DEPENDABILITY ATTRIBUTES FOR SPACE COMPUTER SYSTEMS

Romani, M.A.S., mromani@iae.cta.br
Lahoz, C.H.N., lahoz@iae.cta.br
Institute of Aeronautics and Space (IAE), São José dos Campos, São Paulo, Brazil

Yano, E.T., yano@comp.ita.br
Technological Institute of Aeronautics (ITA), São José dos Campos, São Paulo, Brazil

Abstract. *Computer systems are increasingly used in critical applications in the space area where a failure can have serious consequences such as loss of mission, properties or lives. One way to improve quality of space computer projects is through the use of dependability attributes such as reliability, availability, safety, etc. to assist the identification of non-functional requirements. This work presents and discusses a minimum set of dependability attributes, selected for software applications, to be used in space projects. These attributes resulted from a research carried out on factors extracted from practices, standards and safety models both from international institutions of the aerospace field such as ESA, NASA and MOD (UK Ministry of Defense), as well as renowned authors in the area. An example illustrates the use of dependability attributes to identify dependability requirements.*

Keywords: *dependability, requirements specification, software quality, space systems*

1. INTRODUCTION

Computer systems are increasingly used in space, whether in launch vehicles, satellites, and ground support and payload systems. Due to this fact, the 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 occurrence in a project. Pisacane (2005) emphasizes that spacecraft computer systems require space-qualified microcomputers, memories, and interface devices. Spacecraft applications impose design constraints that are much more severe than those of the commercial market. For example, a software design must attend constraints of a small memory space required to obtain lower power consumption in a spacecraft. In addition, faults and failures in a spacecraft computing system cannot usually be repaired and therefore they must exhibit high reliability levels and should be able to tolerate many kinds of faults such as the originated from physical interferences from the space environment.

Dependability (Barbacci et al., 1995) 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. During the development of space computer systems, it is necessary to give relevant importance to dependability attributes, because these systems in general must meet security, safety, reliability, performance, quality and other requirements. Failure to consider these attributes, especially in the early phases of a project can promote negative impacts in the final product. It is believed that the use of dependability attributes helps the identification of non-functional requirements to be incorporated into the project, improving its quality assurance and helping to minimize the risk levels both in the hardware and in the software parts.

According to (Lauesen, 2002), in many system projects, as some attributes are more important than others, the key point is to consider all of them and evaluate their degree of importance for each project. It is necessary to detail them until it is possible to get a measureable property. It is noteworthy that several dependability attributes may be both associated with a functional requirement as well as a non-functional requirement.

Taking the international space accidents' reports as experience, most problems caused by software were due to requirements and by not understanding what the software should do (Leveson, 2004). Lutz (1992) reports having examined 387 software errors discovered during integration and system tests of the Voyager and Galileo spacecraft and found that most errors were due to discrepancies between the documented requirements and the ones necessary for the proper functioning of the system. Another problem identified was the misunderstanding about the software interface 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 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 mistakes in computer system development.

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. A set of dependability attributes like safety, reliability, availability and others could be used as a reference to define non-functional requirements in the requirements engineering phase.

This paper aims to present and discuss a set of dependability attributes selected for space computing applications, which resulted from practical experience acquired in software projects at Institute of Aeronautics and Space (IAE), and academic research practices (Romani, 2007; Lahoz, 2009). Also, importants international institutions such as ESA, NASA and MOD, and some authors of the area (Kitchenham and Pfleeger, 1996; Camargo JR *et al.*, 1997; Firesmith, 2003, Rus et al., 2003; Sommerville, 2004) were used as reference. Although this work has been given a major focus on computer systems related to space rockets, the dependability attributes presented can also be considered in other space projects involving satellites and ground support systems. In section 2, it is reported how dependability is being treated at the IAE and the initiatives that are being taken for the analysis of dependability in the early phases of future projects. In section 3, the definitions of selected dependability attributes are presented, with examples from space computer systems. A scenario is presented, in section 4, to illustrate how dependability attributes can improve the quality of requirements for a space computer system. Finally, section 5 presents conclusions and an outline for future works.

2. DEPENDABILITY AND SOFTWARE SYSTEM REQUIREMENTS ANALYSIS PROCESS AT IAE

At IAE, several initiatives both for product development and in academic research are being taken to ensure greater confidence in the functioning of systems. The electrical system design of the Brazilian satellite launch vehicle, VLS-1, has undergone several changes to its fourth prototype, in order to improve its reliability and meet the investigation report recommendations of the accident in 2003 with the VLS - 1 V03 (DEPED, 2004), in Alcântara, Maranhão. The Technical Committee of Investigation offered a critical assessment of the necessary conditions for continuing the VLS project, especially with regard to safety and quality. Besides, the system's Preliminary Hazard Analysis (PHA) is being held, which focuses on assessing all the sequence of events regarding the vehicle's launching. A first version of this VLS-1 PHA was accomplished in 2008 and sought to identify hazards as early as possible so that appropriate actions can be implemented to reduce the probability of occurrence or severity of its consequences. The methodology used was based on the NBR 14959 standard (ABNT, 2003b) and the analysis was carried out mainly through interviews with experts related to the vehicle's electrical system and the SOAB (Onboard Software Application), to fill out forms for each event. From the PHA results, a more detailed assessment of the project with regard to safety will be performed through the implementation of Failure Modes and Effects Analysis (FMECA) to the system and also obtain an estimate of value for the VLS-1 mission's reliability. Currently, the software requirements analysis process is performed using the method of Structured Analysis with Real Time Extension (Hatley and Pirbhai, 1991), where the analysis activities are carried out aimed at the understanding, modeling (logical and behavioral) and documentation of operational, functional, performance, reliability and safety requirements. Each requirement must be unambiguously identified to enable its tracking and verification of attendance in the subsequent phases of the life cycle.

A new requirements development model is being developed in order to improve the space software requirements analysis process. This model is based on a cycle proposed by (Wiegers, 2003), as shown in Fig. 1, composed by four activities: Elicitation, Analysis, Specification and Verification & Validation (V&V).

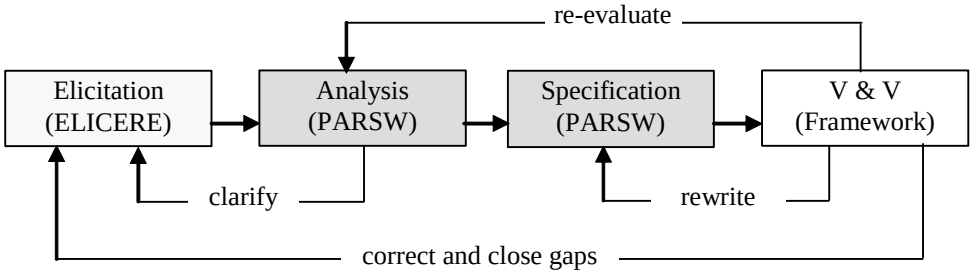

Figure 1. Requirements Development Cycle proposed to be used by IAE Space Software Projects

In the Elicitation activity, goals, needs and high level requirements are identified. In the Analysis activity requirements are detailed, negotiate to decide on which requirements are to be accepted and priorities are defined in order to enable precise effort estimations. In the Specification activity detailed requirements are registered and documented. The developed documentation must be complete and consistent. After this activity, every requirement change is controlled and the requirement can be tracked to design elements, code modules and test cases. In the V&V activity requirements have correctness verified and they are approved for implementation.

The Elicitation activity is supported by a method called ELICERE (Lahoz, 2009) using guidewords based on Hazard and Operability Study – HAZOPS (MOD, 2000) and Software Failure Modes and Effects Analysis – SFMEA (Lutz and Woodhouse, 1997) as main techniques for non-functional requirements elicitation. The ELICERE method looks for increasing confidence in the critical computer systems dependability requirements at the elicitation phase. The method identifies a number of dependability attributes needed to achieve the goals or objectives proposed for the system to be built (Lahoz and Camargo JR, 2009).

Regarding to the Analysis and Specification activities, the method PARSW (System Requirements Analysis Process for Space Software, from the Portuguese acronym) was developed (Romani, 2007) defining a structured method to

apply Preliminary Hazard Analysis – PHA (NASA, 1993), Software Fault Tree Analysis – SFTA and Software Failure Modes and Effects Analysis – SFMECA (JPL, 2005) techniques. Functional requirements are analyzed and used to refine non-functional requirements related to dependability. ELICERE and PARSW were developed to minimize problems related to non-functional requirements as early as possible.

The V&V activity proposes the use of a framework for verification and validation of critical software systems (Alves, 2008). It includes a variety of techniques that can be employed not only in the initial phase of a computer system development, but also in the subsequent activities leading to its implementation. In the proposed requirement development cycle, showed in Fig. 1, formal and traditional techniques may be used to validate the resulting requirements of the specification activity.

3. DEPENDABILITY ATTRIBUTES FOR SPACE COMPUTER SYSTEMS

In this work, to achieve an appropriate set of dependability attributes for space computer systems as a whole, all the 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), an attribute hierarchy helps to ensure that the most relevant attributes are not ignored. Thus, it is thought to be appropriate to classify the dependability attributes selected into three groups: defensibility, soundness and quality. Figure 2 shows the hierarchy created for the dependability attributes selected for space computer systems. In the "defensibility" branch are attributes related to the way the space system or its component can defend itself from accidents and attacks. In this group, the attributes failure tolerance, safety, security, simplicity, survivability and robustness were included. In the "soundness" branch are attributes related to the way the space system or its component is suitable for use. In this group, the attributes availability, completeness, consistency, correctness, recoverability, reliability, self-description, stability and traceability were included. In the "quality" branch others attributes considered as products quality's relevant to space system or its component were classified. In this group, the attributes accuracy, efficiency, maintainability, modularity, portability and testability were included.

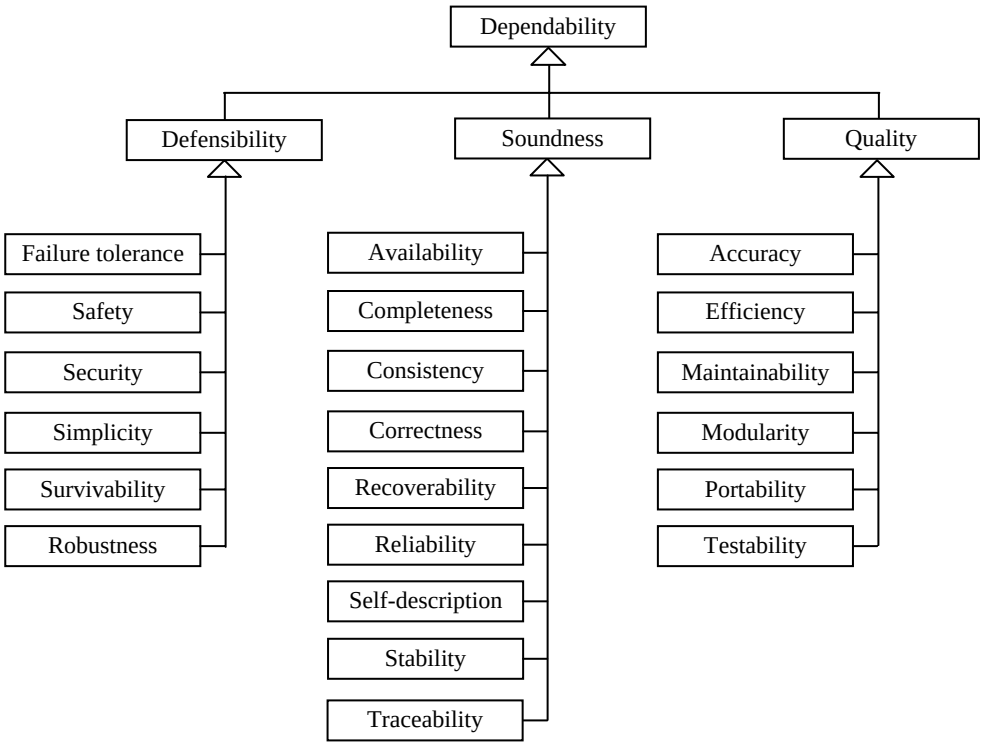

Figure 2. Attributes and Dependability Hierarchy, based on (Firesmith, 2006)

Table 1 shows the definitions of dependability attributes selected in this work for space computer systems. These attributes are results of researches (Romani, 2007; Lahoz, 2009), and also based on international and Brazilian institutions'' standards (ABNT, 2003; MOD, 2003; ESA, 2004; NASA, 2005), as well as studies related to the dependability of some authors in the area (Kitchenham and Pfleeger, 1996; Camargo JR *et al.*, 1997; Firesmith, 2003; Firesmith, 2009; Rus *et al.*, 2003; Sommerville, 2004).

Table 1. Dependability Attributes for Space Computer Systems

Dependability Attribute	Definition
Accuracy	Software's attributes that demonstrate the generation of results or correct effects or according to what has been agreed upon (Camargo JR *et al.*, 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's feature in which there is an omission on some aspect of its application which can cause the system to reach an unsafe state (Camargo JR *et al.*, 1997).
Consistency	Software's feature to contain errors that are not checked which can lead the system to an unsafe situation (Camargo JR *et al.*, 1997).
Correctness	The degree to which a work product and its outputs are free from defects once the work product is delivered (Firesmith, 2009).
Efficiency	It refers to timing aspects that are key factors in a critical system (Camargo JR *et al.*, 1997).
Failure Tolerance	Software's attributes that demonstrate its ability to maintain a specified performance level in cases of software failures or violation in the specified interfaces (Camargo JR *et al.*, 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's 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 JR *et al.*, 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 and Pfleeger, 1996).
Reliability	The probability with which a spacecraft will successfully complete the specified mission performance for the required mission time (Fortescue *et al.*, 2003). The ability of an item to perform a required function under stated conditions for a specified period of time (MOD, 2003).
Recoverability	Software's 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 function correctly in the presence of invalid inputs or stressful environmental conditions (Rus *et al.*, 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, 2005).
Security	System's ability to protect itself against accidental or deliberate intrusion (Sommerville, 2004).
Self-description	Software's attributes that allow greater facility of its understanding and in future maintenance, reduce the possibility of introducing new errors (Camargo JR *et al.*, 1997).
Simplicity	Critical system software's feature to facilitate its safety evaluation (Camargo JR *et al.*, 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, 2009).
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 the face of adverse conditions (Rus *et al.*, 2003).
Testability	Software's attributes that demonstrate the effort needed to validate the modified software (ABNT, 2003).
Traceability	It represents the possibility that all the general safety requirements are perfectly mappable in the software's specification and in its implementation (Camargo JR *et al.*, 1997).

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

3.1. Accuracy

The software application of space computer systems must be able to work with a sufficient accuracy degree to minimize errors arising from the reading of sensors and also the result of critical system routines calculations. As an example, 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. Another example of problems related to accuracy is described by (Leveson, 2009) on one of the causes of the accident with Ariane 5 launcher in 1996. 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's 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 millisecond versus 6 milliseconds in the simulation at the system test facility.

3.2. Availability

This attribute is particularly important for the hardware systems, which require continuous or almost uninterrupted operation during the mission. Availability may be calculated as a function of Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR). One example cited by (Fortescue et al., 2003) says 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 specific information, for a certain period of time of the control cycle of a space vehicle’s rolling, it can destabilize in 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.

3.3. Completeness

Completeness is an attribute often associated with requirements but rarely defined, and one of the most appropriate definitions is that software requirements specifications are complete if they are sufficient to distinguish the desired behavior of the software from that of any other undesired program that might be designed (Levenson, 2009). Space computing systems have become increasingly complex due to the high number of requirements to be met. 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.

3.4. Consistency

An example of the importance of consistency was evident in the investigation of the space accident occurred with the American launcher Titan IV Centaur 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 wasn´t a verification activity of critical data entry (Leveson, 2009).

3.5. Correctness

Correctness is another attribute not well analyzed. In (Lutz, 1992), it is reported that after having examined 387 software errors discovered during integration and system tests of the Voyager and Galileo spacecraft and it was concluded that most errors were due to discrepancies between the documented requirements specifications and the requirements necessary for the proper functioning of the system. 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.

3.6. 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 respond immediately 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.

3.7. 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 an inappropriate software project can compromise the system's failure tolerance. During the real-time software project it is necessary to define a strategy to meet the system's 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 hazard control failures and critical hazards must be able to tolerate a single hazard control failure (NASA, 2004a). An example of software failure to which space systems are subject are the errors in the input and output data of sensors and actuators. This failure could be tolerated by checking the limits of the data, forcing the software to assume a safe value. An example of hardware failure to which electronic components used in space systems are subjected 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.

3.8. Maintainability

It must be easy for the 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 that a system upgrade to include new features of improvements is made. As an example, one can cite the maintenance performed remotely by NASA on Mars Exploration Rovers Spirit and Opportunity, launched toward Mars in 2003. According to Jet Propulsion Laboratory site information (JPL, 2009), the communications with the Earth is maintained through the Deep Space Network (DSN), an international network of antennas that provide the communication links between the scientists and engineers on Earth to 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's 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 rover Opportunity had landed first, it would have had the same problem.

3.9. Modularity

The partitioning of critical systems projects in modules have advantages such as to provide a better understanding, isolation and independent development. A better understanding facilitates the process of verification and detection of errors during the individual test, integration, system, and acceptance, validation and maintenance phases. The isolation allows the containment of failures, preventing their spread to other modules. The independent development facilitates the 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 the pre-flight and flight phases, in the interaction with the communication interfaces, sensors and actuators, and the transmission of data to the telemetry system.

3.10. Portability

The space software projects can be long-term and during its development, there may be situations that require technological changes both to improve the application, and to overcome problems such as the exchange of equipment due to the high dependence on products 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.

3.11. Reliability

Space systems reliability is dependent on others factors like correct selection of components, correct derating, correct definition of the environmental stresses, restriction of vibration and thermal transfer effects from others subsystems, representative testing, proper manufacturing and so on (Fortescue *et al.*, 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. Software reliability is generally defined in terms of the indigenous faults within the software, use of the software and inputs to the software. One way to define acceptable reliability levels for space systems is by regulatory authorities and in the case of components, by the manufacturers 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.

3.12. Recoverability

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

3.13. Robustness

In addition to physically withstand the environment to which they will be submitted, space 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's critical functions should continue to be executed. As an example of software robustness assessing, NASA (2000) cites 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).

3.14. Safety

According to (Fortescue *et al.*, 2003), the overall objective of the safety program is to ensure that accidents are prevented and all hazards, or threats, to people, the system and the mission are 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 done by constructing a safety fault tree. Regarding the software, the software safety requirements should be derived from the system requirements and should not be analyzed separately (ESA, 2009). In the software space projects, an indicator of criticality for each module defining the level of associated risk, called the safety integrity level should be specified. The most critical modules involve greater strictness in their development process (NASA, 2004a).

3.15. Security

Space systems projects 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 projects, 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 teledestruction command of a spacecraft's 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.

3.16. 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 has a description that allows an easy understanding. For example, it is recommended that the code of a software application has comments that explain the operation of its functions, thus facilitating developers to carry out future changes required.

3.17. Simplicity

Simplicity is an essential aspect for the software used in critical systems, since the more complex the software is the greater the difficulty in assessing its safety (Camargo JR *et al.*, 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 and which can produce indeterminate results. Software simplicity is also related to the ease of maintaining its code. For example, a system level fault occurred with Mars Exploration Rover in 2004 put in a degraded communication state and allowed some unexpected commands (NASA, 2004b). IV & V findings related to the system memory showed that portions of the file system using the system memory was consistently reported to be very complex and modules were reported to have poor testability and poor maintainability. The file system was not the cause of the problem, but brought the lack of memory to light and created the task deadlock.

3.18. 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.

3.19. 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 and EMI variations, etc. Fortescue *et al*. (2003) noted that the different phases in the life of a space system, namely, manufacture, pre-launch, launch and finally space operation, all have their own distinctive features. Although the space systems spend the majority of their lives in space, it is evident that it must survive to the 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; recognize accidents or attacks to allow a system reaction in case of its occurrence and recovery after the loss or degradation due to an accident or attack (Firesmith, 2003).

3.20. 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 JR *et al.*, 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.

3.21. Traceability

This attribute is particularly important for the requirements of a computer system. 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 the system development phases.

4. DEPENDABILITY ATTRIBUTES APPLICATION EXAMPLE

In order to illustrate the importance of dependability attributes to analyze requirements, the following example from (Romani, 2008) is presented. In this example, the functional system software requirement “process inertial information necessary to the control algorithms of the vehicle system”, a requirement of vehicle inertial system is analyzed.

The spacecraft has an “*Inertial System*” that communicates through a “*Data Bus*” with the “*On Board Computer*” to periodically provide information of the vehicle’s position and instantaneous acceleration. In order to acquire the inertial system data and their validation to be used by the control algorithms, it should be used a software function called “*Inertial System Data Acquisition*”, which should be executed in less than 10 ms. The lack of this function does not make possible the inertial data acquisition from the “*Inertial System*” thus not allowing the computer to process the vehicle’s position and angular velocity calculations.

The dependability attributes for the “*Inertial System Data Acquisition*” function were identified using the PARSW method comparing the basic events obtained in a SFTA and the failure causes obtained in SFMECA with a reference list of potential fault events/failure modes which was based on the definitions of these factors. These attributes were the starting point for identifying the dependability requirements most adequate to minimize faults. Table 2 shows the dependability attributes identified for this function and their corresponding recommended dependability requirements after PARSW application.

This example shows that dependability attributes can be used as a guide to obtain a more complete set of dependability requirements. Without the use of dependability attributes, some dependability requirements could be not identified and analyzed.

Table 2. Attributes and Dependability Requirements for the “*Inertial System Data Acquisition*” function

Identified Attributes	Recommended Dependability Requirements
Completeness	- Specify the input and output data for the module and the data that are shared internally or with other modules.

Table 2. Attributes and Dependability Requirements for the "*Inertial System Data Acquisition*" function (continuation)

Identified Attributes	Recommended Dependability Requirements
Consistency	- Verify critical commands before the transmission and after the reception of the data. - The function should be able to consist, in each time cycle, the IS acquired values. - Verify the function responding time, the CPU and memory use during the execution.
Efficiency	- Estimate function execution time counting its macroinstructions or measure it using a logic analyzer to capture data or events.
Failure tolerance	- The function should be able to tolerate, within a pre determined time interval, incorrect values acquired by the IS. - For extreme situations, return the program to the previous state considered safe (soft reset capacity or a watchdog timer). - The function should be executed "n" times in case of failure in inertial data acquisition.
Self-description	- Create a complete, simple, concise, and direct documentation, and keep this information always updated. - Make available a good program practice "checklist" to developers.
Testability	- Create "black box" test cases, exercising the different possible sets of inputs and testing the limit values. - Create "white box" test cases to verify the coverage of the commands, branches, and decisions in the function source code.
Traceability	- 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.

5. CONCLUSIONS

The set of dependability attributes selected for space computer systems and discussed in this work originated from the authors' experience and from researches in the area of requirement engineering. Other attributes not included in this proposal can be and should be considered according to the profile of the mission or project, or even complying with the regulations and restrictions of other kinds.

Dependability attributes can be used to improve dependability requirements identification and analysis. The use of selected dependability attributes is an effective way to guide a requirements development team to discover and refine requirements. A dependability attribute persuades an analyst to focus in 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.

The ELICERE and PARSW methods, that explore dependability attributes, are currently being evaluated at IAE. For future work it is intended to incorporate these methods with development frameworks such as RUP – Rational Unified Process (Kruchten, 2000).

6. REFERENCES

Alves, M.C.B., 2008, "A Framework Proposal for Formal Verification and Validation of Space Software Systems", Proceedings of the 59th International Astronautical Congress, Glasgow, Scotland.

Associação Brasileira de Normas Técnicas (ABNT), 2003a, "Engenharia de Software – Qualidade de Produto, Parte 1: Modelo de Qualidade", NBR ISO/IEC 9126-1.

Associação Brasileira de Normas Técnicas (ABNT), 2003b, "Sistemas Espaciais – Gerenciamento de Riscos", NBR 14959.

Barbacci, M., Klein, M.H., Longstaff, T.A. and Weinstock, C.B., 1995, "Quality Attributes", Technical Report CMU/SEI-95-TR-021, Software Engineering Institute/Carnegie Mellon University, Pittsburgh, USA, 56 p.

Camargo JR, J.B., Almeida JR, J.R. and Melnikoff, S.S.S., 1997, "O Uso de Fatores de Qualidade na Avaliação da Segurança de Software em Sistemas Críticos", Proceedings of the 8th International Conference on Software Technology: Software Quality, vol.1, Curitiba, Brazil, pp. 181-195.

ESA, 2004, "Glossary of Terms", ECSS-P-001-B.

ESA, 2009, "Space Product Assurance – Software Product Assurance", ECSS-Q-ST-80C.

Firesmith, D.G., 2003, "Common Concepts Underlying Safety, Security, and Survivability Engineering", Technical Notes 033, Software Engineering Institute/Carnegie Mellon University, Pittsburgh, USA, 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.

Firesmith D.G., 2009, "Open Process Framework (OPF) Repository Organization (OPFRO) Website". http://www.opfro.org (accessed May 20, 2009).

Fortescue, P., Stark, J. and Swinerd, G., 2003, "Spacecraft Systems Engineering", 3rd Ed. John Wiley & Sons, London, UK, 678 p.

Hatley, D.J. and Pirbhai, I.A., 1991, “Estratégias para especificação de sistema em tempo real”. Ed. Makron Books, São Paulo, BR.
Jet Propulsion Laboratory (JPL), 2005, “Software Fault Analysis Handbook: Software Fault Tree Analysis (SFTA) & Software Failure Modes, Effects and Criticality Analysis (SFMECA)”. http://sato.gsfc.nasa.gov/guidebook/assets/SQI_SFA_Handbook_05022005.doc (accessed May 07, 2007).
Jet Propulsion Laboratory (JPL), 2009, “Mars Exploration Rover Mission – Communications with Earth”. http://marsrovers.nasa.gov/mission/communications.html (accessed May 15, 2009).
Kitchenham, B. and Pfleeger, S.L., 1996, “Software Quality: The Elusive Target”, Software, IEEE, vol. 13, No. 1, pp. 12-21.
Kruchten, P., 2000, “The Rational Unified Process: An Introduction”, 2nd Ed. Addison-Wesley, USA.
Lahoz, C.H.N., 2009. “ELICERE: O Processo de Elicitação de Metas de Dependabilidade para Sistemas Computacionais Críticos”, PhD tesis, Universidade de São Paulo, S. Paulo, Brazil, 225 p.
Lahoz, C.H.N. and Camargo JR, J.B., 2009, “ELICERE: A Process for Defining Dependability Goals for Critical Computer Systems”, Proceedings of the 19th Annual International Council on Systems Engineering Symposium, Singapore.
Laplante, P.A., 2004, “Real-Time Systems Design and Analysis”, 3rd Ed. John Wiley & Sons, New York, USA, 528 p.
Lauesen, S., 2002, “Software Requirements: Styles and Techniques”, Ed. Addison-Wesley, Cambridge, UK, 608 p.
Leveson, N.G., 2004, “The Role of Software in Spacecraft Accidents”, AIAA Journal of Spacecraft and Rockets, vol. 41, No. 4, pp. 564-575.
Leveson, N.G., 2009, “System Safety Engineering: Back to the Future”, Aeronautics and Astronautics Massachusetts Institute of Technology, 320 p. http://sunnyday.mit.edu/book2.pdf (accessed May 13, 2009).
Lutz, R.R., 1992, “Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems”, Technical Report 92-27, Iowa State University of Science and Technology – Department of Computer Science, Ames, USA, 19 p.
Lutz, R.R. and Woodhouse, R.M., 1997, “Requirements analysis forward and backward search”, Software Engineering, Special Volume on Requirements Engineering.
Ministério da Defesa, Comando da Aeronáutica, Departamento de Pesquisas e Desenvolvimento (DEPED), 2004, “Relatório da Investigação do Acidente Ocorrido com o VLS-1 V03, em 22 de agosto de 2003, em Alcântara, Maranhão”, São José dos Campos, Brazil. http://www.iae.cta.br/VLS-1_V03_Relatorio_Final.pdf (accessed November 06, 2006).
NASA, 1993, “Methodology for Conduct of Space Shuttle Program Hazard Analyses – Revision B”. http://pbma.nasa.gov/docs/public/pbma/bestpractices/bp_jsc_45.pdf (accessed April 05, 2007).
NASA, 1996, “Flight Assurance Procedure Number P-302-720 – Performing a Failure Modes and Effects Analysis”. http://www.goes-r.gov/procurement/ground_documents/Reference%20Documents/FMEA_Nasa_spacecraft.pdf (accessed April 19, 2007).
NASA, 2000, “Software Fault Tolerance: A Tutorial”, Technical Memorandum NASA/TM-2000-210616, Langley Research Center, Hampton, USA, 66 p.
NASA, 2004a, “Software Safety Guidebook”, NASA-GB-8719.13. http://www.hq.nasa.gov/office/codeq/doctree/871913.pdf (accessed October 19, 2006).
NASA, 2004b, “IV&V Lessons Learned – Mars Exploration Rovers and the Spirit SOL-18 Anomaly: NASA IV&V Involvement”. http://www.klabs.org/mapld04/presentations/session_s/2_s111_costello_s.ppt (accessed May 14, 2009).
NASA, 2005, “Software Assurance Guidebook”, NASA-GB-A201. http://satc.gsfc.nasa.gov/assure/agb.txt (accessed August 25, 2006).
Pisacane, V.L., 2005, “Fundamentals of Space Systems”, 2nd Ed. Oxford University Press, New York, USA, 828 p.
Romani, M.A.S., 2007, “Processo de Análise de Requisitos de Dependabilidade para Software Espacial”, Masters diss., Instituto Tecnológico de Aeronáutica, São José dos Campos, Brazil, 194 p.
Romani, M.A.S., Lahoz, C.H.N. and Yano, E.T., 2008, “Dependability Requirements Analysis Process for Space Software”, Proceedings of the 26th International System Safety Conference, vol.1, Vancouver, Canada.
Rus, I., Komi-Sirvio, S. and Costa, P., 2003, “Software Dependability Properties: A Survey of Definitions, Measures and Techniques”, Technical Report 03-110, High Dependability Computing Program (HDCP), Fraunhofer Center for Experimental Software Engineering, Maryland, USA, 45 p.
Sommerville, I., 2004, “Software Engineering”, 7th Ed. Addison-Wesley, Glasgow, UK, 784 p.
UK Ministry of Defence (MOD), 2000, “HAZOPS Studies on Systems Containing Programmable Electronics – Part 1 Requirements”, Def Stan 00-58, Issue 2.
UK Ministry of Defence (MOD), 2003, “Reliability and Maintainability (R&M) – Part 7 (ARMP -7) NATO R&M Terminology Applicable to ARMP’s”, Def Stan 00-40, Issue 1.
Wiegers, K.E., 2003, “Practical techniques for gathering and managing requirements throughout the product development cycle”, 2nd Ed. Microsoft Press, Redmond, USA, 544 p.

7. RESPONSIBILITY NOTICE

The authors are the only responsible for the material included in this paper.