Dependability Attributes for Space Computer Systems: Quality Factors Approach

Lahoz, C. H. N.[1] and Romani, M. A. S.[2]
Institute of Aeronautics and Space (IAE), Sao Jose dos Campos, Sao Paulo, 12245-75, Brazil

Yano, E. T.[3]
Institute Technological of Aeronautics (ITA), Sao Jose dos Campos, Sao Paulo, 12245-750, Brazil

Computer systems used in space applications have become more complex and critical mainly due to the high number of requirements to be defined and met. One of the important concerns about requirements is the problems related to ambiguity, non-completeness and even the lack of non-functional requirements. In space computer systems, requirements related to dependability should be identified to avoid mission fail, human life threats, and system's environment damages. This paper presents a set of dependability attributes for space computer systems, according to quality factors approaches 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 studies of well-known researchers in this area. It was defined an attribute tree with three groups of common factors and inter-related concepts. The first group of factors includes fault-tolerance, safety, security, survivability, robustness and recoverability attributes. The second group comprises the availability, reliability, stability, efficiency, accuracy and maintainability attributes. And finally, the third group includes simplicity, completeness, consistency, correctness, self-description, modularity, portability, traceability, testability and usability attributes. To obtain an appropriate set of attributes for space computer systems as a whole were considered the components that interact with the hardware, the software, or that have some kind of system dependency relationship. Based on each definition, examples, metrics and relevance of each dependability attribute are discussed and explained in the paper.

I. Introduction

Currently the use of automation software for space applications has considerably increased the mission complexity and the goals definition required to reach it. Ingham et al. (Ref. 1) states that the use of robots, for instance, in spacecrafts, despite the complexity resulting from its use, now becomes an economic necessity for continued progress in space. The authors report that, in the space systems design, there is a gap between the requirements specified by systems engineers and their translation in terms of specified requirements by software engineers. Lutz and Mikulski (Ref. 2) also added that in many critical embedded systems or any kind of human interaction critical systems, such as a spacecraft, information regarding the requirements are discovered during whole system development, and could be extended beyond the expected.

One of the main problems, according Leveson (Ref. 3), is that some systems are being developed in such a way that the management of this increasing interactive complexity makes it almost impossible. This complexity and the coupling make it difficult for the designers to consider all the potential states of the system as well as for the operators to deal with all the situations (handle all normal and abnormal situations) and disturbances safely and effectively. Besides that, Leveson (Refs. 4, 5) stated that in the space project domain, the vast majority of software

[1] Senior Technologist, Electronic Division, Praca. Mal Eduardo Gomes, 50 S. J. Campos, SP 12228-904 Brasil/ lahozchnl@iae.cta.br.
[2] Technologist, Space System Division, Praca Mal Eduardo Gomes, 50 S. J. Campos, SP 12228-904 Brasil/ marcosmasr@iae.cta.br.
3 Associated Professor, Computer Science Division, P. Mal Eduardo Gomes, 50 S. J. Campos, SP 12228-904 Brasil/ yano@ita.br.

accidents were related to flawed requirements and misunderstandings about what the software should do. The generation of software requirements for critical systems is one of the major sources of errors in system development. A manner of minimizing hazards and failures in computer space systems is to introduce, during the requirement engineering activities, a set of dependability attributes to serve as a guide for defining non-functional, system and software, requirements. Also, these attributes helps to promote a deeper interaction between system and software engineers, creating a common language for capturing, specifying and managing requirements. This paper presents a review of the dependability attributes presented at Romani (Ref. 6), Romani et al. (Ref. 7) and Lahoz (Ref. 8) to be used in space computer systems. These attributes were selected from quality factors according to space standards such as the ESA, NASA, UK-MOD, AEB, and other researchers' approaches in this area. This work includes examples of how some attributes could be better applied, as well as, where the lack of them caused failures or accidents involving space projects.

II. Dependability in Computer Systems

Spacecraft applications impose design constraints that are much more severe than those of the commercial market. For example, a spacecraft software design must attend constraints of a small memory space required to obtain lower power consumption. 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 became a concept widely applied in computer systems. The clear and simple definition arises from Sommerville (Ref. 9): dependability reflects the user's degree of trust in that system. The user's confidence that the system will operate as expected and that it will not 'fail' in normal use. In the context of system, dependability may be interpreted to mean that the system behaves how we want it to behave, in all respects that are of concern (Ref. 10). The authors emphasize that the dependability is not a single unique system attribute, but a multi-dimensional concept, with attributes of its own which must be considered if the lower level precision, which is necessary in a specific circumstance, is to be defined. 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 (Ref. 11).

Specifically for ESA (Ref. 12) dependability is a collective term used to describe the availability performance and its influencing factors as reliability, performance, maintainability performance and maintenance support performance. The ESA policy for space projects is applied by implementing an assurance program, which comprises methods and techniques to support the assessment of dependability. Also, the standards define the dependability requirements for space products, including those for system functions implemented in software, and the interaction between hardware and software (Ref. 13). In this work, the set of 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 (Ref. 14), 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, performance-related and product assurance. These attributes were reviewed from the paper presented by Romani et al. (Ref. 7), and based on international and Brazilian institutions' standards (Refs. 12, 15, 16, 17), as well as studies related to the dependability of some authors in the area (Refs. 9, 14, 18, 19, 20, 21). Figure 1 shows the hierarchy created for the dependability attributes selected for space computer systems.

In the "Defensibility" branch are attributes related to the way the system or its component can defend itself from accidents and attacks. In this group, the attributes fault tolerance, safety, security, survivability, robustness and recoverability were included. In the "Performance-related" branch are attributes related to the way the system or its component is suitable for use. In this group, the attributes availability, reliability, stability, efficiency, accuracy and maintainability were included. In the "Product assurance" branch others attributes considered relevant to system or its component were classified. In this group, the attributes simplicity, completeness, consistency, correctness, self-description, modularity, portability, traceability, testability and usability were included. Following, based on their definitions, the dependability attributes selected for space computer systems are described.

III. Defensability Attributes

Fault tolerance: Represents the ability of a functional unit to continue to perform a required function in the presence of faults or errors. 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 (Ref. 22).

Comments: Fault tolerance could be defined as the designed characteristics that maintain prescribed functions or services to users despite the existence of one or more faults. Fault tolerance is implemented by redundancy, fault detection, and response capability (Ref. 46). During the real-time software project it is necessary to define a strategy to meet the system's required level of fault tolerance. If it is well designed, the software can detect and correct errors in an intelligent way. NASA has presented two techniques applicable to software: single version and multi-version software techniques. Single version techniques focus on improving the fault tolerance of a single piece of software by adding mechanisms into the design targeting the detection, containment, and handling of errors caused by the activation of design faults. Multi-version fault tolerance techniques use multiple versions (or variants) of a piece of software in a structured way to ensure that design faults in one version do not cause system failures (Ref. 23). Examples of software fault are the input and output errors of sensors and actuators. Checking the data range and forcing the software to assume an acceptable value could tolerate this fault. An example of hardware fault 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.

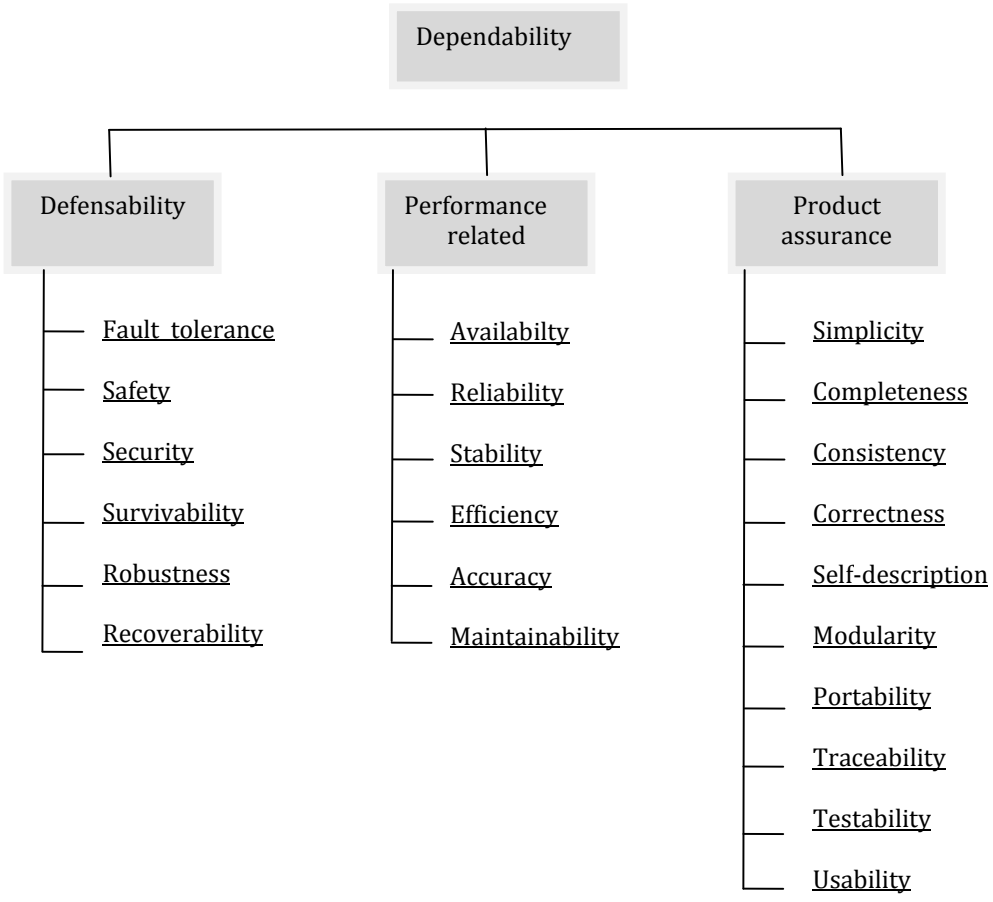

Figure 1. Dependability Attributes based on Quality Factors.

Safety: Represents the degree where a acceptable level of risk is not exceeded with respect to fatality, injury or occupational illness, damage to launcher hardware or launch site facilities, damage to an element of an interfacing manned flight system, the main functions of a flight system itself, pollution of the environment, atmosphere or outer space, and damage to public or private property is not exceeded (Ref. 12). Safety critical software is the software

that performs safety critical functions that, when failing in the performance of its function (if lost or degraded) or as a result of incorrect or inadvertent operation, can result in catastrophic or critical consequences (Ref. 24).

Comments: According to Fortescue et al. (Ref. 25), 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, constructing a safety fault tree can do the most intensive and complete analysis. Software safety is definable only in the system context. Software has no inherent dangers; however, systems controlled or monitored by software do fail, and some failures of some systems will have safety impacts. To the extent that system failures can be caused or fail to be prevented by software, there is a need for an activity called "software safety" (Ref. 26). The software safety requirements should be derived from the system safety requirements and should not be analyzed separately (Refs. 27, 28). In the software space systems, an indicator of criticality for each module defining the level of associated risk, called the safety integrity level (SIL) should be specified (Ref. 29). Another indicator, as Design Assurance Level (DAL) starts to be used in space projects where each DAL (A, B, C, D, or E in descending order of assurance) has associated with it a number of assurance objectives to be satisfied, and if any required objectives are not satisfied, that software will contribute to higher assessed system safety risk (Ref. 30).

Security: Represents the ability to protect itself against accidental or deliberate intrusion (Ref. 9). Protection from unauthorized access or uncontrolled losses or effect (Ref. 12). Representing the degree to which a system or component prevents, detects, reacts, and adapts to malicious harm to valuable assets caused by attackers (Ref. 14).

Comments: 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. (Ref. 11) 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. In the internal audit report (Ref. 31) found critical vulnerabilities on six servers connected to the Internet at the NASA. The vulnerabilities could have endangered Space Shuttle, International Space Station and Hubble Telescope missions, according to the report. These deficiencies occurred because NASA had not fully assessed and mitigated risks to its Agency-wide mission network and was slow to assign responsibility for IT security oversight to ensure the network was adequately protected. In a previous audit report, it was recommended that NASA immediately establish an IT security oversight program for this key network. It is necessary to establish strategies that specify how users are identified, its privileges and access permissions, how a system should protect itself against viruses and worms, how data corruption can be avoided, what mechanisms should be used to detect attacks on the system, how data privacy is to be maintained and how system use can be audited and checked.

Survivability: Represents the ability of a computer-communication system to continue to deliver its services to users in the face of deliberate or accidental attack (Refs. 21, 32).

Comments: 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. (Ref. 25) 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, mitigating 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 (Ref. 20).

Robustness: Represents the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions (Ref. 21). Robustness is defined as the intrinsic capability of a

system to maintain its nominal mission in the existence of various internal and external perturbations to the system throughout the mission lifetime (Ref. 33).

Comments: 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's critical functions should continue to be executed. As an example of software robustness assessment, NASA (Ref. 23) 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). The systems robustness should be specified, for example, defining the time to restart after failure.

Recoverability: Represents the ability of the system to recover from user or system errors (Ref. 32). 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 (Ref. 15).

Comments: In the autonomous embedded systems, i.e. that does not require human operators and interact with sensors and actuators failures with severe consequences are clearly more damaging than those where repair and recovery are simple (Ref. 9). 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.

IV. Performance Related Attributes

Availability: Represents the ability 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 (Ref. 12). Availability defines the proportion of time that the system is functional and working (Ref. 34).

Comments: Availability can be measured as a percentage of the total system downtime over a predefined period. Availability will be affected by system errors, infrastructure problems, malicious attacks, and system load. Availability could be specified through the Probability of Failure on Demand (POFOD), used for systems that provide intermittent rather than continuous service (Ref. 32). For example, a protection system that is intended to shut down equipment in the event of a power surge - a POFOD of 0.001 means that, on average, 1 in 1000 demands on the system will result in failure. 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. 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.

Reliability: Represents the probability with which a spacecraft will successfully complete the specified mission performance for the required mission time (Ref. 25). The ability of an item to perform a required function under stated conditions for a specified period of time (Ref. 16).

Comments: Space computer 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 (Ref. 25). 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. Reliability may be calculated as a function of Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR). One example cited by Fortescue at al. (Ref. 25) 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. One way to define acceptable reliability levels for space systems is by regulatory authorities and in the case of components, by the manufacturers industries. Leveson and Weiss (Ref. 5) argue that in software-intensive systems, component reliability is only indirectly related to safety, and increasing component reliability is not adequate to prevent losses. Most system accidents involve "correct" operation of the

individual components. 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 (Ref. 22).

Stability: Represents 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 (Ref. 14).

Comments: 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.

Efficiency: Represents the timing aspects that are key factors in a critical system (Ref. 19).

Comments: 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 (Ref. 35). Considering a real-time software system, efficiency is a relevant attribute for the care of temporal constraints, and is related to performance, as the checks for 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. As proposed by Firesmith (Ref. 36) a group of subfactors should be considered, like jitter (the maximum/minimum time intervals between specific periodic events or behaviors when in a given state), latency (the time that a application takes to execute specific tasks), response time (the time that a application takes to initially respond to a client request), schedulability (determine which events and behaviors are deterministic and can be accurately scheduled) and throughput (the number of times that a application is able to complete an operation or provide a service in a specified unit of time).

Accuracy: Represents the resource applied to demonstrate which the results are correct or according to what has been agreed upon (Ref. 19).

Comments: 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 (Ref. 35). 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 was possibly 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.

Maintainability: Represents 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 (Ref. 12).

Comments: 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 to repair a discovered error, or allow that a system upgrade to include new features of improvements can be 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 (Ref. 37), 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.

V. Product Assurance Attributes

Simplicity: Represents the feature to facilitate the dependability activities in the whole software life cycle development (Ref. 19). Simplicity is a quality that not only evokes passionate loyalty for a product design, but also has become a key strategic tool for businesses to confront their own intrinsic complexities (Ref. 38).

Comments: Good software design must be simple and easy to understand. Make easier changes in the future by keeping your code simpler now. 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 it in a degraded communication state and allowed some unexpected commands (Ref. 39). 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.

Completeness: Represents the feature to avoid any omission of some aspect of the software, which can lead the system to reach an unsafe state (Ref. 19).

Comments: The report of the fault, that caused the destruction of the Mars Polar Lander (Ref. 40) during entry and landing stage, 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 (Ref. 4). This demonstrates that the non consideration of the completeness attribute in the requirements may lead to occurrence of a system failure. According to Ref. (45), to be considered complete, the requirements document must exhibit three fundamental characteristics: (1) No information is left unstated or “to be determined”, (2) The information does not contain any undefined objects or entities, (3) No information is missing from this document.

Consistency: Represents the feature to avoid omissions or errors not checked which carries the system to an unsafe situation (Ref. 19).

Comments: An important aspect in space computer systems is the consistency between various parts of the software system being designed and implemented. During the American launcher Titan IV Centaur space accident investigation 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 wasn´t a verification activity of critical data entry (Ref. 5).

Correctness: Represents the feature which demonstrates that the software and its outputs are free from defects, once the product is delivered (Ref. 14).

Comments: In (Ref. 41), it is reported that after having examined 387 software errors discovered during integration and system tests of the Voyager and Galileo spacecraft 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 (Ref. 35) 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.

Self-description: Represents the feature that allows greater facility and understanding of the software in future maintenance, reducing the possibility of introducing new errors (Ref. 19).

Comments: Reuse 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.

Modularity: Represents the feature that demonstrates the coupling and cohesion degree between modules, i.e. the interdependence between modules and its low cohesion (Ref. 19).

Comments: The partitioning of critical systems in modules provides advantages such as easily maintainability, traceability of the design to code and allow the distributed software development. Modularity contributes to the verification and validation process and errors detection during the unit, component and integration tests as well maintenance activities. The modularity facilitates the failure isolation, preventing their spread to other modules. The independent development assists 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.

Portability: Represents the ability of software to be transferred from one environment to another, including the organizational, hardware or software environment (Ref. 18).

Comments: 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.

Traceability: Represents the capability of all functional and non-functional requirements is perfectly mappable in the software specification and in its implementation (Ref. 19).

Comments: This attribute is particularly important for computer system requirements. In a software application, the code should be mapped 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.

Testability: Represents the feature to demonstrate the effort needed to validate the modified software (Ref. 15).

Comments: 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 (Ref. 22). In the case of a critical software system, this feature is crucial, especially during the unit test, integration, system and acceptance and validation phases (Ref. 19). 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.

Usability: Represents the cost/effort to learn and handle a product (Ref. 22). It is the ease with which a user can learn to operate, prepare input for, and interpret outputs of system or component (Ref. 19).

Comments: Usability could be related to well-structured user manual, informative error messages, help facilities and well-formed graphical user interfaces. Human-computer interaction is the discipline that applies ergonomic principles to the design of user interactions with computers (Ref. 43). Ergonomics, in general, is concerned with equipment and environment. The selection of display screens, keyboards, work chair, space requirements, lighting and distracting reflections, noise, heat radiation and humidity should be considered. Specifically software ergonomics is concerned with topics like how suitable is the software for the intended operations and easy to learn. Human factors like perception, memory and human senses are included. Redmill and Rajan (Ref. 43) suggested usability metrics for specify the type of user, the task they are performing, the training they should have received,

and the conditions under which they operate. They then state the metric to be used to gauge performance (normally, how quickly the task should be achieved and with how many errors).

VI. Conclusion

Dependability attributes like safety, reliability, availability and others should be used as a reference to define non-functional requirements in the requirements engineering phase. It is necessary that all computer-related resources, on board systems, ground systems and any other support space systems, are submitted to some kind of dependability analysis or evaluation of their contribution or not to the success of the mission. All the development phases of a product should be observed under the dependability point of view, and in case any type of unsafe behavior is detected, non-functional goals should be defined in the sense of guaranteeing that the system keeps a safe behavior.

For another side, the dependability attributes are not independent. For example, availability and maintainability are closely related. If a system has to be taken out of service to be maintained, it is clearly not available. If a system is insecure, data corruption can cause the system to become unreliable (Ref. 32). The author states that there is a trade-off between the different dependability attributes and there may be conflicts between them. The security requirements, for example, may limit the number of people who have privileged access to a system but maintainability requirements may require that all maintenance staff have such access. Also, systems may often be made safe by making them unavailable (e.g. when a robot is switched off, it cannot cause any damage).

In conclusion, as some attributes are more important than others, in many system (and software) projects, the key point is to consider all of them and evaluate their degree of importance for each project (Ref. 44). 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.

Acknowledgments

This research paper and poster presentation has a financial support of National Council for Scientific and Technological Development (CNPq), Brazil (Process 559973-2010-1).

References

[1]Ingham M. D., Rasmussen R. D., Bennett M. B., and Moncada A. C., "Generating requirements for complex embedded systems using state analysis", *Proceedings of the International Astronautical Congress*, 55, Vancouver, 2004.

[2]Lutz, R. R., and Milkulski I. C. Ongoing requirements discovery in high-integrity systems. *IEEE Software*, March / April 2004, v. 21, n. 2, March 2004. pp. 19-25.

[3]Leveson, N.G. "*Safeware: System Safety and Computers*". New York: Addison-Wesley, 1995. 680 p.

[4]Leveson, N. G. "The Role of Software in Spacecraft Accidents". *AIAA Journal of Spacecraft and Rockets* 41, no. 4, July 2004, pp. 564-575.

[5]Leveson, N. G, Weiss K. A., *Safety Design for Space Systems, Software System Safety*, The International Association for the Advancement of Space Safety, Elsevier, Chapter 15, 2009, pp. 475-505.

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

[7]Romani, M. A. S., Lahoz, C. H. N., and Yano, E. T., "Dependability Attributes for Space Computer Systems", *Proceedings of. 3rd CTA-DLR 'Brazilian Symposium on Aerospace Engineering and Applications/Workshop on Data Analysis and Flight Control*" São José dos Campos, BR, 2009.

[8]Lahoz, C. H. N., "Elicere: O processo de elicitação de metas de dependabilidade para sistemas computacionais críticos: estudo de caso aplicado a área espacial." PhD dissertation, Universidade de São Paulo, São Paulo, 2009.

[9]Sommerville, I., *Software Engineering*, 7th Ed. Addison-Wesley, Glasgow, UK, 2004, 784 p.

[10]Redmill, F., Chudleigh, M. and Catmur, J., *System safety: HAZOP e software HAZOP*, Sussex: John WILEY, 1999. 248 p.

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

[12]ESA ECSS-P-001-B, European Cooperation for Space Standardization - *Glossary of Terms*, 2004.

[13]ESA ECSS-Q-ST-30C, European Cooperation for Space Standardization, *Space Product Assurance – Dependability*, 2009.

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

[15]ABNT, *Associação Brasileira de Normas Técnicas. Sistemas Espaciais – Gerenciamento de Riscos*, NBR 14959, 2003.

[16]MOD, UK Ministry of Defence, Reliability and Maintainability (R&M) – Part 7 (ARMP -7) NATO R&M , "Terminology Applicable to ARMP's", Def Stan 00-40, Issue 1, 2003.

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

[18]Kitchenham, B. and Pfleeger, S.L. "Software Quality: The Elusive Target", *IEEE Software*, 12-21, January 1996.

[19]Camargo Junior, J.B., Almeida Junior, J.R. and Melnikof, S.S.S. "O Uso de Fatores de Qualidade na Avaliação da Segurança de Software em Sistemas Críticos". Anais da Conferência Internacional de Tecnologia de Software: Qualidade de Software, (8), v.1 1997, pp. 181-195.

[20]Firesmith, D. G., "Common Concepts Underlying Safety, Security, and Survivability Engineering", Technical Notes 033, Software Engineering Institute/Carnegie Mellon University, Pittsburgh, USA, 2003 70 p.

[21]Rus, I., Komi-Sirvio, S. and Costa, P., "Software Dependability Properties: A Survey of Definitions, Measures and Techniques". High Dependability Computing Program (HDCP). Fraunhofer Center for Experimental Software Engineering, Maryland, Technical Report 03-110, 2003.

[22]Pisacane, V. L. *Fundamentals of Space Systems*, 2nd Ed. Oxford University Press, New York, USA, 2005, 828 p.

[23]NASA, "Software Fault Tolerance: A Tutorial", Technical Memorandum NASA/TM-2000-210616, Langley Research Center, Hampton, USA, 2000, 66 p.

[24]ESA ECSS-Q-80-03, European Cooperation for Space Standardization - *Space Product Assurance – Methods and Techniques to Support the Assessment of Software Dependability and Safety*, Draft, 2006.

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

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

[27]ESA ECSS-E-ST-40C, European Cooperation for Space Standardization - *Space Engineering - Software*, 2009.

[28]ESA ECSS-Q-ST-80C, European Cooperation for Space Standardization, *Space Product Assurance – Software Product Assurance*, 2009.

[29]Herrmann, D. S., *Software Safety and Reliability*, IEEE Computer Society Press, Los Alamitos, 1999. 503 p.

[30]McNeil, J., Fitzpatrick, W., Kuettner, H., Vance, R. "Use of SED DALΔ Software Risk Estimator (DΔSRE) Methodology for Estimation of Software Contribution to System Safety Risks in a Military Environment", *International System Safety Conference*, Las Vegas, 2011

[31]NASA. "Inadequate Security Practices Expose Key NASA Network to Cyber Attack", Office of Inspector General, Report No. IG-11-017 (Assignement No. A-10-011-00), March, 2011, URL: http://oig.nasa.gov/audits/reports/FY11/IG-11-017.pdf [cited 02 April 2012]

[32]Sommerville, I., Dependability Requirements, Lectures Notes, 2004

[33]Yaglioglu, B., "A Fractionated Spacecraft Architecture For Earth Observation Missions", Master dissertation, Luleå University of Technology, Sweden, 2011.

[34]Microsoft Application Architecture Guide, 2nd Edition, 2009. URL: http://msdn.microsoft.com/en-us/library/ff650706.aspx [cited 05 May 2011]

[35]Leveson, N.G. "System Safety Engineering: Back to the Future", Aeronautics and Astronautics Massachusetts Institute of Technology, 2009, 320 p. URL: http://sunnyday.mit.edu/book2.pdf [cited 13 May 2009].

[36]Firesmith, D. G., Open process framework (OPF), quality factors. URL: http://www.opfro.org/index.html?Components/WorkProducts/ModelSet/QualityModel/QualityFactors/QualityFactors.html~Contents [cited 21 May 2007].

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

[38]Maeda, J. *The laws of simplicity*, The MIT Press (August 21, 2006). 117 p.

[39]NASA, "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, 2004b. (accessed May 14, 2009).

[40]JPL, Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions, Jet Propulsion Lab -Special Review Board, 22 March 2000.

[41]Lutz, R.R. "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems". Ames: Iowa State University of Science and Technology – Department of Computer Science, Technical Report 92-27, 1992.

[42]McCall, J.A., Richards, P.K. and Walters, G.F. "Factors in software quality", Vols. I-III, Rome Air Development Centre, Italy, 1977.

[43]Redmill, F., Rajan, F., *Human Factors in Safety-Critical Systems*, Oxford: Butterworth-Heinemann, 2004. 354 p.

[44]Lauesen, S., *Software requirements, styles and techniques*. Harlow: Addison-Wesley, 2002. 591 p.

[45]Boehn, B. W. *Verifying and validating software requirements and design specifications*, IEEE Software, vol. 1, no. 1, January-March, 1984. pp. 75-88.

[46]Musgrave, G. E., Larsen, A. M., Sgobba, T. Safety Design for Space Systems, AIAA, Else Butterworth-Heinemann, Oxford, 2009. 919p.