input data, represented by a base-4 code using the nucleotides A, C, G and T, can be read by enzymes. The actual blend of DNA molecules defines the software program to execute. The simple operations that can be performed result from reactions between the DNA molecules. This allows each DNA molecule to act as a separate processor, albeit at low speed, e.g. a factor $10^{-9}$ compared to the silicon-based computer. Despite this, complex combinatorial problems can be solved quicker by means of massive parallel DNA-computing between the molecules in a test tube, than with a network of conventional processors. As an example, a tiny DNA computer residing in a volume of one $cm^3$ can hold $10^{13}$ DNA molecules being capable of storing $10^{13}$ bytes of data and performing $10^{13}$ simultaneous calculations. In this way NP-hard mathematical problems, such as the 'travelling salesman' (finding the shortest round-trip route visiting all of a number of cities only once), can be solved in days, rather than in years as for the silicon-based CPU (refs. 8, 9). This comes to a price, though: the number of molecules needed grows exponentially with the size of the problem.
Still in its infancy, molecular computing is interesting, not for general-purpose computing, but rather for the special types of mathematical problems that could be answered by a simple yes or no. It will involve a completely new way of thinking, new solution strategies, computing environments, tools, and safety challenges. To speculate whether the technique eventually will be mature enough for professional computers, and when, would at this stage be futile.

Amorphous and cellular computing: The basic principle of this paradigm is an immense parallelism offered by a vast number of autonomous computational units ($10^6$-$10^{12}$ cells) with limited memory and computational power – all identically fabricated and programmed to follow the same rules. The cell provides simple computation and local communication with the closest neighbours. The interaction is asynchronous, and no detailed knowledge of the location and orientation of the communicating units are required. Not even do all units need to be reliable. In this way large and robust self-organizing systems can be formed, regardless of the exact number of cells and how these are arranged. This is a research field still in its infancy. Yet, an inexpensive production of micro-mechanical electronic elements have been demonstrated, integrating logic circuits, micro sensors, actuators and communication units on the same chip. Among the prospective applications are elements included in building material and coatings, e.g. for monitoring and control purposes. Already, wireless networks of millimeter-sized sensor elements are commercially available (ref. 10).

C*ellular computing* rests on micro- nano- and biotechnology together with ideas from the cellular automata model. As an example, digital-logic circuits within biological cells are envisioned, where the branching in a logical gate is controlled by enzymes. This allows several types of gates to be built depending on which protein the gate should use for different signals (ref. 11 ).
Disconnected from the concept of a central controller, this paradigm involves an entirely different way of building computers and systems. The indistinct control of the amorphous, agent-based system, executing without human interaction, will require new architectures, programming methods and safety approaches. As an example, safety-constraints need to be specified for evaluation during execution, in order to compensate for the existence of unreliable elements.

Quantum mechanics – quantum computing: The laws of classical physics are not applicable to phenomena at the atomic scale or smaller – they have to be replaced by the principles of *quantum mechanics*. While many of these theories may seem counter-intuitive, they have nevertheless fertilized other areas, such as nanotechnology and electronics. As the technology development has approached the subatomic level, advancements in information processing has followed.

An example of this is *quantum computing* based on a quantum mechanical system of entangled elements, (refs.12, 13). The selected element could be either of an elementary particle, photon, ion or electron etc. The quantum state of the selected element type is then used as the fundamental unit of information. This quantum bit (*qubit*) has the

potential to exist in all states simultaneously. The state of an electron, for instance, could be spin up and down, along with all states in-between, as a *superposition* of these states. The theory also includes the probability to be in a specific state. This means that the discrete logic used in conventional computing – based on binary states (e.g. either current on or off) and represented by 1 and 0 – is replaced by a continuum. This involves also a move from deterministic to probabilistic models. In this way, calculations on all possible states of an element can be performed concurrently, allowing massive parallel processing. However, the measurement of the state of a qubit in superposition – or any other interaction between the environment and the qubit – will make this collapse into a single, definite state (*decoherence*). As measurement is the way to obtain the final result of a quantum computation, this means that any intermediate result has to be acquired through some other, indirect technique to allow continued fault-tolerant computing, despite decoherence. Also, the effect of external disturbances has to be considered. The problem might be solved by use of another amazing property of the system, the *entanglement*. This relation, arising from elements generated from the same origin (e.g. photons emanating from the same process), creates a dependence making them act as a single unit. Consequently, a measurement on one of them will also affect the others, regardless of distance. A suggested technique is to extract any discrepancy between their states without measurement and then use this information to restore superposition and entanglement.
The quantum computer is still in the experimental stage. So far, only small systems of 12 (perhaps 16) entangled qubits have been demonstrated. Provided these could be scaled up (>$10^3$ qubits) and made robust to decoherence, hard problems could be solved exponentially faster than any other computer available today. Also, these would offer a storage capacity exponential to the number of unmeasured qubits. With such an immense computing power, problems classified as intractable could be solved, e.g., vast simulations and database searches (a quadratic performance increase), factorization of large numbers (e.g. $10^{200}$ digits in seconds) – and consequently – decryption of all factorization-based encryption schemas. As a fundamentally new paradigm for information processing, quantum computing, once realized for practical usage, will require a completely new type of algorithms, solution strategies, development tools and environments as well as novel approaches to achieve dependability, i.e. availability, reliability, safety and security.

A concluding observation from this limited survey is that it seems possible that current research might provide enough computing power in the future to allow realization of the proactive systems described in the previous section.

## Measures towards safety consistency

The problem, when combining properties related to safety with those for openness, adaptability and flexibility, is that previous safety analyses must be repeated for the modified configuration in its new context. This is a demanding task both with respect to time and effort: any new risk introduced has to be identified and mitigated before deployment of the new system configuration.
Apart form the traditional software safety requirements (e.g., ref. 2): What will be required from the people, processes and the production environment to succeed in this task – provided that enough computing resources are available?

Production and target environments: Two alternatives can be seen for these support environments.
One is that the new mix of system components has been analysed in advance with respect to safety (e.g. during development) and found tolerable for the system in its intended environment, usage and operational profile.
The other is that techniques and tools supporting safety assessments on the fly are available.

The first alternative, achievable even with today's technology, will require a correct, complete and consistent specification of the – from a safety perspective – tolerable system configurations. This is necessary in order to enable elimination of unacceptable combinations. Even a seemingly harmless change to the system (a new COTS version, a different operational condition or usage profile etc.) may impose serious consequences.

The second alternative assumes a revolutionary evolution of the underlying computer technologies: an immense increase of the computing power capable of processing vast volumes of information and providing super-fast execution. Envisioned is a scaled-up realization of what presently only is available in isolated laboratories, be it nano-, bio- or quantum computing. The massive parallel processing and the extreme miniaturization these will offer would make it possible to create proactive systems. Simultaneous simulations, testing, acceptance-rejection of results, and searches for acceptable system configurations would then be possible. At the same time, the system variant (with the requested properties) could be built before needed in operation. This assumes an extremely high degree of automation of, for instance, evaluations, which have to be based on correctly specified acceptance rules. Formal technique will therefore be an essential ingredient, useful also for the other tools, e.g. to support (ref. 14)

- Formal specification of what the system shall do and must not do, respectively,
- Analysis of formal specifications for timely evidences that the requested system configuration will fulfil specified requirements without safety compromises (e.g. no violation of specified safety rules),
- Immediate build of a sufficiently safe configuration from given specifications,
- Regeneration of components from relevant preconditions (instead of reuse of a previously developed implementation),
- Generation of documentation from up-to-date specifications, architectures and models,
- Coupling between the analysis tool and the execution environment allowing the execution to be stopped, if continued operation would violate specified safety rules (to overcome the need for special simulators).

All these tools would need to be correct (formally verified). Also, all requirements need to be specified formally (to enable automatic analysis and verification).

Software Personnel : The only candidate of the emerging technologies mentioned in the previous section not involving a paradigm shift concerning software development seems to be the field of grapene-based nanoelectronics. This means that computing still could rest on the sequential von Neumann model, where time is represented by an ordered set of events driven by discrete clock ticks.
But, a successful deployment of any of the other enabling technologies will require a new way of thinking. Thus, new logic may evolve as a foundation for new programming languages, primitives, abstractions and building blocks. This will call for a conceptual reorientation with completely new kinds of solution strategies.

Software Processes: What requirements will future software systems impose on the processes for its production and support? What opportunities will the technological development offer?

- Control processes: These basic processes – addressing project management, configuration control, quality assurance, quality control and engaged in the plan-do-check-act cycle – are proactive by nature. As such, they are all essential for security and safety. But, the realization of future computers with their extreme computational power will increase the opportunities to create disorder. Even more proactive and structured measures will thus be required with support from automated tools. The basic control tools therefore need to be combined with automated analysis, verification and validation activities. This will relieve humans from actual execution and control, in favour for preparations, something that will further strengthen the proactive capability.
- Production processes: In developing the flexible and adaptable systems requested for the future, evolutionary / progressive life cycle models with short lead time will be needed. These have to be supported by well-defined specification procedures, highly-automated and verified tools, along with super-fast processors (allowing computation faster than real-time). These prerequisites would then enable, e.g.,
  - System upgrades through new analysis- and generation tools (rather than through new components and platforms)
  - Immediate Build from specifications and models (instead of Daily Build from implementation components)

Much of the complexity is thus transferred from the system's implementation phases to the supporting tools and the specification phase.

## Challenges in the near future

As long as we do not have access to graphene-based nanoelectronics, quantum computers, and the like, what can be done today in order to meet requirements conflicting with safety?

Flexibility: To facilitate smooth updates of an already implemented system with new software components, subsystems, or just a new software version, well-defined interfaces are necessary. These should reflect, not only the functionality and behaviour provided by the software, but also all assumptions and conditions on which the realization resides. Often, these types of preconditions are not manifest in the software interface and may also be missing in its documentation. Careful analyses are needed to find the varying aspects that may come into question in the presumptive systems, environments and user profiles – aspects that need to be considered in the early design of the software. Among the problems are how to reflect and overcome initial, often implicit assumptions. These could be the presumed speed range of an observed or controlled object (cf. refs. 15, 16), the rate or the volume in which information is expected to be entered into the system, etc. Such assumptions need to be expressed explicitly

in order to build a system that will be able to take appropriate action in time, even when these assumptions would be violated or false. Provided that these type of preconditions could be specified (e.g. in the software interface) integration could also be denied whenever any precondition not is met. Much of the safety violations emerging during execution could in that way be prevented.
Fundamental to the ease at which system upgrading could be performed, is to which extent the system is based on classical design principles such as modularization, object-orientation etc. The effort of updating a system that also is safety-critical depends, in addition to that, on how well-partitioned the system is with respect to, e.g. parts of different criticalities (ref. 17). As long as these aspects not are considered in the early design phase, the effort that later has to be spent on updating both the system itself and the previously performed safety analyses will be substantial.

Adaptability: The ability of a system under operation to adapt to new or changed situations requires a behaviour that is dynamically adjustable (i.e. during run-time). To find and provide the relevant behaviour under different circumstances require close analysis and design of both the software and the possible changes that may occur to its context during system operation. Different solution patterns, such as alternative realization parts or execution paths, can be made selectable during run-time via dynamic dispatching, variable control, etc. Again, this variability must be limited to avoid combinatorial explosion and guarantee a deterministic behaviour allowing prior analysis of potential unsafe execution patterns. The combinations found not safe enough must be blocked out prior to execution.
To rule out any unsafe behaviour that still may arise during operation, a diverse set of fault-tolerant, failure-aware and safety-related techniques may be applied. Safety requirements and design restrictions are used to specify what the software system is not allowed to do, and the implementation is built to fulfil these restraints. Furthermore, the implementation may be provided with alternative execution paths or complemented with special protection features activated upon demand. Different programming concepts exist to support this, e.g., design-by-contract, invariants (conditions that always must be true), pre- and post-conditions (evaluation before and after execution of a code block), type constraints, exception handling, (refs. 18, 19).

Openness: An open architecture is not *per se* a feature compromising the safety of a system. On the contrary, it might be seen as the natural base on which a safety-critical system should be built. However, in order for safety to be met by the system as a whole, a safety-oriented view of the architecture is necessary, addressing every system in the hierarchy. This safety view expresses the system-wide safety strategies, a complement to the overall principles (mentioned earlier), that included systems have to fulfil. The safety principles need to prescribe, e.g., on which level and by which system in the system hierarchy, the different safety threats should be met. This is important, especially when the included systems originate from different vendors and it is not clear, who is responsible for mitigating an overarching safety threat. In addition, this safety view will have to describe the special safety mechanisms needed in the architectural design to fill the remaining holes in the safety net. This safety view must be shared by the safety-oriented architectures on all levels, in order to support the joint actions required in the strive for safety at the topmost level. Otherwise, the smooth component replacement afforded by an open architecture, will, for a safety-critical system, be undermined by the increased effort in performing the necessary safety analyses and safety mitigations.

Design for reuse: A success factor, in the effort to construct a system that is flexible and/or adaptable, is to apply the principles of 'design for reuse'. Among the key issues are, to identify the main abstractions and its primitives, the component interrelationships and the user interface. A combination of generality and speciality is often needed, where generality stands for isolation of the abstraction/framework, and speciality for the development of variants.
These variants may implement special aspects of the context in which the software is intended to be integrated (ref. 20). This type of encapsulation is supported by classical language concepts such as procedures, packages, classes and methods, available through procedural languages and object-oriented technique. The reusable components can be realized via a family of variant components for various context requirements (ref. 21) or via skeleton parts with slots for insertion of specific variant parts. Different approaches can be applied: composition and generation. In *composition* the components are static (essentially atomic) and reused directly or created via special component constructors from more basic fragments. In *generation* the components are created dynamically from code fragments or from a basic pattern.

The inability of the traditional reuse techniques to cope with cross-cutting concerns scattered throughout the system has motivated the development of Aspect-Oriented Software Development and Aspect-Oriented Programming (AOSD/AOP). Special for AOP is that it allows a basic feature spread over a number of modules to be captured, encapsulated and reused as a separate module (called *aspect*). Examples of such cross-cutting concerns are logging, security and possibly safety, even if this last aspect not is mentioned by those advocating the technique. The aspect mechanism allows a certain code fragment (*advice*) to be associated to specified places (*join points*) in the main program, where the aspect is invoked. It also provides inter-type declarations (an open class allowing declarations in one place to affect other classes), etc.

From a safety perspective AOSD has its benefits: For safety, being a system-wide concern, the aspect abstraction seems natural. The advice concept, embracing the three types *before*, *after* and *around*, could also be useful, e.g. for safety checks, pre- and post-conditions. Moreover, the aspect mechanism simplifies the implementation structure, easing the maintenance effort. Old features could easily be changed and new features added. At the same time, multiple implementations of the same feature across the system are avoided.

On the other hand, some AOSD features may introduce unwanted side effects: Aspects added to a later system configuration could result – if not in a non-deterministic behaviour – at least in an unpredictable deterministic behaviour. A simple change of one aspect may have ripple effect on the entire system. The safety analyses of the design and implementation will be complicated by the fact that every aspect in every possible join point also has to be considered. Furthermore, the previous tests (test procedures, test cases and test results) may be outdated and hence not reusable (refs.22, 23).

A consequence of the above is that, to allow safety-critical as well as security-critical applications to be built on the aspect paradigm, the different AOSD features need to be analysed more thoroughly, as has been done for the other reuse techniques (e.g. , ref. 24). Among the things to explore are the possibilities to impose restrictions of its usage.

## Summary

Safety is an essential property, especially for systems in our every-day environment that need to perform complicated and potentially unsafe operations in real time. This type of systems often requires large investments in time and money. The incentive to keep them operational over an extended period of time is therefore high. In order to achieve this, the system is required to be flexible to allow easy upgrading, open to enable component replacements and adaptable to changes in the operational context. These properties may interfere with safety and other qualities on which safety depends. This paper has discussed what will be needed now and in the future from the system itself, the processes used in building the software systems and the people involved, to balance these conflicting properties.

## References

1. N.G. Leveson, Safeware: System Safety and Computers, ISBN 0-201-11972-2, 1995.
2. H ProgSäkE, Handbook for Software in Safety-Critical Applications, M7762-000621, Swedish edition: 2001, English edition: 2005.
3. I-L Bratteby-Ribbing, Software Safety activities in Sweden, ISSC 2007.
4. J.C. Laprie, Dependability: Basic Concepts and Terminology, ISBN 0-387-82296-8.
5. Nanotechnology, Encyclopedia Britannica. 2007.
6. A. K. Geim et al, The rise of graphene, Nature Materials, Vol 6, Mar 2007.
7. K. E. Drexler, Nanosystems: Molecular Machinery, Manufacturing, and Computation, Wiley sons, 1992.
8. L. M. Adleman, Molecular Computation Of Solutions to Combinatorial Problems, Science, Vol.266, no 5187, Nov 1994.
9. S. Lovgren, Computer Made from DNA and Enzymes, National Geographic News, Feb. 2003.
10. H. Abelson et al, Amorphous Computing, Computer Science and Artificial Intelligence Laboratory, June 2007.
11. M. Sipper, The Emergence of Cellular Computing, IEEE Computer, vol. 32, no.7, pp.18-26, July 1999.
12. A. Hagar, Quantum Computing, Stanford Encyclopedia of Philosophy, Feb. 2007.
13. D. Deutsch, IT from Qubit, Centre for Quantum Computation, The Claredon Laboratory, Univ. of Oxford, Sep. 2002.
14. I-L Bratteby-Ribbing, Development Trends within Software Engineering and Software Safety (translation from Swedish of "Utvecklingstendenser inom dator-/programvaruteknik och programvarusäkerhet", FMV, KC Ledstöd 14910:18897/04, 2004.
15. J.L. Lions, Ariane 5, Flight 501 Failure, Report by the Inquiry Board, 1996.
16. GAO/IMTEC-92-26, Patriot Missile Software Problem, Feb 1992.
17. J. Rushby, Partitioning in Avionics Architecture: Requirements, Mechanisms, and Assurance, NASA/CR-1999-209347, june 1999.

18. J-M Jézéquel, B. Meyer, Design by Contract: The lessons of Ariane, Computer, Vol.30, no 2 pp 129-130, Jan 1997.
19. K. Waldén, J-M Nerson, Seamless Object-Oriented Software Architecture: Analysis and Design of Reliable Systems, Prentice Hall, ISBN 0-13-031303-3, 1995.
20. I-L Bratteby-Ribbing, Reuse Design with Ada, Bofors Electronics AB, PM 145611, 1990.
21. G. Booch, Software Engineering with Ada, ISBN 0-8053-0604-8, 1986.
22. R. Chitchyan et al., Survey of Analysis and Design Approaches, AOSD-Europe-ULANC-9, May 2005.
23. G. Pollice, A look at aspect-oriented programming, Worcester Polytechnic Institute, Feb 2004.
24. G. Huffman et al., Some Pit-Falls in Object-Oriented Programming, Proc. of the 23rd ISSC Conference, 2005.

## 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 Defence Research Agency (FOI), the defence industry (e.g. Saab 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 and produced a handbook for software in safety-critical applications (ref.2).