Safety and dependability analysis to complement testing of safety-critical software

Abstract. Computers are increasingly being introduced as an integral part of critical systems upon which millions of lives depend. Even the most expensive, fully tested and independently certified system can fail months or even years afterwards. These software products are usually very large and complex, so they cannot be fully tested and their dependability is never 100%. There are several techniques (still immature, not properly/systematically used) to be applied for software in critical systems as the contributors to its safety and dependability. All of them present important aspects that cannot be afforded within one single paper. This paper focuses on the positive effect of using fault avoidance techniques, and their influence to both the process and the technological aspects of the software development life cycle. Results from real case studies are shown for demonstration of usefulness of this method in different domains of application (space software, automotive devices, etc.)

1 Introduction

Importance of software in critical applications. Computers are increasingly being introduced into critical systems, such as defibrillators, avionics suites, nuclear power plant controls or anti-lock braking system in new cars, all these becoming an integral part of everyday systems upon which millions of lives depend [6]. Automatic cash machines, internet-based systems, mobile phones, communication satellites, administration and database systems, etc. are other critical systems that might not kill people in case of failure but on which our day to day life is based.

The safe and reliable operation of these systems cannot be taken for granted. Malfunctions of these systems can have potentially catastrophic consequences and they have already been involved in serious accidents. There have been some dramatic examples of failures that shown not only that software is used widely in many areas of our daily life but that software malfunctions cause, at best, inconvenience and irritation, but at worst, life threatening. The following well-known disasters attest it [3][4]:

• An example of catastrophic failure was the crash of the Korean Air 747, Flight 801, in Guam in 1997, where 225 of the 254 people on board were killed. National Transportation Safety Board investigators said that a software error might have been a contributing factor in the crash of the aircraft [16].
• Problems with the baggage handling system caused a delay of more than a year in opening Denver Intl. Airport with an estimated delay’s cost ranged as high as $1.1 million per day [5].
• In Seattle, computer controlled ferries caused more than a dozen dock crashes in the 1980s, resulting in damage worth more than $7 million. The state of Washington recommended spending more than $3 million to revert the operation of ferries to manual controls [5].

Finding out software-caused failures details. Despite what can be learned from the few available software-caused failure investigations, it is difficult to find out the details behind many existing engineering mistakes. The origin of many failures is almost never published and so nothing can be learned from them.

In any case, significant pressure is now being put on suppliers to provide ‘bug-free’ software systems, especially for safety critical applications. However, even the most expensive, fully tested and independently certified systems that are put through rigorous acceptance testing, can fail months or even years afterwards.
Complexity of critical software systems. It is a common characteristic that software products in use in safety critical applications are very large and complex (meaning here real time, with complex algorithms and interactions, etc.), so they cannot be fully tested and their dependability is never stated as 100%. This implies that one has to accept that there is a possibility of errors occurring in such software.

Safety and dependability. One should realize that there are many indirect failure cases caused by software problems in almost all areas of our life for which liability should be demanded. Safety and dependability are characteristics to implicitly or explicitly demand in almost all systems. They are related, but certainly not at all identical concepts. Dependability of a computing system is the ability to deliver service that can justifiably be trusted, while safety is a measure of a system running without catastrophic failures [2].

Therefore, software safety and dependability are related with software failures (as an end effect in itself when only considering the software product) and software faults. This paper deals with software failures and fault handling in relation with avoidance and/or reduction of failures and the consequences of software failures. Human originated failures are another vast research object, not being possible to approach it in this paper.

Techniques to avoid damages from software faults. Many simple techniques exist, which allow the critical software to be monitored for correct operation. Other techniques help in identifying, reducing, eliminating and even preventing these potential problems to happen. The current problem is that yet they are not used to analyse software products systematically and really complementing the in-bearable incomplete testing activities.

This paper examines some of the software fault analysis techniques with focus on fault removal techniques that can be immediately implemented to almost any critical software product.
2 Software fault removal

2.1 Introduction

Software fault prevention, tolerance, removal and forecasting [2] are the contributors or techniques contributing to safety and dependability of software. Fault prevention and removal are often grouped in the notion of fault avoidance, but we will keep removal as the preferred form in the remain part of the document, considering that when removal techniques are used in early phases, they are used as prevention techniques at the same time.

All of the above mentioned techniques are equally important for critical systems as already required by different standards ([7], [8], [9]). This paper focuses specially on fault removal techniques and their influence to both the process and the technological aspects of the software development life cycle.

To use these techniques when developing a software product, a relationship must be established between them and each development stage, the methods and techniques to be used to develop software, as well as with the different product architectures.

SFMEA and SFTA techniques. Traditional system safety and dependability analysis techniques such as SFMEA (Software Failure Modes and Effects Analysis) and SFTA (Software Fault Tree Analysis) can be successfully applied to systems with significant software content to complement dynamic techniques. Their greatest advantage is when they are used in combination with each other: SFMEA concentrates in identifying the severity and criticality of the failures and SFTA in identifying the causes of the faults [1].

The choice of these techniques is supported by:

- use from initial stages of development, helping to find out and remove potential failures earlier;
- integration with safety demonstrations of systems;
- both are well-known and popular as safety and dependability analysis techniques, such as in the following standards [11][12][13][14][15] and,
- their use do not require any special infrastructure.

Despite that, SFMEA and SFTA are not a “silver bullet”, but they can be easily used for software, with certain adaptation to consider software specific properties when using them at the different software development stages.

Integration within the development process. SFMEA and SFTA are respectively a bottom-up and top-down techniques which can be used for software from early development stages and both are necessary to analyse complementary aspects of the system. [10]

The use of the SoftCare method at the later stages can be the basis for accumulating historical data on the use of these methods in safety-critical systems. It can serve as support to the final system safety and dependability assessments and this is needed to provide inputs to the safety demonstration of the system.

Combination of these SFMEA and SFTA techniques, along with the necessary software specific adaptation, defines the SoftCare method, being explained in the following pages.

2.2 The SoftCare method: Introduction

The purpose of the SoftCare method is to identify and eventually remove software faults, which could lead to a software failure with severe consequences, which could in turn reduce the safety and dependability of the system, to be applied at successive software development phases.

The definition of the method is based on the idea that by systematically checking potential software faults while developing a critical software product, the potential for system hazards originated by software malfunctioning, can be drastically reduced. This technique effectively amounts to prevention of system hazards by removing software faults. From user viewpoint, any function not performed, performed untimely (i.e. unexpectedly, too early or too late) or wrongly should be consider [1].
Above figure outlines the SoftCare method, identifying its three main stages, which are introduced subsequently.

2.3 The SoftCare method: Preparatory Tasks

As stated above, execution of the SoftCare method as well as any other analysis technique requires a previous data gathering and a definition of the scope.

Data gathering. Several aspects will need to be considered during data gathering, such as the available input data, expertise required and availability of tools.

Input data. The input data for the analysis should include system and software level data: requirements, design, user manuals, code, results of preliminary criticality analysis and results of verification and validation activities. According to the scope of the analysis and its depth some of the previous data might not be relevant nor yet even existing.

Expertise required. It is important that the analysis team has knowledge in the domain of the software product or part to be analysed.

Availability of tools. The availability of tools for performing one or more analysis-related tasks (e.g. forms and tables) is useful to reduce the effort required for the execution of some systematic activities for the analysis.

Definition of the scope. The purpose of this task is to identify and prepare the items to be analysed. This task requests to:

- Identify the items to be analysed, checking any specific requirements which may imply applying the analysis only to a specific set of functions or software items, or specific requirements about the available input data defining the items to be analysed, etc.
- Define the level of depth of the analysis, whether it is to be applied only at requirements, design or at code development stages (depending on different reasons like current development life-cycle, only the functionality to be analysed, etc).
- Familiarise with the system. As the systems object of this analysis may be too complex, the process of familiarisation requires that specialist knowledge be obtained and incorporated as appropriate into the analysis results.
- Characterise the environment from which the input data for the analysis will be obtained. This characterisation will consist in the definition of various elements from the three axes of any generic development framework: development process, architecture and used technology.
2.4 The SoftCare method: Execution

The execution of the criticality analysis takes place in three sequential steps.

Software Failure Modes and Effects Analysis (SFMEA). First of all, a Software Failure Modes and Effects Analysis should be performed in order to determine the top events for lower level analysis. SFMEA analysis will be performed following the procedure presented in Figure 2.

SFMEA will be used to identify the critical functionalities based on the applicable software specification. The severity consequences of a failure, as well as the observability requirements and the effects of the failure will be used to define the criticality level of the function and thus, whether this function will be considered in further deeper criticality analysis.

The formulation of recommendations of other fault prevention or removal or tolerant techniques that may help reduce failure criticality is included as part of this analysis step. Steps depicted in Figure 2 below will be used to fill in tables with SFMEA results as depicted in the example provided in Figure 3.

![SFMEA procedure](image)

**Fig. 2. SFMEA procedure**

<table>
<thead>
<tr>
<th>ITEM no.</th>
<th>Failure Mode</th>
<th>Possible causes</th>
<th>Effects</th>
<th>Observable symptoms</th>
<th>Prevention and compensation</th>
</tr>
</thead>
</table>

Software Fault Tree Analysis (SFTA). After determining the top-level failure events, a complete Software Fault Tree Analysis shall be performed following the procedure presented below (Figure 4). SFTA is performed to analyze the faults that can cause those failures. This is a top-down technique that determines the origin of the critical
failure. The top-down technique is applied following the information provided at the design level, descending to the code modules. SFTA will be used to confirm the criticality of the functions (as output from the SFMEA) when analysing the design and code (from the software requirements phase, through design and implementation phases), and to help:
− reduce the criticality level of the functions due to software design and/or coding fault-related technique used (or recommended to be used), or
− detail the test-case definition for the set of validation test cases to be executed, or
− to really remove the fault(s) that can cause those critical failures.

For any basic event (software fault) found, which might be the cause of the corresponding failure modes identified in the SFMEA, one recommendation about how to remove or tolerate it should be provided. Samples of an SFTA tree and an SFTA table are depicted in Figures 5 and 6 respectively.

![SFTA Procedure Diagram](image-url)

**Fig. 4. SFTA Procedure**

The main added value of the SoftCare method is not just the order of application of the SFMEA and SFTA techniques, but the software failure and fault taxonomy accompanying them. [1]

The use of a *taxonomy* of the different failure modes and software fault types improves the performance of any software safety and dependability analyses, making them more systematic, complete and objective.

The nature of the system or subsystem under analysis can imply different software fault types. As a reference, the following taxonomy could be adopted:
1. hardware faults causing a subsequent software fault are physical faults with their origin in the physical device;
2. the remaining faults correspond to the human-made ones and can be subdivided in design faults (including development faults), interaction faults and intentionally malicious faults.

All potential faults from the architecture and from the lower level design and code are covered by, for example, interface faults, dynamic faults and internal faults as design level fault types, and data faults, logic faults, calculation faults and building faults as code level fault types.
Safety and dependability analysis to complement testing of safety-critical software

Fig. 5. SFTA Fault Tree Sample

<table>
<thead>
<tr>
<th>Item nr.</th>
<th>Top level event</th>
<th>SW fault</th>
<th>SW fault</th>
<th>SW Fault</th>
<th>RECOMMENDATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1.1.1</td>
<td>USR7 function is not performed</td>
<td>Task Control not started OR</td>
<td>Start entry of task Control never called OR</td>
<td>Recommendation 1: Check and unit test by which external procedures is this task entry called</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Wrong task elaboration OR</td>
<td></td>
<td>Checking the building procedures, this task is always elaborated properly</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Wrong scheduling mechanism OR</td>
<td></td>
<td>Unit test the Commercial Off The Shelf scheduler stressing scheduling mechanisms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wrong allocation of heater registers OR</td>
<td></td>
<td></td>
<td>SEE BELOW TABLE FURTHER DETAILING THIS SUB-TREE</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 6. SFTA Sample
**Evaluation of results.** The evaluation of the results is performed after the above two steps (i.e., SFMEA and SFTA) in order to highlight the potential discrepancies and prepare recommend corrective measures. In addition, recommendations can be given to design and coding rules.

An SFTA might become an enormous tree, difficult to be reduced manually. These reductions could be automatically done by a tool. SoftCare method evaluates the SFTA tree and reduces and unifies common tree branches. Only the basic software faults (basic events) potentially causing a software failure are reported together with a recommendation to avoid it or eliminate it.

### 2.5 The SoftCare method: Conclusion of analysis

**Report of findings.** The purpose of this task is to report about the activities performed for the analysis, the identified concerns and the recommendations emanating from the criticality analysis.

The final analysis report should contain the information depicted below in order to be complete, to be useful and understandable by the reader and software developer.

**Feedback from customer and supplier.** Feedback is collected from both the customer and the software developer about satisfaction on the findings and misunderstandings and corrections of the report respectively.

## 3 Experiences

### 3.1 Results provided by the method

The above method has been successfully applied in the industry at different domains: space software, automotive devices. As a consequence of this, lots of lessons have been gained, leading to a continuous improvement in its application.

**Space Domain.** Application of criticality analysis to space software domain has been relevant and profitable to a great extent.

Some results obtained from criticality analyses in a satellite onboard software are mentioned here below:

- providing NULL data from particular software components to others would cause a critical system modules collapse and, as a consequence, the whole system failure;
- inconsistent names throughout documentation could difficult the final overall integration and commandability;
- absence of log information in some modules would prevent later availability of valuable information for error recovery;
- inconsistencies among requirements, design and code are pointed to avoid lack of functionality, redundant/unnecessary code or any other fault causing from an inefficient to a catastrophic situation (an unclear use of grades and radians detected would cause a situation with unpredictable consequences);
- fulfillment of particular restrictions in the form of forbidden features in a real-time kernel for safety-critical applications became a key point to be analysed;
- the use of additional defensive programming techniques checks were necessary to prevent any other kind of errors, such as unhandled exceptions (division by zero, constraint errors, etc.) or existing dead code zones.

It should be noticed that some of the above results were pointed out at early stages of development (even at design and requirements), with the subsequent advantages from it. In the analysis of any critical applications, it can be easily observed the importance of applying a criticality analysis as a complement for testing this kind of safety-critical applications.

**Automotive devices.** Automotive sector is another good example of the growing importance of software in critical systems with a strong presence in our lives. As a result of the criticality analysis, numerous useful recommendations were defined, such as:

- suggestion to change a system safety mechanism, when some software errors were causing a system crash (and its subsequent stop);
- elimination of all unreachable/dead code;
- use of recommended practices for critical software to avoid run-time exceptions;
3.2 Conclusions

Definition and mostly application of these software failure and fault static analysis techniques (i.e. SFMEA and SFTA) has given valuable results for critical software industry where it has been applied, as a complement for traditional testing techniques.

Where applied, this method has unveiled in a short time and with an inexpensive cost many key aspects that could have led to those critical software systems to undesirable situations if they had been left uninspected.

This situation has been recognised by all participants in the analyses and, specially, by those who are responsible for the critical systems. Probably the best contribution to this method has been the confidence on it already provided by very significant customers in European critical software industry.

References

7. ECSS (European Cooperation for Space Standardisation) series of standards (http://www.ecss.nl)
11. Defence Standard 00-55 (PART 1 and 2)/Issue 2. Requirements for safety related software defence equipment. UK MoD. 1/08/97. (http://www.dstan.mod.uk/)
15. EN50128 Railway Applications: The specification and demonstration of dependability, reliability, availability, maintainability and safety (RAMS). CENELEC.
17. ECSS (European Cooperation for Space Standardisation) series of standards (www.ecss.nl)