doi: 10.5028/jatm.2010.02037810

Marcos Alécio dos Santos Romani*
Instituto de Aeronáutica e Espaço
São José dos Campos − Brazil
marcosaleciomasr@iae.cta.br

Carlos Henrique Netto Lahoz
Instituto de Aeronáutica e Espaço
São José dos Campos − Brazil
lahozchnl@iae.cta.br

Edgar Toshiro Yano
Instituto Tecnológico de Aeronáutica
São José dos Campos − Brazil
yano@comp.ita.br

*author for correspondence

# Identifying dependability requirements for space software systems

***Abstract:*** *Computer systems are increasingly used in space, whether in launch vehicles, satellites, ground support and payload systems. Software applications used in these systems have become more complex, mainly due to the high number of features to be met, thus contributing to a greater probability of hazards related to software faults. Therefore, it is fundamental that the specification activity of requirements have a decisive role in the effort of obtaining systems with high quality and safety standards. In critical systems like the embedded software of the Brazilian Satellite Launcher, ambiguity, non-completeness, and lack of good requirements can cause serious accidents with economic, material and human losses. One way to assure quality with safety, reliability and other dependability attributes may be the use of safety analysis techniques during the initial phases of the project in order to identify the most adequate dependability requirements to minimize possible fault or failure occurrences during the subsequent phases. This paper presents a structured software dependability requirements analysis process that uses system software requirement specifications and traditional safety analysis techniques. The main goal of the process is to help to identify a set of essential software dependability requirements which can be added to the software requirement previously specified for the system. The final results are more complete, consistent, and reliable specifications.*

***Keywords:*** *dependability, software systems, requirements, space computer systems, criticality analysis.*

## INTRODUCTION

The aerospace systems, which involve critical software, are increasingly complex due to the great number of requirements to be satisfied, which contributes to a higher probability of hazards and risks in a project. Taking the reports of international space accidents as experience, most problems caused by software were related to requirements and to the misunderstanding of what it should do (Leveson, 2004). Lutz (1992), having examined 387 software errors during integration and system tests of the Voyager and Galileo spacecraft, found that most errors were caused by discrepancies between the documented requirements and the implementation of the functioning system. Another identified problem was the misunderstanding about the interface of the software with the rest of the system. All the reports of accidents are related to improper specification practices.

Regarding the Brazilian scenario, there is no official reporting of space accidents involving software problems. However, as the complexity of space computer systems increases with an equivalent raise of presence of functions implemented by software, there is an increased risk of accidents that can be caused by problems in computer system development. According to the recommendations of the Brazilian Satellite Launcher VLS-1 V03 accident investigation (DEPED, 2004), the technical commission proposes that the safety and quality issues should be improved as a necessary condition for the continuation of the project.

Problems related to requirements such as ambiguity, non-completeness and even the lack of non-functional requirements should be minimized during the development of space computer systems. Thus, a set of dependability attributes could be used as a start point to define most adequate non-functional requirements to minimize the possible fault or failure occurrences in the engineering phase of the requirements.

For this work, dependability is the property of a computer system to provide its services with confidence, and dependability attributes are the parameters by which the dependability of a system is evaluated (Barbacci et al., 1995). During the development of space computer systems, it is necessary to give relevant importance to security, safety, reliability, performance, quality and other dependability attributes. It is believed that the

Received: 17/06/10
Accepted: 01/10/10

use of these attributes helps the identification of non-functional requirements to be incorporated into the system, improving its quality assurance and helping to minimize the risk levels both in hardware and software parts.

This paper presents the dependability requirements analysis process for space software systems (called in this work as DEPROCESS) (Romani, 2007), which is based on a dependability attribute set and software safety analysis techniques, selected according to space standards such as the European Space Agency (ESA), the National Aeronautics and Space Administration (NASA), the UK Ministry of Defence (MOD), the Brazilian Space Agency (AEB), and other approaches of well known researchers in this area.

First, the DEPROCESS is presented, emphasizing the project phase where it is applied and its steps with the activities to be executed. Also in this section, safety analysis techniques and dependability attributes used in the process are mentioned. Then, a case study applied in embedded software used in a hypothetical space vehicle is presented. The idea is to show the application of the process in a functional requirement related to the vehicle inertial system, which has an important role in its mission. Finally, there are some considerations about the application of the process the software requirement that was analyzed, and conclusions with recommendations for improving the process are reported.

## THE DEPROCESS APPROACH

The DEPROCESS purpose is to identify dependability requirements at the beginning of software projects using safety analysis techniques (PHA, SFTA and SFMECA) and a dependability attributes classification (such as availability, reliability, safety, and others) specifically applied to the space area.

According to the lifecycle project phases proposed by ESA (2009a), the DEPROCESS is applied after the "system engineering related to software" and before the "software requirements" and "architecture engineering" processes. As the input, it uses the system requirements specified for software, and the output is a set of software dependability requirements which must be discussed during the Preliminary Design Review (PDR), for the analysis of their viability and effective incorporation to the software in the software requirement specification document.

The DEPROCESS is composed by four steps, whose activities are applied to each requirement as described in Fig. 1.

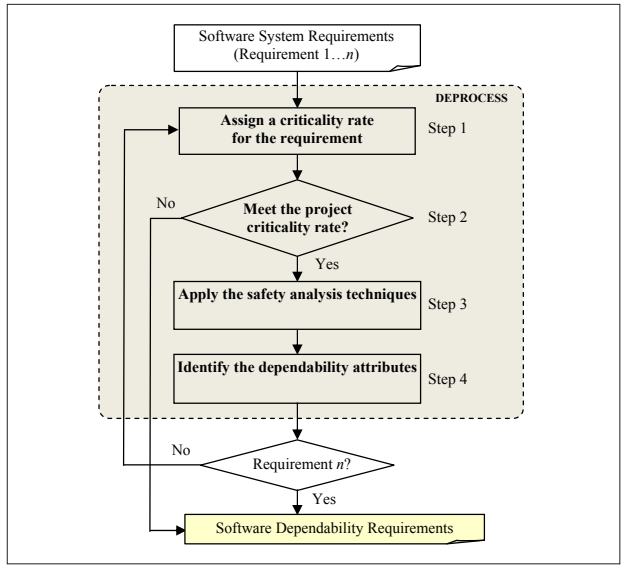


Figure 1: Dependability requirements analysis process for space software systems (DEPROCESS).

A project criticality rate must be specified for the whole project as a way to define the extension of the application of the process. This extension can vary according to the strategic conditions of the project, like the available resources, the execution schedule, and other information that should be evaluated. This case study was based on NASA criticality scale (NASA, 2005a) (Table 1).

Table 1: Criticality scale and its effects

| Criticality | Effect |
|---|---|
| 1 | Minor or negligible |
| 2 | Significant degradation |
| 3 | Subsystem loss |
| 4 | Significant loss or degradation of mission |
| 5 | Major loss or degradation of mission |
| 6 | Complete loss of mission |

The sequence of the DEPROCESS four-step execution for each studied requirement is as follows:

1. assign a criticality rate for each requirement: in this step, a criticality rate is attributed for each software system requirement, based on the results of the interviews with the project specialists, in order to compare the requirement criticality rate to the project criticality rate.

2. select if the requirement will be analyzed: in this step, it is decided if the requirement will be submitted to the application of the safety analysis techniques. It is carried out by comparing the requirement criticality rate with the project criticality rate before the start of

the application of the process. In case the requirement is not selected (requirement criticality rate < project criticality rate), it does not need a dependability analysis.

3. apply the safety analysis techniques: in this step, the requirement is submitted to the safety analysis techniques PHA, SFTA and SFMECA, considering the software interface requirements, functional requirements, performance requirements, safety requirements, and so on. As a support to this activity, keywords (NASA, 2005b) can be useful to find potential fault events and failure modes due to not meeting the requirement.

4. identify the dependability attributes: in this step, the dependability attributes are identified. They are obtained through the comparison of the results of SFTA and SFMECA techniques as to the potential system fault events/failure modes. These dependability attributes will be recommended as dependability requirements to minimize the occurrence of fault/failure related to each analyzed requirement. The dependability requirements shall be evaluated during the PDR and those considered more relevant must be incorporated into software requirement specification document.

## DEPENDABILITY EVALUATING TECHNIQUES FOR THE DEPROCESS

In the third step of DEPROCESS, the safety analysis techniques are applied to identify the potential fault events and failure modes, which will be used to help the identification of the attributes and the dependability requirements.

These techniques were selected according to two criteria:

1. comparative survey of the safety analysis techniques according to international and Brazilian standard institutions (NASA, 2005a; NBR 14857-2), shown in Table 2, and consideration of well proved techniques used in accident investigations and also their predictive analysis (DEPED, 2004; Leveson, 1995; Camargo Junior, Almeida Junior, Melnikof, 1997).

2. selection of the specific techniques for software, like SFTA and SFMECA, considering also the studies previously carried out in the software for the Brazilian space vehicles (IAE, 1994; Reis Filho, 1995). Then, the following safety analysis techniques have been chosen: Preliminary Hazard Analysis (PHA), Software Fault Tree Analysis (SFTA) and Software Failure Modes, Effects and Criticality Analysis (SFMECA).

According to NASA (2005a), PHA identifies and classifies regarding to severity that potential hazards associate to the mission due to not meeting the analyzed requirement. SFTA is a "top-down" analysis, working from hazard (top event) to possible causes (basic events), using AND and OR logic gates to connect the events; while SFMECA is a "bottom-up" analysis searching the failure modes of each function, their effects while they propagate through the system, and the hazard criticality rate at the upper level.

When used together, SFTA and SFMECA allow finding possible failure modes and areas of interest, which cannot be found by applying only one technique. This bi-directional analysis can provide limited assurances. Nevertheless, they are essential to assure that the software has been systematically examined, and that it satisfies the safety requirement for software. However, during the beginning stages of software development, like the requirement phase, only a preliminary safety analysis can be executed.

## DEPENDABILITY ATTRIBUTES IDENTIFICATION FOR THE DEPROCESS

In this work, in order to achieve an appropriate set of dependability attributes for space computer systems as a whole, all attributes related to the components that interact with the hardware, the software, or that have some kind of dependency relation were considered. As proposed by Firesmith (2006) it was defined an attribute hierarchy composed by quality factors with common concepts and related processes. These dependability attributes were classified in three groups: defensibility, soundness and quality.

These attributes are also results of researches (Romani, 2007; Lahoz, 2009), and based on Brazilian and

Table 2: Safety analysis techniques used by aerospace and defense institutions

| Techniques/ Institutions | FMEA/ FMECA | FTA | SFMECA | SFTA | HSIA | PHA | SCCFA |
|---|---|---|---|---|---|---|---|
| ESA | X | X | X | X | X | X | X |
| NASA | X | X | X | X | - | X | - |
| MOD | X | - | - | X | - | - | - |
| AEB | X | X | - | X | - | X | - |

international standard institutions (NBR 14959; MOD, 2003; ESA, 2004; NASA, 2005a), as well as studies related to the dependability of some authors in the area (Kitchenham and Pfleeger, 1996; Camargo Junior, Almeida Junior and Melnikof, 1997; Firesmith, 2003 and 2006; Rus, Komi-Sirvio and Costa, 2003; Sommerville, 2004). The definitions of dependability attributes selected in this work are presented in section Glossary, at the end of this paper.

Figure 2 shows the hierarchy created for the dependability attributes selected for space computer systems.

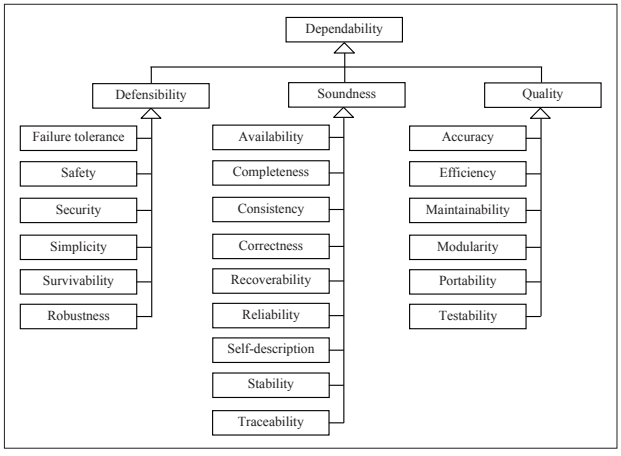


Figure 2: Attributes and Dependability hierarchy, based on Firesmith (2006).

In the "defensibility" branch, attributes are related to the way the system or its component can defend itself from accidents and attacks. In this group, failure tolerance, safety, security, simplicity, survivability and robustness attributes were included.

In the "soundness" branch, attributes are related to the way the system or its component is suitable for use. In this group, availability, completeness, consistency, correctness, recoverability, reliability, self-description, stability and traceability attributes were included.

In the "quality" branch, other attributes considered as quality factors relevant to the system or its component were classified. In this group, accuracy, efficiency, maintainability, modularity, portability and testability attributes were included.

Following, based on its definitions, the relevance of each dependability attribute selected for space computer systems is discussed.

**Accuracy**

An inaccurate value resulting from the calculation of the logic of a spacecraft control may lead to insertion of errors, accumulated during its flight, leading it to follow an unexpected trajectory, and the insertion of the satellite out of the desired orbit. An inaccurate value was one of the causes of the accident with Ariane 5 launcher in 1996 (Leveson, 2009). The precision of the navigation software in the flight control computer (on-board computer) depends on the precision of the inertial reference system measurements, but in the Ariane system test facility this precision could not be achieved by the electronics creating the test signals. The precision of the simulation may be further reduced because the base period of the inertial reference system is 1 *versus* 6 miliseconds in the simulation at the system test facility.

**Availability**

Availability may be calculated as a function of mean time to failure (MTTF) and mean time to repair (MTTR). One example cited by Fortescue, Stark and Swinerd (2003) is that for a "service" type spacecraft, such as the telephony/television communications satellite, down time or "unavailability" constitutes loss of revenue, and hence the cost benefits of design improvements to increase reliability can be optimized against their impact on revenue return. As another example, the lack of navigation data during a certain period of time of the vehicle control cycle can destabilize it, in such a way to cause the loss of the mission. Therefore, subsystems or components of the vehicle as the on-board computer, the inertial system and the data bus should be available to perform their functions in the moment they are requested.

**Completeness**

The report of the fault that caused the destruction of the Mars Polar Lander during entry and landing stage in 2000 says that the document of requirements at the system level did not specify the modes of failure related to possible transient effects to prematurely identify the touch of the ship on the ground. It is speculated that the designers of the software, or one of the auditors could have discovered the missing requirement if they were aware of its rationale (Leveson, 2004). This demonstrates that the non-consideration of the completeness attribute in the requirements may lead to occurrence of a system failure.

**Consistency**

During investigation of the American launcher Titan IV Centaur space accident, occurred in 1999, one of the causes found arose from the installation procedure of the inertial navigation system software, where the rolling rate

-0.1992476 was placed instead of -1.992476. The fault could have been identified during the pre-launch, but the consequences were not properly understood and the necessary corrections were not made because there was not a verification activity of critical data entry (Leveson, 2009).

**Correctness**

Leveson (2009) stated that in the Titan/Centaur accident, there was apparently no checking of the correctness of the software after the standard testing performed during development. For example, on the day of the launch, the attitude rates for the vehicle on the launch pad were not properly sensing the Earth's rotation rate (the software was consistently reporting a zero roll rate) but no one had the responsibility to specifically monitor that rate data or to perform a check to see if the software attitude filters were operating correctly. In fact, there were no formal processes to check the validity of the filter constants or to monitor attitude rates once the flight tape was actually loaded into the Inertial Navigation Unit at the launch site. Potential hardware failures are usually checked up to launch time, but it may have been assumed that testing removed all software errors and no further checks were needed.

**Efficiency**

Control actions will, in general, lag in their effects on the process because of delays in signal propagation around the control loop: an actuator may not immediately respond to an external command signal (called dead time); the process may have delays in responding to manipulated variables (time constants) and the sensors may obtain values only at certain sampling intervals (feedback delays). Time lags restrict the speed and extent, with which the effects of disturbances, both within the process itself and externally derived, can be reduced. They also impose extra requirements on the controller, for example, the need to infer delays that are not directly observable (Leveson, 2009). Considering a real-time software system, efficiency is a relevant attribute in the care of their temporal constraints, and is related to performance, as the checks from time response, CPU and memory usage. For example, a function that performs the acquisition and processing of inertial data to the space vehicle control system must strictly comply with their execution time, to ensure proper steering of the spacecraft during its flight.

**Failure tolerance**

There are many ways in which data processing may fail – through software and hardware, and whenever possible, spacecraft systems must be capable of tolerating failures (Pisacane, 2005). Failure tolerance is achieved primarily via hardware, but inappropriate software can compromise the system failure tolerance. During the real-time software project, it is necessary to define a strategy to meet the system required level of failure tolerance. If it is well designed, the software can detect and correct errors in an intelligent way. NASA has established levels of failure tolerance based on two levels of acceptable risk severity: catastrophic hazards must be able to tolerate two control failures and critical hazards must be able to tolerate a single control failure (NASA, 2000). Examples of software failure are the input and output errors of sensors and actuators. This failure could be tolerated by checking the data range and forcing the software to assume an acceptable value. An example of hardware failure in electronic components is the single-event upset (SEU), an annoying kind of radiation-induced failure. SEUs and their effects can be detected or corrected using some mitigation methods like error detection and correction (EDAC) codes, watchdog timers, fault rollback and watchdog processors.

**Maintainability**

It must be easy for space computer systems to maintain their subsystems, modules or components during any phase of the mission, whether on the ground or in space. The purpose of maintenance can be repair a discovered error, or allow a system upgrade to include new features of improvements. As an example, one can cite the maintenance remotely performed by NASA on Mars Exploration Rovers Spirit and Opportunity, launched toward Mars in 2003. According to Jet Propulsion Laboratory site information (JPL, 2007), the communications with the Earth is maintained through the Deep Space Network (DSN), an international network of antennas that provide communication links between the scientists and engineers on Earth and the Mars Exploration Rovers in space and on Mars. Through the DNS, it was possible to detect a problem in the first weeks of the mission that affected the Spirit rover software, causing it to remain in silence for some time, until the engineers could fix the error. The failure was related to flash memory and it was necessary a software update to fix it. It was also noted that if the Opportunity rover had landed first, it would have the same problem.

**Modularity**

The partitioning of critical systems in modules provides advantages, such as easy maintainability and traceability of the design to code, and allows the distributed software

development. Modularity contributes to the verification and validation process and errors detection during the unit, component and integration tests as well as maintenance activities. The modularity facilitates the failure isolation, preventing their spread to other modules. The independent development assists implementation and integration. As an example, a space software configuration item (ICSW) can be divided into software components (CSW), which can be divided into units or modules (USW), which correspond to the tasks to be performed during pre-flight and flight phases, in the interaction with the communication interfaces, sensors and actuators, and the transmission of data to the telemetry system.

**Portability**

The space software projects can be long-term and, during its development, there may be situations that require technological changes to improve the application, and to overcome problems such as the exchange of equipment due to the high dependence on product suppliers. For example, it is desirable that the code can be compiled into an ANSI standard in the space software systems. This will enable the code to be run on different hardware platforms and in any compatible computer system, making only specific adaptations to be transferred from one environment to another.

**Reliability**

The reliability of Space computer systems reliability depends on other factors like correct selection of components, correct derating, correct definition of the environmental stresses, restriction of vibration and thermal transfer effects from other subsystems, representative testing, proper manufacturing and so on (Fortescue, Stark and Swinerd, 2003). Reliability is calculated using failure rates, and hence the accuracy of the calculations depends on the accuracy and realism of our knowledge of failure mechanisms and modes. For most established electronic parts, failure rates are well known, but the same cannot be said for mechanical, electromechanical, and electrochemical parts or man. The author states that, in modern applications in which computers and their embedded software are often integrated into the system, the reliability of the software must also be considered. One way to define acceptable reliability levels for space systems is by regulatory authorities and, in the case of components, by the manufacture industries. An example of a space system reliability case history was cited by Pisacane (2005). The Asteroid Rendezvous (NEAR) spacecraft had a twenty-seven month development time, a four-year Cruise to the asteroid, and spent one year in orbit about the asteroid EROS. The spacecraft was successfully landed on EROS in February 2001 after one year in orbit. Reliability was maximized by limiting the number of movable and deployable mechanical and electromechanical systems.

**Recoverability**

In the autonomous embedded systems, i.e., that do not require human operators and interact with sensors and actuators, failures with severe consequences are clearly more damaging than those in which repair and recovery are simple (Sommerville, 2004). Therefore, the embedded computer systems must be able to recover themselves during the space mission situations where it is not possible to perform the maintenance. As an example, in the execution of a embedded software during the unmanned rocket flight, it is recommended that the function responsible for acquiring the data have a mechanism for recovery. In case of a failure, that does not allow the Inertial System data reading; it is necessary a recovery mechanism to provide this information to the control system so that the vehicle is not driven to a wrong trajectory.

**Robustness**

In addition to physically withstand the environment to which they will be submitted, computer systems must also be able to deal with circumstances outside the nominal values, without causing the loss of critical data that undermine the success or safety of the mission. In case of hardware failure or software errors at run time, the system critical functions should continue to be executed. As an example of software robustness assessment, NASA (2000) mention fault injection, which is a dynamic-type testing because it must be used in the context of running software following a particular input sequence and internal state profile. In fault forecasting, software fault injection is used to assess the fault tolerance robustness of a piece of software (e.g., an off-the-shelf operating system).

**Safety**

According to Fortescue, Stark and Swinerd, (2003), the overall objective of the safety program is to ensure that accidents be prevented and all hazards or threats to people, the system and the mission be identified and controlled. Safety attribute is applied to all program phases and embrace ground and flight hardware, software and documentation. They also endeavor to protect people from man-induced hazards. In the case of manned spacecraft, safety is a severe design requirement, and compliance

must be demonstrated prior to launch. Hazards can be classified as "catastrophic", "critical" or "marginal" depending on their consequences (loss of life, spacecraft loss, injury, damage etc.). Also, the most intensive and complete analysis can be carried out by constructing a safety fault tree. The software safety requirements should be derived from the system safety requirements and should not be analyzed separately (ESA, 2009a). In the software space systems, an indicator of criticality for each module defining the level of associated risk, called safety integrity level, should be specified. The most critical modules involve greater strictness in their development process (NASA, 2004a).

**Security**

Space systems have as a feature to protect information, due to the strategic interest of obtaining the technology of satellite launch vehicles, currently still dominated by few countries in the world. There should be a strict control in the access to information in these systems, because if a change occurs accidentally or maliciously, this can compromise the success of a mission. Barbacci et al. (1995) emphasizes that in government and military applications the disclosure of information was the primary risk that was to be averted at all costs. As an example of the influence of this attribute, a remote destruction command of a spacecraft launch system must be able to block another command maliciously sent from an unknown source, which seeks to prevent the vehicle from being destroyed, when it violates the flight safety plan.

**Self-description**

Re-use of technology is common in the course of space programs, that is, many systems or subsystems are reused in subsequent missions, and so require maintenance or adjustments. To minimize the possibility of introducing errors in the project, it is desirable that the computer system to be reused have a description that allows an easy understanding. For example, it is recommended that the code of a software application have comments that explain the operation of its functions, thus facilitating developers to carry out future required changes.

**Simplicity**

Simplicity is an essential aspect for the software used in critical systems, since the more complex the software, the greater the difficulty in assessing its safety (Camargo Junior, Almeida Junior and Melnikof, 1997). This is a desirable feature in a space software application because functions with simple code have expected operation and are therefore safer than others with difficulties in their understanding, which can produce indeterminate results. Software simplicity is also related to the ease of maintaining its code. For example, IV & V lessons learned from Mars Exploration Rover project (NASA, 2004b) provided evidence of the importance of this attribute. According to NASA report, portions of the file system using the system memory were very complex and modules have poor testability and maintainability. This factor contributed to a system level fault that put the Rover in a degraded communication state and allowed some unexpected commands. The file system was not the cause of the problem, but brought the lack of memory to light and created the task deadlock.

**Stability**

Space computer systems require high reliability, and their subsystems and components must continue to perform their functions within the specified operational level without causing the interruption of service provision during the mission, even if the system is operated for an extended period of time. Examples are the satellites that depend upon the performance of solar cell arrays for the production of primary power to support on-board housekeeping systems and payloads throughout their 7 to 15 years operational lifetime in orbit. The positioning systems of solar panels must have stable operation during the long-term missions, so that the satellite keeps the solar cell arrays towards the sun when going through its trajectory.

**Survivability**

The space systems are designed to operate in an environment with different features from those on Earth, such as extreme gravity, temperature, pressure, vibration, radiation, EMI variations etc. Fortescue, Stark and Swinerd (2003) noted that the different phases in the life of a space system, namely, manufacture, pre-launch, launch and finally space operation, have their own distinctive features. Although the space systems spend the majority of their lives in space, it is evident that it must survive on other environments for complete success. Critical systems should continue to provide their essential services even if they suffer accidental or malicious damage. This includes the system being able to resist to risks and threats, eliminating them or minimizing their negative effects, besides recognize accidents or attacks to allow a system reaction in case of their occurrence and recovery after the loss or degradation due to an accident or attack (Firesmith, 2003).

**Testability**

A comprehensive spacecraft test program requires the use of several different types of facilities. These are required to fulfill the system testing requirements and may include some facilities like clean room, vibration, acoustic, EMC, magnetic and RF compatibility (Pisacane, 2005). In the case of a critical software system, this feature is crucial, especially during the unit test, integration, system and acceptance and validation phases (Camargo Junior, Almeida Junior and Melnikof, 1997). The real-time software application should be tested as much as its functionality and its performance, ensuring the fulfillment of its functions during the mission within the specified time.

**Traceability**

This attribute is particularly important for computer system requirements. In a software application, the code should be linked to the requirement that originated it, thus enabling the verification through the test cases if its specified functionalities were correctly implemented. This also represents the possibility of mapping the safety requirements in all system development phases.

Based on the definitions of these factors, a table was elaborated. It generically describes the potential fault events or failure modes that can result from the application of the SFTA and SFMECA techniques and the corresponding dependability attributes recommended to minimize the occurrence of fault/failure. This table is used as a reference to execute the last DEPROCESS step, helping the analyst to identify the dependability attributes according to each fault/failure obtained. Part of this reference table is presented in Table 3.

## CASE STUDY

The chosen example for DEPROCESS application was the requirement of "process inertial information necessary to the control algorithms of the vehicle system". This requirement was extracted from the Software System Specification document (SSS) and it is related to the control system of a space vehicle. This system has an inertial system (IS) that communicates, through a data bus (DB), with the on board computer (OBC), to periodically provide the vehicle position and instantaneous acceleration data. In order to acquire the IS data and their validation to be used by control algorithms, a software function called ISDA (Inertial System Data Acquisition) should be used and executed in less than 10 miliseconds.

Table 3: Correspondence between the fault events/failure modes and the dependability attributes

| SFTA and SFMECA results | Dependability attributes |
|---|---|
| Function omits some aspect in its implementation, which leads to the occurrence of a failure in its functioning. | Completeness |
| Function contains unverified errors, which leads to the occurrence of a failure in its functioning or performance. | Consistency |
| Function does not maintain a certain performance level specified in case of software failures or violation of the specified interfaces. | Failure tolerance |
| Function operates without of its designated temporal constraints. | Performance |
| Function faults generating incorrect/unexpected results or effects. | Precision |
| Function fails in the reestablishment of its performance level and in the recovery of the data directly affected. | Recoverability |
| Function whose source code does not allow easy understanding of its functioning. | Self-description |
| Function does not continue to satisfy certain critical requirements due to adverse conditions. | Survivability |
| Function was not correctly validated. | Testability |
| Function with its general safety requirements not mapped in the specification or in its respective implementation. | Traceability |

In this case study example, the DEPROCESS was applied in the ISDA function. The lack of this function does not make possible the inertial data acquisition from the IS, not allowing the OBC to process the vehicle position and angular velocity calculations.

Applying the DEPROCESS steps, the following results were obtained:

1. assign a criticality rate for the requirement: for this case study, it was defined a criticality rate 6 (complete loss of mission). The lack of information from the IS does not allow that the data related to position be correctly processed, which can leave the vehicle out of control and/or head it into an off-nominal trajectory.

2. select if the requirement will be analyzed: as this requirement has a maximum criticality rate, the next step was automatically executed. This means that the project criticality rate did not need to be considered.

3. apply the safety analysis techniques:

   3.1. PHA – the PHA identified the potential hazard to the vehicle system, due to not meeting this requirement: "vehicle out of control during the flight". Having the classification of NASA severity categories (ref. 8) as a reference, shown in Table 4, it was classified as category I (catastrophic).

   3.2. SFTA – as shown below, the fault trees for the ISDA function are presented in Fig. 3, from the root (top event) and expanding until the leaf levels (pre-conditions to the top event occurrence).

   3.3. SFMECA – as shown below, a SFMECA built for the ISDA function is presented in Table 5, according to a model proposed by ESA (2009b).

Table 4: Hazard severity definitions according to NASA

| Hazard severity category | Definitions |
|---|---|
| I – Catastrophic | Loss of human life or permanent disability; loss of entire system; loss of ground facility; severe environmental damage. |
| II – Critical | Severe injury or temporary disability; major system or environmental damage. |
| III – Moderate | Minor injury; minor system damage. |
| IV – Negligible | No injury or minor injury; some system stress, but no system damage. |

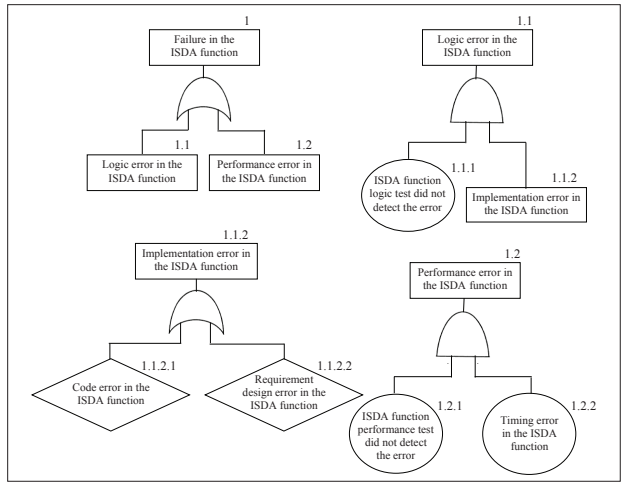


Figure 3: FTA of the ISDA function.

Table 5: SFMECA worksheet for the ISDA function

| Failure mode | Failure cause | Failure effect | Criticality | Failure detection method/ Observable symptoms | Compensation provisions |
|---|---|---|---|---|---|
| ISDA-1: no inertial data is acquired by the OBC (omission) | ISDA function not responding | No inertial data is acquired by the OBC to process the vehicle control algorithms | I | Monitoring the function status/Data not received by the OBC | Create logic recovery mechanisms for the function |
| ISDA-2: error in the inertial data (null, corrupted, spurious, or incorrect value) acquired by the OBC | Failure during the execution of the ISDA function | Incorrect results in the calculations of the inertial information processed by the OBC | I | Comparison of the previous inertial data with the current trajectory data at each instant/Trajectory data out of the specified limit | Create function logic test and create fault tolerance mechanisms for incorrect values |
| ISDA-3: ISDA function with incorrect timing | ISDA function responding after the specified time | Inertial data acquired by the OBC out of time | II | Verify the data input time in the OBC/Control actuators being activated out of the specified time | Create function performance test |

4. Identify the dependability attributes: the dependability attributes for the ISDA function were identified by comparing the basic events obtained in SFTA (step 3.2) and the failure causes obtained in SFMECA (step 3.3) with the list of potential fault events/failure modes (Table 3).

The recommended dependability requirements for this case study (Table 6) were based on the recommendations of NASA (2005b) and from some authors in the critical system area (Storey, 1996; Laplante, 2004).

The set of non-functional requirements extracted by the DEPROCESS must be discussed during the Preliminary Design Review (PDR), for the analysis of their viability and effective incorporation to the software project in the software requirement specification document.

As the SFTA and the SFMECA are bidirectional techniques, in this case study it was possible to map the possible hazards in a detailed and complementary way. The compensation provisions presented through the SFMECA provided some information that helped to define the recommended dependability requirements.

## CONCLUSIONS

It is important to point out that each project that will apply the DEPROCESS can be tailored to obtain the most effective result. For example, criticality scale, safety analysis techniques and dependability attributes set can be adjusted according to the technical features of the project. Besides, the previous knowledge about different safety techniques used by the organization should be

Table 6: Attributes and dependability requirements for the ISDA function

| **Basic event (SFTA)/ Failure causes (SFMECA)** | **Identified attributes** | **Recommended dependability requirements** |
|---|---|---|
| Function logic test did not detect error/Failure during the execution of the ISDA function | - Consistency<br>- Testability<br>- Failure tolerance | - Verify critical commands before the transmission and after the reception of the data<br>- The function should be able to consist, in each time cycle, the IS acquired values<br>- Create “black box” test cases, exercising the different possible sets of inputs and testing the limit values<br>- Create “white box” test cases to verify the coverage of the commands, branches, and decisions in the function source code<br>- The function should be able to tolerate, within a predetermined time interval, incorrect values acquired by the IS |
| Code error in the function | - Self-description<br>- Precision | - Create a complete, simple, concise, and direct documentation, and keep this information always updated<br>- Make available to the implementers a good program practice “check list” |
| Requirement design error in the function | - Completeness<br>- Traceability | - Specify the input and output data for the module and the data that are shared internally or with other modules<br>- List all possible failures inside the module or in the associated I/O devices. For each failure module, indicate how the failure can occur and how it can be detected and treated |
| Timing error in the function AND Function performance test did not detect the error / ISDA function responding out of the specified time | - Consistency<br>- Performance<br>- Testability | - Verify the function responding time, the CPU and memory use during the execution of the function<br>- Estimate function execution time counting its macroinstructions or measuring it using a logic analyzer to capture data or events |
| ISDA function not responding | - Survivability<br>- Recoverability<br>- Failure tolerance | - The function should be executed “n” times in case of failure in inertial data acquisition<br>- For extreme situations, return the program to the previous state considered safe (soft reset capacity or a watchdog timer) |

considered with DEPROCESS in order to facilitate the application of the process and the acquisition of more significant results. The set of dependability attributes can and should be discussed and adapted according to the mission or project profile.

Other relevant factor to be considered during the DEPROCESS application is the prioritization of the requirements to be analyzed. If the project criticality rate is very low, a huge set of requirements were selected, and it could lead to the impracticable DEPROCESS application.

As the DEPROCESS dependability attributes identification is a qualitative approach, its interpretation is subjective. A dependability attribute can have different meanings depending on by whom it is being evaluated, or even on its importance in the project or in the organization. For instance, diverse interpretations for the "simplicity" attribute can induce different recommendations. One view of simplicity, in computer program issues, recommended breaking up complex instructions. Another view of simplicity argues that segmented code instructions can lead to an increase of the code length, and consequently impact other quality attributes. One way to deal with this subjectiveness interpretation would be mitigate it through more than one person applying the DEPROCESS and then compare the results to find out what dependability attributes have been identified in common.

Dependability attributes can be used to help identification and analysis of dependability requirements. The use of selected dependability attributes is an effective way to guide a requirement development team to discover and refine requirements. A dependability attribute persuades an analyst to focus on a dependability issue related to a functional requirement. As result, the analyst can discover new issues and identify requirements to deal with these new demands.

In conclusion, this paper presented a structured and systematic process that addresses the dependability, focused on software systems for Brazilian space vehicles. Through pre-established criteria, such as the criticality rating scale, proper safety analysis techniques, and a set of dependability attributes, it was possible to generate some important information, such as the dependability requirements. The purpose of these recommendations is to guarantee the software functioning, and also the preliminary survey of possible vulnerable points that should be investigated in the project as whole in order to improve its quality.

## GLOSSARY

**Accuracy**
Software attributes that demonstrate the generation of results or correct effects or according to what has been agreed upon (Camargo Junior, Almeida Junior and Melnikof, 1997).

**Availability**
The ability of an item to be in a state to perform a required function under given conditions at a given instant of time or over a given time interval, assuming that the required external resources are provided (ESA, 2004).

**Completeness**
Software feature in which there is an omission on some aspect of its application which can cause the system to reach an unsafe state (Camargo Junior, Almeida Junior and Melnikof, 1997).

**Consistency**
Software feature to contain errors that are not checked, which can lead the system to an unsafe situation (Camargo Junior, Almeida Junior and Melnikof, 1997).

**Correctness**
The degree to which a work product and its outputs are free from defects since the work product is delivered (Firesmith, 2003).

**Efficiency**
It refers to timing aspects that are key factors in a critical system (Camargo Junior, Almeida Junior and Melnikof, 1997).

**Failure tolerance**
Software attributes that demonstrate its ability to maintain a specified performance level in cases of software failures or violation in the specified interfaces (Camargo Junior, Almeida Junior and Melnikof, 1997).

**Maintainability**
The ability of an item, under given conditions of use, to be retained in, or restored to, a state in which it can perform a required function, when maintenance is performed under given conditions and using stated procedures and resources (ESA, 2004).

**Modularity**
Software attributes that demonstrate the coupling degree, i.e., interdependence between its modules and low cohesion, that is, the module includes two or more independent functions (Camargo Junior, Almeida Junior and Melnikof, 1997).

**Portability**
A set of attributes that bear on the ability of software to be transferred from one environment to another, including the organizational, hardware or software environment (Kitchenham, Pfleeger, 1996).

**Reliability**
The probability with which a spacecraft will successfully complete the specified mission performance for the required mission time (Fortescue, Stark, Swinerd, 2003).
The ability of an item to perform a required function under stated conditions for a specified period of time (MOD, 2003).

**Recoverability**
Software attributes that demonstrate its ability to restore its performance level and recover the data directly affected in case of failure and the time and effort necessary for it (ABNT, 2003).

**Robustness**
The degree to which a system or component can correctly function in the presence of invalid inputs or stressful environmental conditions (Rus, Komi-Sirvio and Costa, 2003).

**Safety**
The possibility of catastrophic failure of systems in such a way as to compromise the safety of people or property, or result in mission failure (NASA, 2005a).

**Security**
Ability of the System to protect itself against accidental or deliberate intrusion (Sommerville, 2004).

**Self-description**
Software attributes that allow greater facility of its understanding and, in future maintenance, reduce the possibility of introducing new errors (Camargo Junior, Almeida Junior and Melnikof, 1997).

**Simplicity**
Critical system software feature to facilitate its safety evaluation (Camargo Junior, Almeida Junior and Melnikof, 1997).

**Stability**
The degree to which mission-critical services continue to be delivered during a given time period under a given operational profile regardless of any failures whereby the failures limiting the delivery of mission-critical services occur at unpredictable times and root causes of such failures are difficult to identify efficiently (Firesmith, 2003).

**Survivability**
The ability of a computer-communication system-based application to continue satisfying certain critical requirements (e.g., requirements for security, reliability, real-time responsiveness, and correctness) in face of adverse conditions (Rus, Komi-Sirvio, Costa, 2003).

**Testability**
Software attributes that demonstrate the effort needed to validate the modified software (NBR 14959).

**Traceability**
It represents the possibility that all the general safety requirements are perfectly mappable in the software specification and in its implementation (Camargo Junior, Almeida Junior and Melnikof, 1997).

## REFERENCES

Barbacci, M. et al., 1995, "Quality Attributes, Technical Report CMU/SEI-95-TR-021", Pittsburgh, USA: Software Engineering Institute/Carnegie Mellon University, 56 p.

Camargo Junior, J.B., Almeida Junior, J.R. and Melnikof, S.S.S., 1997, "O uso de fatores de qualidade na avaliação da segurança de software em sistemas críticos". Proceedings of Conferência internacional de tecnologia de software: qualidade de software, 8, Curitiba : CTIS, pp. 181-185.

Departamento de Pesquisas e Desenvolvimento (DEPED), Ministério da Defesa, Comando da Aeronáutica, 2004, "Relatório da investigação do acidente ocorrido com o VLS-1 V03, em 22 de agosto de 2003, em Alcântara, Maranhão", [cited November 06, 2006], Available at: http://www.iae.cta.br/VLS-1_V03_Relatorio_Final.pdf

European Space Agency (ESA), 2004, European Cooperation for Space Standardization "ECSS-P-001-B, Glossary of Terms", The Netherlands: ESA.

European Space Agency (ESA), 2009a, European Cooperation for Space Standardization "ECSS-E-ST-40C, Space Engineering – Software", The Netherlands: ESA.

European Space Agency (ESA), 2009b, European Cooperation for Space Standardization "ECSS-Q-ST-80C, Space Product Assurance – Software Product Assurance", The Netherlands: ESA.

Firesmith, D.G., 2003, "Common Concepts Underlying Safety, Security, and Survivability Engineering,

Technical Note CMU/SEI-2003- 033", Pittsburgh, USA: Software Engineering Institute/Carnegie Mellon University, 70 p.

Firesmith, D.G., 2006, "Engineering Safety-Related Requirements for Software-Intensive Systems", Proceedings of the 28th International Conference on Software Engineering, ACM SIGSOFT/IEEE, Shangai, China, pp. 1047-1048, 2006.

Fortescue, P., Stark, J. and Swinerd, G., 2003, "Spacecraft systems engineering", 3rd Ed., London: John Wiley & Sons, 678 p.

Instituto de Aeronáutica e Espaço (IAE), 1994, "Plano de Confiabilidade do Software Aplicativo de Bordo (SOAB) para o Veículo Lançador de Satélites VLS PT-01 – Preliminar – (PCS-P)".

Jet Propulsion Laboratory (JPL), 2007, "Mars Exploration Rover Mission – Communications with Earth", [cited May 15, 2009], Available at: http://marsrovers.nasa.gov/mission/communications.html

Kitchenham, B., Pfleeger, S.L., 1996, "Software Quality: the elusive target", IEEE Software, Vol. 13, N° 1, pp.12-21.

Lahoz, C.H.N., 2009, "Elicere: o processo de elicitação de metas de dependabilidade para sistemas computacionais críticos: estudo de caso aplicado a Área Espacial." PhD thesis, Universidade de São Paulo, São Paulo.

Laplante, P.A., 2004, "Real-Time Systems Design and Analysis". 3rd Ed. New York: John Wiley & Sons.

Leveson, N.G., 2009, "Engineering a safer world. System safety for the 21st century (or Systems thinking applied to safety)", Aeronautics and Astronautics Engineering Systems Division. Massachusetts Institute of Technology, [cited May 13, 2009], Available at: http://sunnyday.mit.edu/book2.pdf

Leveson, N.G., 1995, "Safeware: system safety and computers". New York: Addison-Wesley.

Leveson, N.G., 2004, "The role of software in spacecraft accidents". AIAA Journal of Spacecraft and Rockets, Vol. 41, N° 4, pp. 564-575.

Lutz, R.R., 1992, "Analyzing software requirements errors in safety-critical, embedded systems. Technical Report 92-27". Ames, Iowa, USA: Department of Computer Science, Iowa State University of Science and Technology.

NASA, 2000, "Software fault tolerance: a tutorial, technical memorandum NASA/TM-2000-210616", Hampton, USA: Langley Research Center.

NASA, 2004a, "Software Safety Guidebook, NASA-GB-8719.13", [cited October 19, 2006], Available at: http://www.hq.nasa.gov/office/codeq/doctree/871913.pdf

NASA, 2004b, "IV&V Lessons Learned – Mars Exploration Rovers and the Spirit SOL-18 Anomaly: NASA IV&V Involvement", [cited May 14, 2009], Available at: http://www.klabs.org/mapld04/presentations/session_s/2_s111_costello_s.ppt

NASA, 2005a, "Software Assurance Guidebook, NASA-GB-A201", [cited August 25, 2006], Available at: http://satc.gsfc.nasa.gov/assure/agb.txt

NASA, 2005b, "Software Fault Analysis Handbook: Software Fault Tree Analysis (SFTA) & Software Failure Modes, Effects and Criticality Analysis (SFMECA)", [cited May 07, 2007], Available at: http://sato.gsfc.nasa.gov/guidebook/assets/SQI_SFA_Handbook_05022005.doc

Pisacane, V.L., 2005, "Fundamentals of Space Systems", 2nd

Reis Filho, J.V.B., 1995, "Uma abordagem de Qualidade e Confiabilidade para Software Crítico". Masters dissertation, Instituto Tecnológico de Aeronáutica.

Romani, M.A.S., 2007, "Processo de Análise de Requisitos de Dependabilidade para Software Espacial". Masters dissertation, Instituto Tecnológico de Aeronáutica.

Rus, I., Komi-Sirvio, S., Costa, P., 2003, "Software dependability properties: a survey of definitions, measures and techniques. Technical Report 03-110. High Dependability Computing Program (HDCP)", Maryland: Fraunhofer Center for Experimental Software Engineering.

Sommerville, I. "Software Engineering", 2004, 7th Ed. Glasgow, UK: Addison-Wesley.

Storey, N., 1996, "Safety-Critical Computer Systems". Boston: Addison-Wesley Longman.

UK Ministry of Defence (MOD), 2003, "Reliability and Maintainability (R&M) – Part 7 (ARMP -7), NATO R&M Terminology Applicable to ARMP's, Defence Standard 00-40 Part 7".