Methodologies of safety-related Software development

J. Börcsök
HIMA Paul Hildebrandt GmbH + Co KG
Albert-Bassermann-Str. 28
68782 Brühl
Germany

S. Schaefer
University of Kassel
Department of computer architecture
and systems programming
Institute of computer science
Germany

Abstract: Safety-systems mostly comprise hardware and software solutions. Until today, a lot of fixed wired systems are still operating without a microprocessor, i. e. without the assistance of software. Due to the increasing application of such complex hardware and software systems, the software systems have to be considered regarding safety as well as hardware systems. The development of a safety-related software system is similar to the development of a safety-related hardware system. However, the calculation of reliability and availability for safety-related software systems is far more complex. Mathematical approaches are derived from the models for safety-related hardware.

Key–Words: software-development, safety-related systems

1 Introduction

This paper discusses the methodical analysis of software used in safety-related applications. It provides an description on a safe computer system software technology and specifies in more detail the required test procedures in the last section. However, the paper does not claim to be complete, since studies and methods have rapidly increased, particularly with respect to object-oriented software system design and programming design ([3]).

For several years, no combined software-hardware solutions have been used in high risk environments in which safety systems are usually employed. The first developments in software environment did not follow methodical and/or structured procedures. Before methodical program development procedures and structured software design techniques were used, the program code was generated when the programmer believed that the problem was understood. The result of this approach was a set of extremely expensive test procedures leading in many cases to the rejection of the software, to reprogramming and improvement measures and finally to completely rewrite the software or parts of it.

In the early Seventies, the existing idea was to approach software development in a structured way. The well-known waterfall model with its severe phase concept started to get accepted in academia and industries. The negative aspect of the waterfall model is the considerable documentation effort resulting in a great documentation quantity, even before a single software part is developed. In the last few years, several software developments and testing procedures were established. Unfortunately, not every approach achieves the same performance. Therefore, the selection of the software development strategy is important for the project. The IEC 61508, the standard application valid for safety systems, even describes tool support for designing safety-critical software and gives a structured overview of the safe software development ([3]).

The remaining sections of this paper presents different design models to be followed during the development phase such as the Waterfall model or the V-Model and the advantages and disadvantages are discussed. Furthermore, the paper describes different software development procedures and safety requirements for safety related systems. In section 7, different implementation procedures for the development of reliable software are detailed. Section 8 presents different methods for proving and testing the software. Finally, a short summary concludes at the end of the paper.

2 Waterfall Model

In the waterfall model ([9]) the software development phases are strictly separated from each other. The following phases are characteristic:

- Requirements analysis,
- Design,
- Specification,

- Implementation,
- Integration,
- Operation.

Following a inflexible up-down method, a new phase may only begin if the previous one is successfully completed. If problems, resulting from an upstream phase occur, development has to continue from this phase. A consequence is that in some cases parts of the performed development work has to be rejected. The waterfall model is characterised by a clear and systematic procedure juxtaposed too little efficiency and flexibility. In praxis, projects may not be elaborated in such linear form and it might be necessary to adapt already completed phases. Figure 1 shows a typical strategy to develop software for safe systems.

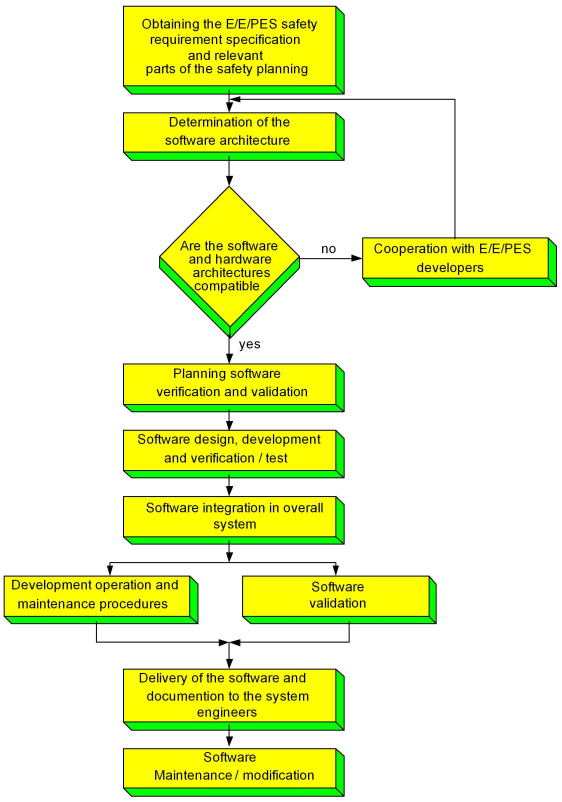

Figure 1: Structural Software Creation for Safe Systems

3 Spiral Model

The software design in the spiral model is divided in repetitive phases, until the desired result is achieved. Therefore, the result is a repetitive design cycle instead of linear procedure like in the waterfall model ([2]). Typical phases are:

- Object definition,
- Alternatives and risks analysis,
- Selection of an alternative,
- Development,
- Testing and result assessment,
- Next step planning, depending on the test result
- Original objective modification, if required.

The spiral model is much more flexible compared to the waterfall model.

4 Rapid Prototyping

Using this model, a prototype with a particular specification is created as soon as possible. The prototype is equivalent to the end software product without the complete range of functions. The prototype is tested to evaluate the design, specifications and requirements ([1]). If necessary, a new or modified specification has to be created. Finally, the software is further developed to the final software product in accordance with the updated specifications. It is typical for this design to create different prototypes. The advantage in this case is that, with the person requiring the software accompanying the development process, the product corresponds exactly to the customer's instructions and desires. Fault or undesired development approaches are early recognised and corrected. Because no fixed specification exists, the requirements are still modified in the implementation phase. Therefore, the resulting software is "grown" and its design is not optimal.

5 V Model

Nation wide, the V model is increasingly popular when developing software for governmental or military customer. The V model is a generic description, which does not describe any tools or methods ([8]). For a specific project, the actual procedure has firstly to be determined. Therefore, the V model can perfectly adopt to the defined objectives. The V model uses specific models to support project management, system creation, quality management, and configuration management.

6 Software development for safety-related systems

In industries and academia, the development of safety-related software proceeds in three general phases:

- Defining objects for the reliability of the software to be developed

- Taking appropriate measures for achieving those objectives
- Using verification methods to quantify the success.

Generally, this partitioning is not sufficient for a formal structuring of safety-related software development. It is necessary that the programming models adapt precisely to the specific problem. However, the next section describes fundamental measures and techniques for developing safety-critical software based on the simplified partitioning above.

One of the most important phases, whether with software or hardware, is the phase of creating the specification. During this phase, not only the functional requirements for the program and/or hardware are described in detail, but also the required characteristics in terms of operating safety.

During the second phase, also known as the actual software implementation, the developer should use not only techniques leading to a possibly fault free program code, but integrate specific functions into the program. The latter does not help accomplishing the actual task, but detecting and repairing faulty states during operation.

After completing the software, the third step begins with a testing phase to determine if the specification is correctly implemented. For software with high safety and reliability requirements, it has to be tested if the desired operating safety is achieved. This is achieved with different methods, such as correctness proof or empirical test procedures.

6.1 Determining the Safety Requirements

As mentioned earlier, the reliability and safety requirements for a specific system including hardware and software result from the intended application. The higher the hazard potential, the more severe the possible damages in case of incorrect functioning, the higher the requirements for developing the system.

Specific standards, such as DIN 31000/VDE 1000, DIN V 19250 or IEC 61508 [5], describe measures for quantifying the hazard potential and, starting from this point, a rating in risk or requirement classes. While analysing the safety requirements important for the system to be developed, it has to be verified, which features are essential for the software. These features have to be described as a specification part and considered when test procedures are applied.

7 Implementation Procedure

Generally, to maintain a safe and reliable software during implementation, the developer should follow the common rules contributing to as few faults as possible and code readability. The systematic application of correct programming techniques is also economically profitable. The next paragraphs briefly structured programming, modularisation, object-orientation, coding rules and how to prove the reliability of software.

7.1 Structured Programming

For applications with safety-technical background a large range of different programs are essential. When developing safety-relevant software, a structured programming style has to be developed. In this case, only three different sequence constructs are allowed:

- Sequence (normal command sequence)
- Branching (for example if...then...else)
- Repetition (for example loops with for or while)

Limiting the constructs to these three fundamentals, has not only the advantage to reduce programming faults and improve the code readability, but also eases the correctness proof using formal methods ([4]).

7.2 Modular Design

Structured programming alone cannot help controlling the increasing program complexity. This and the fact that several programmers have to work on the safety related program simultaneously are the reasons for software modularisation. Each software part represents then a more or less cohesive block comprised of data and algorithms belonging together thematically or functionally.

Due to the obvious smaller module size compared to an individual program, such a module can be far better managed and tested. Faults can therefore be avoided and/or detected more easily. In this way, for each module a rather high reliability level may be achieved. A large and complex program comprising several modules will consequentially have a reduced number of implementation faults. To achieve optimal results in accordance with this concept, some specific procedures have to be followed.

Module interfaces have to be defined and documented. A developer wishing to use an outside module and access it from his module, has to consider the other module as a ”black box”. This means that the developer is able to use the outside module just supported by knowing the interfaces and not having any knowledge on its internal functions (capsuling). The advantage is that internal algorithms of the module may be modified and optimised, without effecting other program parts, as long as the conformity with the interface specification is given.

If the module coupling is kept as small as possible (”loose coupling”), the faults occurring in a module do not affect the functioning of the other modules. The module interface has to be designed in such a way that an access from outside is only allowed for essential data and functions.

7.3 Object-Orientation

The approach of the object-oriented programming style is to create a program as problem reproduction in such that the structure and commands of the program conceptual formulation is reproduced ([7]).

Further, the problem is divided in entities each representing a cohesive partial aspect of the problem. Every entity is then modelled in the program code and represented by an object. The same objects belong to the same class.

A class consists in actual things or ideas, abstracted from the problem context. All members of this class have the same characteristics or features. Starting from a quite general base class, further precise classes can be derived by inheritance. Features defined for the base class are automatically inherited to the derived class. Therefore, common features are defined in common base classes and the classes implicitly derived from them, have this features plus some additional peculiarities. A base class describes a characteristic or capacity more abstract, whereas the description provided for the derived classes is always concrete. Base classes often only help to describe general concepts. Parameters and objects actually contained in the program exist only because of the derived classes. Therefore, a class represents a cohesive description of a sub problem.

Requirements specific for a module such as object minimisation in an interface or practicability of the ”black box” principal, have to be met during the class implementation. Due to the object-oriented programming, fulfilling modern requirements such as: data capsuling, avoidance of global data structures, repetitively, readability, maintainability etc., a particular elegant and easy approach is possible. All these features represent main requirements for safety-related software development.

7.4 Coding rules

When developing safety-oriented software, coding rules are described by different test houses, even if both approaches of structured programming and object-orientation were applied consistently ([10]).

Without claiming to be complete, the next paragraphs lists further guidelines, which a programmer should consider when creating safe and reliable software to fulfill the testing requirements demanded by a test house.

When implementing algorithms, clear structures should be preferred, as particular artful constructs do not necessary lead to increased efficiency, but the code can be difficult to understood.

The testing and verification methods should be already considered during the implementation phase. The documentation has to be detailed and has to accompany the complete project. Tested functions or class libraries for a specific task should be used preferentially. A part or the whole machine code can be stored in a ROM rather than in a RAM to ensure the program unalterability.

No undocumented or undefined language behaviours should be exploited, as the program functioning with another compiler, libraries from other manufacturers or another operating-system version is not ensured.

When using real-time and multitasking applications, one should consider that the resources are blocked as short as possible. The different, synchronous executed program threads have to be correctly synchronized when using common data or resources.

The actual program code implementation should meet the instructions specified in the coding rules. The result is a good readable and easily verifiable source code. These two qualities are essential and desirable with several proof procedures to verify the correctness of the program.

8 Proving the Reliability

Informal proof procedures may already be used during the development phase. Due to this measure, faults can be detected in an early stage, leading to improved quality and economic advantages. Such informal procedures are: inspection, review and walkthrough ([6]). A comparison of different informal proof procedures are presented in Table 1.

Using appropriate tools, structural analysis may be performed. An extensive automated semantic study of the control and data streams may be performed based on the program code.

The program correctness proof is carried out in accordance with severe formal and mathematical methods. To provide such a proof, it is reasonable and often necessary to make the program ”proofable”.

A fundamental method for testing the software quality is the execution of tests, aiming to verify if the program with a specific input behaves in accordance to the specification. However, when testing complex programs, the procedure cannot be complete, i.e. it cannot be executed for the whole input range.

Table 1: Comparison of Informal Proof Procedures

Criterion	Inspection	Review	Walkthrough
Group composition	Moderator, author, tester, user	Project team, client, contractor	Author, colleagues, testing person
Member number	3-6	5-15	2-6
Duration	2 hours max. plus preparation	1-2 days	2 hours max.
Objects	Documents, including agreements, codes	Phase products, Project plans, project reports, problem field	Documents, SW preliminary design, SW detailed design (e.g. codes)
Objective	Documents testing	Analysis of project states, problem solution, measures	Detecting faults and incompleteness
Moment	After document creation	Phase ending, milestone	After document creation
Execution	Check lists	Agenda	Idea exposition in front of a public
Result	Protocol with tested doc, new inspection if nec.	Protocol with decisions	Protocol with faults, incompleteness

8.1 Inspection

During an inspection, a group of developers or system designers verify the current software project state using checklists. The attention is turned rather to the formal content and document correctness and less to the program correctness. Documents compiled in different project phases are compared with each other to verify the development. During the actual inspection meeting, the detected defects are gathered. The actual program developers have no firm task during the inspection. They should neither explain nor describe their work as the documentation is examined on comprehensibility.

In the inspection wrap-up phase, a to-do list for repairing the detected defects is created. The meeting moderator should take care of defining the execution changes, but does not have to control its success, as this will be part of the next inspection.

8.2 Review

During a review, not only the quality and the state of the developed program are inspected, but suggestions for improvement are recommended and discussed. In this phase, modifications or integrations to the project guidelines are possible.

A review takes place at the end of a development phase and is conducted by the project team members. It has to be verified that the intermediary results, defined in the project plan, have been achieved and if the result corresponds to the requirements specification in form and content. The review results, including suggestions and new decisions, are recorded.

8.3 Walkthrough

During a walkthrough, the program is systematically tested on the functional or source code level. The software developer explains the program and with his assistance it is checked for faults or inconsistencies. In contrast to inspection and review, the software developer is always active during a walkthrough. This proof procedure includes a more detailed analysis of the object to be tested, compared to the previously mentioned procedures. Walkthroughs rank among the best modern instruments to proof the design correctness. Hidden implementation faults can be early detected, but a great effort is involved.

8.4 Structural Analysis

During the structural analysis, formal software features are investigated using automated tools. Using this procedure, one can receive information about the control stream, the data stream, and the program semantic. The analysis tool creates an output in form of pseudo codes, formulas, algorithms, or graphical structure images, which can be tested using the specification. If the correctness of such an image can be immediately verified, this procedure can be roughly compared to the program correctness proof. Furthermore such an analysis can help deducing reasonable test procedures for white-box or black-box tests.

8.5 Correctness Proof

The program correctness can be proved, using scientific mathematical procedures. Should the proof fail, faultiness is not automatically demonstrated. One can deduce from the failure an example situation, which could lead to a program fault. To use this procedure, an intensive training is essential. Furthermore it is useful, if not necessary, to design and determine the form of the program in such a way that this proof procedure can be easily applied. By using this concept appropriately, the program efficiency and development may improve. The effort increases disproportionately to the dimensions of the program to be examined. Consequentially, this procedure cannot always be utilized.

8.6 Testing

A test can be defined as a process to verify and validate a system or its components. Testing safety-critical software (not general functional tests) is one of the most important requirements in addition to the testing of hardware, when safety-related systems are developed.

The results of the test procedure can be used to evaluate integrity or safety. Test processes detect faults which have to be repaired for increasing reliability. The testing of safety-related hardware and software is complex. Generally, tests are performed in different phases during the system development. They are referred to as:

Table 2: Test Procedures in Different Life Cycles

Life Cycle Phase	Dynamical Testing	Structural Testing	Modelling
Analyzing and specifying the requirements		✓	✓
Top level design	✓	✓	✓
Detailed design		✓	✓
Implementation	✓	✓	
Integration tests	✓	✓	✓
System validation	✓		✓

- Module tests
- System integration tests
- System validation tests

Module tests include the assessment of small, clear hardware and/or software functions. Due to the program or component simplicity, detected faults at this level are often easy to locate and repair.

System integration tests investigate the characteristics of a module collection and aim for an actual module interaction. Faults detected in this phase are probably more difficult to repair than those detected during a module test because the test arrangement is more complex.

System validation tests should demonstrate that the whole system meets the system requirements. Occurred defects are often the result of lacks in the specification. They are extremely difficult to remove because these faults can only be corrected by reviewing the whole development process. The test can be performed in a dynamical or structural approach, or may be based on a mathematical model. Test procedures of different test cycles are presented in Table 2.

At the end of this section, the black-box and white-box test procedure are briefly presented. Test methods are often classified using the information available to the persons performing the tests.

In black-box tests, the test engineer does not know anything about the system implementation and has to use the information about specification as base for the tests. The black-box test checks whether the system provides the results defined in the specification. Black-box tests provide the highest level of independence between developer and the person executing the tests and are extremely profitable for independent validation.

In white-box tests, information about the system implementation is known. Most test methods are based on the white-box approach. This technique can be applied in all development stages, to hardware as well as software. Knowledge of the internal structure of an module simplifies the dynamical testing as tests can be developed specifically for each module.

9 Conclusion

The paper gave a detailed overview of modern software design and development for safety critical environments. It discussed different design models popular in academia and industries.

It described the requirements to develop reliable, safety-related software according to international and national standards. Finally, the paper focused on different methods for actually proving the reliability of the software, which are widely used in academia and software companies.

References:

[1] D.C. Andrews *JAD: A crucial dimension for rapid applications development,* Journal of systems management, 1991

[2] B. W. Boehm *A Spiral Model of Software Development and Enhancement,* In: IEEE Computer. Vol. 21, 1988.

[3] J. Börcsök *Electronic Safety Systems,* Hüthig publishing company, 2004.

[4] W. Ehrenberger *Software-Verifikation* Hanser, 2002.

[5] IEC/EN 61508 *International Standard 61508 Functional safety: Safety-related System,* Geneva, International Electrotechnical Commission.

[6] G.Myers *The art of software testing,* Wiley, 2004.

[7] B. Oesterreich *Erfolgreich mit Objektorientierung,* Oldenbourg, 2001.

[8] A. Rausch et al. *Das V-Modell XT. Grundlagen, Methodik und Anwendungen,* Springer, Heidelberg 2006

[9] W.W. Royce *Managing the Development of Large Software Systems,* Proceedings of IEEE WESCON, August 1970.

[10] VDI *Software-Zuverlssigkeit, Grundlagen, konstruktive Manahmen, Nachweisverfahren,* VDI-Verlag, 1993.