An Overview of the Development of Safety-critical Software

Nils Forsberg
MSc Student in Intelligent Embedded Systems Program
Mälardalen University
nfg08001@student.mdh.se

Niklas Gillström
Master programme in Software Engineering 120 credits
Mälardalen University
ngm12001@student.mdh.se

ABSTRACT
Safety-critical systems are an important part of our daily life. We depend on them in many situations and if a safety-critical system fails it can result in tragic events. It is not acceptable under any circumstances that a safety-critical system malfunction with the results of human lives being lost, and for this reason the development of such systems is a sensitive process. This article is an overview of different strategies for the development process, and parts that are important to design and build a reliable safety-critical system are identified, with the focus being on the software development. In this survey paper we have collected much information taken primarily from current research, but other sources are also represented. We start by defining what a safety-critical system is and why special care should be taken during development. We then describe our findings about strategies and technologies for guaranteeing safety in these kinds of systems, primarily from a development point of view, but ways of controlling running systems online are also described. We end the survey by talking about the future of the research and development of the field.

Keywords
safety-critical, overview, development, time, validation, testing, real-time, software

1. INTRODUCTION
The definition of a critical system is a system that may cause harm to human beings or its surroundings when malfunctioning [2][6][15][16][17]. Because of this, the pressure on developing correctly working safety-critical systems is high [22], and there have over the years been created special methods for this kind of development, specifying the design process, the developing, producing, testing and even operation of these kinds of systems. Methods for guaranteeing the safety of a running system have also been devised [1] as yet another step to avoid disaster. This survey paper will take a look at the current state of the art of development of safety-critical system software and the available measures that can be taken to make sure they operate without causing harm to their surroundings. The design of safety-critical systems span many areas of engineering, the paper will be limited to safety critical software. The method will be to examine research papers and resources to get an overview of the topic and then display the collected information in the pages to come.

The paper is organized as follows: First, we will explain thoroughly what a safety critical system is, and what its different aspects are, what issues and matters should be considered that can cause or trigger a failure or an accident. We will also describe the methodology of the work for this paper. After a brief introduction to the history of the subject, we will justify the different methods of safety critical development and fault prevention. There are two main parts, what strategies exist to design safety critical systems and some techniques that exist to prevent a running system from entering a faulty state and cause accidents. We will end the paper with a look at the future of the development of safety critical systems and a summary of the work.

2. SAFETY-CRITICAL SYSTEMS
The definition of a safety-critical system is that it is a system that may cause the death of human beings or bring harm to them as the consequence of a failure of the system in question [2][6][15][16][17]. This failure may either be a breakdown where the system ceases to function or operates erroneously [16][17]. Both of these are most likely caused by incorrect code, this in turn being the result of improper programming [2]. It should be clear that a safety-critical system contains a device that operates in an environment that has the possibility of being lethal for humans, or has properties that makes itself so, and is in some way involved with a computer system, most likely controlling it or one of its attributes in some way. Other sources also takes into account the damage upon property, equipment or surrounding environment as a definition for being safety-critical. Another definition [15] (lecture 2, slide 4) is that if a system failure will lead to an aftermath of an undesired nature, then it should be imposed with safety-critical regimes and restrictions in development. Examples of safety-critical systems are [1][2][14][23]

- Navigational computers in avionics and spacecrafts
- Medical equipment such as radiation therapy machines, pace-makers and surgical equipment
- Control systems in a nuclear reactor
- Many safety systems in cars, brakes and airbags being primary examples
- Control and monitoring systems for traffic use, for example traffic lights and train signaling and switching systems
Examples of safety critical systems exists outside the world of computer systems, as safety-criticality should be considered in mechanics and electronics as well, but in this article our primary interest lies in the development of software.

2.1 Criteria

There are a number of essential criteria that need to function in order for a safety-critical system to work in an expected way [6, 9]. When developing software for safety-critical systems it is required to handle all essential criteria in a correct way in order to provide a safe solution [6, 9]. The criteria that are considered essential have to do with one of the two phases; design techniques or runtime techniques. The criteria in design techniques are; anomalies, coding guidelines, human factors, timing and verification & validation [6]. An anomaly is a criteria that address the deviations that can occur in a system. Coding guidelines is a criteria that express the recommended way to develop a safety-critical system. Human factors are a criteria that address the importance of mistakes made by humans. Timing is a criteria that address the importance of accuracy in safety-critical systems that involve timing aspects. Verification & validation is a criteria that address the importance of verifying and validating software before they are implemented in live safety-critical systems. Runtime techniques are addressing events that can occur when using a safety-critical system. If those criteria are not met then it can result in death of people or a catastrophe; such incidents can be avoided [6][9].

2.2 Methodology

For this survey paper we have chosen the following method of conduct: We will research the subject of software development for safety-critical systems and collect a number of works, (i) research papers, (ii) survey articles, (iii) lectures and (iv) journalistic articles concerning the subject. We will then gather the information and categorized and divided it according to their content and affliction within the overall subject. This selection was made based upon the grade of relatedness of the content to the subject at hand, and also the most general ones were primarily considered. The sections ahead are arranged after the categories we determined and also describes the respective content we found. We have chosen this method based on the definition of what a survey article is [26][27][28][29], that is a collection or overview of a particular field or topic; its content is not original or previously unpublished and the aim is to provide a quick and light insight into a subject without having to examine every single document. We feel that our procedures of researching and writing matches this description.

3. DESIGNING SAFETY-CRITICAL SYSTEMS

As we have said above safety-critical systems are so delicate in their operation, designing one has to be done with the outmost care; a mistake in its construction would then have catastrophic consequences. For that reason there needs to be certain strategies in the development process of these systems in order to avoid such mistakes. Over the years several such strategies have been developed, and many standards have been written specifying the steps and measures to be taken when dealing with a system of this classification. In the following subsections we will take a look at the current state of the art of development of safety-critical systems, both in past history and on the research that is being done to improve it.

3.1 IEC 61508 STANDARD

According to [1], a summary of the IEC 61508 Standard, up until the 1980s there was still many prohibitions on using programmable software code in machinery or devices which meant some sort of safety risk. It was for this reason that the International Electrotechnical Committee's Advisory Committee Of Safety (IEC ACOS) started the project of putting together a standard for such systems. It was finalized in 2000 and has been used frequently since [24], a standard that would later be viewed as a great step forward for the industry at large[1][24]. Further according to the easily accessible overview [1], this standard is characterized by requiring a great deal of documentation and the usage of statistical analysis coupled with a scrutinizing eye for details. Its purpose was as stated mainly to provide guidelines regarding systems associated with dangerous conditions, but it was also meant to be used as the base for other organizations to create their own specialized version of the standard. Many such versions have indeed been compiled, such as for example IEC61513 for nuclear industries. The overview in [1] illustrates three phases as part of the design process in the standard, (see Figure 1) which is used to reach safety in a system: analysis, where the requirements are identified; realization, which is essentially the implementation; and operation where the system is used (until it is deemed too worn out to continue to operate). These steps does of course consist of many smaller parts which further defines the process, but we will not describe them in detail here, due to the complexity of the illustration, but a short walkthrough will be made. In Figure 1 the start is at the analysis phase, which is primarily performed by the system’s end user and the projects consultant, consists of the creation of a concept, followed by specifying an overall scope definition. An analysis of any risks and hazards are made and compiled into an overall collection of safety requirements. The requirements are placed into correct context in an allocation step before the process moves on into the realization phase (Figure 1), carried out by primarily the contractor, but also the end user is involved. This phase has simultaneous steps, where one specific group concerns planning the operation and maintenance, the validation process and how the system should delivered, pertaining installation as well as commission. This group of processes will affect many steps later on in the project. Other steps in the realization phase that are based directly on the analysis are primarily concerned with realization of the project product, and are primarily about implementation and realization of the different and all aspects of the system, such as identified in the analysis. Next, the system is installed, and its safety is validated, after which the project is moved into the final operation phase (figure 1) which, fairly straightforward consists of operating the system and, in case of repairable fault, modified and/or repaired, ending the phase with a decommission of a broken down system.
There are also defined a number of levels (see table 1) for classifying the reduction of risks; four levels specify the reduction factor and the failure probability [1][25], where for example in level SIL 4 the factor 100000 means that the risk for a breakdown has been reduced by ten thousand times, and the probability of failure means the chance of a failure occurring where 1 means it will occur, and 0 means it is impossible that it will occur. Note that table 1 specifies levels for a non-continuous system, which is a system that is turned on and off as supposed to being in a permanent state of execution. The levels and values for such a system is similar to those of table 1, and so we do not specify them.

**Table 1. The Safety integrity levels, as specified in [1][25]**

<table>
<thead>
<tr>
<th>Safety Integrity Level</th>
<th>Probability of failure on demand, average (Low Demand mode of operation)</th>
<th>Risk Reduction Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL 4</td>
<td>&gt;=10^-5 to &lt;10^-4</td>
<td>100000 to 100000</td>
</tr>
<tr>
<td>SIL 3</td>
<td>&gt;=10^-4 to &lt;10^-3</td>
<td>10000 to 1000</td>
</tr>
<tr>
<td>SIL 2</td>
<td>&gt;=10^-3 to &lt;10^-2</td>
<td>1000 to 100</td>
</tr>
<tr>
<td>SIL 1</td>
<td>&gt;=10^-2 to &lt;10^-1</td>
<td>100 to 10</td>
</tr>
</tbody>
</table>

3.1.1 Standards for Engineers

The engineering profession released a standard in 1998, IEC61508 Functional safety of electrical/electronic/programmable electronic safety-related systems [1][3]. IEC61508 have a framework that can be used to create and use safety-critical systems in a safe way, and is used as a guide by most industries. It is up to the engineers of a project to use it efficiently. The first action in order to use this framework is to train and get a certification in principles and practices [1][3]. The second action is to have a commitment to safety at a personal level by using these principles and practices [1][3]. The third action is to create a safety culture, so the organization can create a system that has a high amount of trust [1][3]. The fourth action in the framework is to make sure that safety integrity is not jeopardized by engineers that put some economic interest first [1][3]. The fifth action is that development should get enough funding to make sure they are safety-related [1][3]. The sixth and last action is to take responsibility for safety in an active way. It is stated that if these actions are followed, then it will ensure that the safety integrity is not jeopardized [1][3].

**4. DESIGN TECHNIQUES AND STRATEGIES**

In order to construct a safety-critical system that will function in a correct way and to ensure that its operational stability can be guaranteed, care must be taken when designing it. When designing safety-critical systems some kind of standard should be followed to ensure that the targeted safety requirement specification is followed [6][7]. If done correctly it will function in the desired way during its whole life cycle. This means that a safety plan must be made, which must have a configuration management system that is able to manage changes in software in a way that satisfy the software safety requirements [6]. The software functional safety plan must also guarantee that the required integrity in software safety is achieved. The design techniques that are used should support good software engineering practices like for instance modularity and the ability to hide low level information from the high level construction process [6]. There should also be an architectural design description, which defines the software's different components, and if they exist, also the subsystems. That description must show how those components are connected and also contain fault tolerance strategies. The architectural design description must also include all interactions that will occur during the life cycle of the software and how important each interaction is [6][7]. This is usually done in a later stage of the development by running test cases of different kinds that was created in the beginning of the design and development stage. In order to meet all those requirements the developer of the safety-critical system must check for anomalies, follow coding guidelines, and make test scenarios that address errors caused by human factors. Two critical parts regarding design techniques are timing and verification & validation; timing is one of the main aspects in many safety-critical systems that would fail if timing anomalies occur [6][7]. Verification and validation are usually done during the project to ensure that the safety-critical system has no hidden flaws. The following will talk about these aspects in greater detail.

**4.1 Anomaly Analysis in Software Development**

In order to ensure stability in software used in a safety-critical system the developer must detect anomalies which is done by finding patterns that do not have the expected behavior, for example in output data [6][7]. Anomalies can exist in every part of the development process, ranging from the environment the developer uses to small parts of a program like the data in a vector of any kind. Patterns that do not have the expected behavior are referred to as anomalies. One way to detect anomalies is to define what the expected pattern should look like [6][7]. An expected pattern can in some cases be an undetected anomaly because of malicious actions, which makes it harder to identify it as an anomaly because they have been adapted to appear as normal. It can also be hard to identify patterns because most solutions that
exist only solve a specific version of the problem and a problem can have multiple formulations. Researchers have identified three categories and states that an anomaly can have, which are important aspects of an anomaly detection technique [7].

4.1.1 Point Anomalies

The simplest form of an anomaly is a small part of data called individual data instance [7] which most researchers focus on researching. If an occurrence of data appears outside the expected range of a data collection, then this is a point anomaly. This anomaly can be explained in the following way; a bank customer withdraws $150 every day from an ATM, but then one day the customer withdraws $300; this would be considered a point anomaly.

4.1.2 Contextual Anomalies

When a data instance is anomalous in some special cases then it would be considered to be a contextual anomaly. A contextual anomaly has attributes that is used to specify the conditions for that instance. The author [7] gives an example of spatial data sets, latitude and longitude, where a location is the contextual attribute. Contextual anomalies also have behavioral attributes, which are used to specify the conditions that are out of context. In a spatial data set that describes the medium heat in Sweden, the highest measured heat anywhere in Sweden is considered to be a behavioral attribute. Depending on the context the values within that context determine if it is a contextual anomaly or not, but only for that specific context.

4.1.3 Collective Anomalies

When considering large datasets, if a group of related data instances is anomalous then it is considered to be a collective anomaly [7]. The individual parts of a dataset can differ from each other but still does not have to be considered to be an anomaly; but when more instances show the same differences from the expected behavior then it becomes a collective anomaly. Differences between collective anomalies and the other anomalies are that a collective anomaly can only be found in data sets where relations between instances exist; while the other anomalies can occur with only one instance. According to the author [7] a series of actions that occur on a computer, for instance that some different protocols are used in a special order and that particular order is typical for a web-based attack from a remote machine and then downloads data from the host to the remote computer are an example on a collective anomaly.

4.2 Coding Guidelines

In the design of safety critical software systems the process of coding is, for obvious reasons, essential. There are initially many programming languages to choose from, but of due to mainly unpredictable behavior not all of them are suitable for the development of these kinds of systems, as taught in many courses on the subject. Languages that are popular for embedded systems development are Ada, Assembler and C, with its subset embedded C becoming more popular [9][15], although when it comes to the aspects of predictable, readable and maintainable coding C is generally considered to be a good choice [9]. Although a carefully devised and exact strategy is needed during development of safety-critical systems to guarantee safety, this article [9] argues that a small set of simple but vital guidelines and rules can be used to drastically improve the quality of such a project. These include the choice of hardware, where it is advised to use a processor that has a long history of being proven and tested, unless the requirements of the system in question is high enough to motivate the use of a specialized architecture. Unless such a processor is used it is always recommended to complement the existing fault prevention measures with a watchdog timer since a general architecture may have undocumented modes of operation, triggering unpredictable behavior. It could also be the case that a mistake has been made in the construction of the system that will cause some sort of exception or undesired behavior, like for example a stack overflow or a breakdown following a divide-by-zero operation. Also of note are the timing aspects which will be discussed in a section below, but we will mention here that a system should be time-triggered in order to improve predictability and that it is important to consider scheduling overhead when making a timing and WCET analysis. It is further advised that each company adopt a special coding style that suits their needs, and to save what solutions have been devised in libraries for future use. It is also important to choose a good compiler, since this greatly affects the end product, and to use its settings optimally, that is with all warning flags and messages turned on. When it comes to writing code itself [10] has collected the following rules: In order to avoid out of control-code, proper bounds should be used on loops, recursive functions should not be used at all and if possible there should not exist multiple points of return. Furthermore according to [10] there are several libraries and functions within libraries which should not be used at all, among them the malloc and free which is used for dynamic memory allocation, methods which should also be avoided. Highly related is the use of pointers which according to advice is much too complicated to guarantee safety; they should ever only be used for indexing of arrays and should be checked and monitored carefully.

Other advice [6] when it comes to software states that since it is such a vital part ("software is system design"), it should be modeled after the limitations and requirements of the safety of the system in question. During the design phase, run-time issues such as deadlocks and scheduling should also be considered and measures to avoid these should be included in the design from the start [9].

4.3 Human Factors

Most errors in systems are because of the involvement of the human factor in one way or another. If a safety-critical system malfunctions because of a hardware failure, and would have been done correctly from the beginning, then it can be possible to avoid such failure [1][2][3]. That could be avoided either by better hardware design or better software design choices. There are a number of important things that should be dealt with in order to ensure safety and prevent hazards according to the author [3]. Some methods mentioned is adopting a safety culture in a company and organizations that are in some way connected to the development of safety-critical systems and safety-critical systems in general [1][2][3]. This can be done by having a good leadership that defines the responsibility for every individual in a company. Dedication is also important and that a company regularly holds meetings for all employees to discuss the safety specifications and improvements. The motivation for an employee should be the possibility of a reward, when an employee help improve the safety performance there should be some kind of acknowledgement to encourage such behavior. If an employee is the reason for lower safety performance he should get some of punishment according to the author [3]. Other important things are to evaluate risks, not trying to save money with inadequate
safety as a guarantee, identifying all possible hazards. One important thing is to decide to which degree the adopted safety standards should be followed [1][2][3]. The implementation of this safety culture is harder when it involves a major infrastructure project, such as the development of new highways or new railways. The disaster at Bhopal [3] is mentioned as an example of what can happen when a series of bad decisions are made in a critical situation. It is discussed what went wrong and how it could have been avoided. The reason behind this tragedy was two workers with lack of competence [3].

4.4 Timing

In order to have robust safety-critical software time is an important aspect. Time can be expressed in many ways but the most important definitions are best-case execution time (BCET), average-case execution time (ACET) and worst-case execution time (WCET). In order to obtain these properties of a system an analysis must be performed. The most important of these execution times are worst-case execution time (WCET) [13], because if a system can handle the worst case, then all other cases are also taken care of and the entire system can be considered safe.

4.4.1 Worst-Case Execution Time

Worst-case execution time is the longest amount of time a specific processor takes to execute a specific software program, also referred to as the upper bound on the execution time [4]. The aim of doing this is to get a provably correct upper bound instead of having to guess. When there a provably correct bound has been found it can be guaranteed that the safety-critical system will meet its deadlines. To get worst-case execution time a specific model of the target processor are evaluated [5]. The worst-case execution time is the only one of the three different time measures that guarantees the predictability. An aspect to take to consideration is that it assumes that you run a sequential machine [11].

4.4.2 Analysis and Verification

When verifying time accuracy there exist two groups; exhaustive enumeration based and syntactic transformation based. Exhaustive enumeration techniques are based on examining a graph or structure, and are often made automatically. Syntactic transformations are based on verifying with logic deduction. The specification and requirements is expressed in a descriptive way and the verification contains successive applications of some deduction schemes according to the author [10].

4.5 Verification and Validation

A very important process, or processes in designing and constructing safety critical software is that of verification and validation [15][21]. According to standard procedures taught in classes on the subject, these processes, which could be simplified and called tests, are usually performed at or close to the end of a single software development iteration (see Figure 2). In commercial systems this step of the project lifecycle is often called quality assurance, but should not be seen as an identical process, since the testing is less rigorous [15]. In industry grade companies verification and validation is done more frequently and carefully [15].

![Figure 2. Example of simple development cycle](image)

It is recommend [15] that such testing, as is necessary to reach any conclusions in validation and verification, should be done after corresponding phases throughout the process, that is after a product design has been done it should be verified, and if it passes this process work can move on to the next step, but otherwise the design has to be examined and redone (see figure 4).

![Figure 3. Suggested System Development, Waterfall Model according to [15]](image)

First the terminology should be made clear. Verification is a control process in which it is examined whether or not the produced software meets the goals set up during requirement assessment and design. These should be detailed in specification so there can be no question as to whether or not the result is accurate [12, 15].

Validation on the other hand is the process of ascertaining whether or not the requirements used to verify the product are correct. It is a test of the testing specification [15]. In a professional project detailed specifications should be used, detailing when, how and by whom the validation should be carried out, complete with descriptions of the testing environment, tools and methods used [1, 15]. As far as testing of systems is concerned, [9] remarks that it is sometimes and for various
reasons impossible to test a system in the environment or state it is ultimately supposed to be used, and that in such cases special simulation models should be used to assess the system by checking its inputs and outputs. In short, verification and validation are methods that is employed in order to make sure that what is being developed corresponds with what is desired and that the requirements are being followed.

Bug hunting is a process present in any software development project. The method employed at the NASA software development division will in particular be discussed [14]. The software produced is used in extremely critical applications; if even a slight mistake is made will not only humans most likely die, but equipment and technology worth large sums of money will be destroyed. The group is divided into two teams, where one group does the coding and the other examines the written code looking for bugs. Motivation is said to be important for this kind of vital work, and according to the management this strategy creates a competitive spirit where the teams compete in striving for perfection. They also have an extensive documentation process, in which all information about every error or mistake ever made is stored, as well as every decision or design choice [14]. This is in line with the IEC 61508 standard’s specification about documentation, which states that detailed records should be made about any and all changes or choices in the life of a project [1].

4.6 Runtime Techniques

While a clever development strategy is the best guarantee that a safe-critical system is produced [6][9], there are other non-precautious ways of making sure a system performs safely. The following are ways of implementing a program so that validation is done at runtime:

- N-Version programming, where a number of programming teams are set to work individually using the same design specification. The resulting programs will most likely be different, and all of them will be used in the deployed system, along with a voting system that will decide which output to use[8][11][18][19].
- TMR (Triple Modular Redundancy Pattern) is similar to this technique, where several programs produces output judged by a voter. The difference is however that the programs are not different, but identical, and the voting is a way of making absolutely sure that the program produces correct output signals. The protected single channel pattern method puts validation and monitoring as a part of the system, where all output and signals is checked[8][18].

These are mostly used for control programs producing output. Another popular method, which has been mentioned previously, is the

- Watchdog timer. This is often implemented as a separate unit, which initiates a system reboot or a similar action if the main system has become unresponsive and stops sending regular signals to the watchdog [8][20].

4.7 Future Directions

Before we summarize the paper we will talk about the future of the development of safety-critical systems, according to what we have found about upcoming research and directions within the field. Future systems are believed to rely more on safety-critical systems than they do today [2][22]. Future systems are also believed to be more automated than today’s systems. It is possible that airplanes and its control system use Global Positioning System (GPS) in the future to achieve better accuracy in navigation. The article [2] states that cars have microprocessors in many of its safety-critical applications and that it is close to “Drive-by-wire” systems. Such system would make systems less targeted for human mistakes. The downside of this development is that we are forced to rely on safety-critical systems to a higher degree. The author in this article [2] believes that because we are using more advanced technologies the underlying systems will become more critical based on the quantity of used systems. Future systems like telemedicine will adapt and support real time remote services and by that it will need increased attention when developing safety-critical systems in that field [2][22].

4.7.1 Challenges

Since exhaustive testing is needed to ensure that a safety-critical system meets the requirements specification it will be difficult to develop such systems in the future because of the extended development times. Many times it will not be possible to deliver those systems when the safety cannot be guaranteed with less testing and increased effort. In recent research it is pointed out that we should be aware of the importance regarding today’s limitations in software engineering. When safety-critical systems are developed with new techniques it should be considered that this kind of engineering involves many technical fields. Now and in the future software like this will be a key component in any safety-critical system and practitioners of other technical fields that are not directly connected with the development of software might have a hard time to understand the complexity [2].

5. Summary

In this survey paper we have made an investigative overview of strategies and techniques for developing safety-critical systems, a system that if it malfunctions can cause harm to its surroundings, and we have done so primarily from a software development point of view. Because of professional interest, we the authors wanted to know what measures are taken to be able to guarantee that the finished product is free from error and how to handle a coding project from a safety aspect. So we first defined what a safety-critical system is and showed some real life examples, and then identified some aspects around the subject that would require attention and what would be good to remember during a development phase. We then went on to talk about the design of safety-critical systems, and some early steps towards defining guidelines for such projects. We have described many techniques and strategies related to development, and in our source material we have identified several aspects of this process which were also detailed in sections of their own above. We have also mentioned some real life examples of development phases, along with a few sample algorithms that is designed to run in the finished software to make sure that the system does not perform erroneously. We finished our survey with an investigation of the future of this research field, and the direction in which it is headed.

6. REFERENCES


[29] http://users.soe.ucsc.edu/~karplus/185/w99/reader/10_Survey_article.html [last accessed 2012-12-16]


[Figure 3] Chris Johnson, *Lecture 14: Validation and Verification*, Available at http://www.dcs.gla.ac.uk/~johnson/teaching/safety/powerpoint/14_VandV.pdf, slide 6, [last accessed 2012-11-02]