4

A Process for Finding and Tackling the Main Root Causes that Affect Critical Systems Quality

Nuno Silva[1], Francisco Moreira[1], João Carlos Cunha[2,3] and Marco Vieira[3]

[1]CRITICAL Software S.A., Coimbra, Portugal
[2]ISEC – Coimbra Institute of Engineering, Polytechnic Institute of Coimbra, Portugal
[3]CISUC, Department of Informatics Engineering, University of Coimbra, Portugal

4.1 Introduction

Following standards and applying good engineering practices during software development is not enough to guarantee defects free software, thus additional processes, such as Independent Software Verification and Validation (ISVV), are required in critical projects. The objective of ISVV is to provide complementary and independent assessments of the software artifacts in order to find residual defects and allow their correction in a timely manner. Independence is the most important concept of ISVV and it has been referred to and used in safety-critical domains such as civil aviation (DO-178B [1]), railway signalling systems (CENELEC [2]), and space missions (European Cooperation for Space Standardization (ECSS), e.g., [3, 4]). However, such systems are still far from being perfect and it is common to hear about software bugs in aeronautics, train accidents caused by software problems, satellite systems that need to be patched after launch, and so on.

Previous studies have analysed the results of ISVV activities [5, 7], looked into consolidated ISVV metrics [8] and studied the importance of independent test verification [9], showing that existing standards and good engineering practices are not enough to guarantee the required levels of safety and dependability of critical systems (CSs). Independence of Verification and Validation (V&V) avoids author bias and is often more effective at finding defects and failures. Independence can be managerial, financial or technical, brings separation of concerns, complementarity, second/alternative opinions, and also has the merit of pushing development and in-house V&V teams to focus on the quality of their work. The role of independence at early development phases is highlighted in EasterBrook [10] and clearly stated in the requirements of several standards such as CENELEC [2] (depending on the SIL level), and DO-178 [1] (where, for example, for the most critical level – Level A – 33 out of the 71 objectives/requirements of the standard must be satisfied with full independence).

The Orthogonal Defect Classification (ODC) [11] is a generic classification technique that turns semantic information in the software defect stream into a measurement on the process where defects have been caused, enabling an efficient root cause analysis. ODC [11] can be applied to the defects identified during ISVV in order to study their classifications (namely: type, the fix that removed the defect; trigger, the defect identification activity/condition; and impact, the effect of the defect if not corrected). ODC is the most commonly used defect classification scheme, but it was not specifically developed for CSs, or for systems that need to fulfil specific certification requirements.

The application of ODC to the defects identified during ISVV has been described in Silva and Vieira [12]. In that work, we used ODC to classify a dataset of 1070 development, validation, and operation defects from space applications that followed ECSS standards. The conclusions were that most of the defect types found are related to: (i) documentation issues (this is logical since the ECSS processes are heavily based on documentation evidences); (ii) functionality issues (generally related to requirements understanding and source code bugs that compromise the foreseen functionalities); and (iii) defective implementations of the planned functions (algorithms). The classification has also shown that the main defect triggers are related to document consistency, traceability activities, and test activities. Also, the main impacts include system capability, reliability, maintainability, and documentation quality. However, the key conclusion is that a large number of issues could not be classified due to unfit taxonomy of defect types, triggers and impacts, causing many doubts in the classifications (more that 30% of the cases).

In order to enhance ODC for better applicability to CSs, thus covering all ISVV defects and easing the classification for industry, as in Silva and Vieira [13, 29], we proposed specific adaptations of the taxonomies of three classification attributes: Type, Trigger, and Impact. The enhanced classification enabled the full coverage of the defects in the dataset, providing more precision and a sounder root cause analysis support. The adaptation has been defined after conducting the classification of the 1070 defects with the original ODC (presented in Silva and Vieira [12]) and by carefully analysing the classification gaps. To validate the modifications, this enhanced version of ODC was used to reclassify the entire dataset, allowing its full classification. However, the work presented in Silva and Vieira [13] does not concretely contribute to understanding the problems that lead to the defects, which motivates the root cause analysis and the suggestions for improvements performed in the present work. The work described in this chapter represents the definition of a defects assessment process and the results of the application of this process to a space systems defects dataset.

This chapter presents an analysis on trends, common (and uncommon) problems and their causes, and look at the general picture of critical defects within the software development lifecycle of space systems, considering our dataset of 1070 defects. The results are intended to help engineers in tackling the problems starting from the most frequent ones, instead of dealing with them one by one, as is traditionally done in industry nowadays. In practice, this work brings to light the main root causes of issues in space projects, which were identified, based on the defects classification and on relevant expert knowledge about those defects and about the software development process, contributing toward proposing improvements to the processes, methodologies, tools, standards, and industry culture.

The ultimate objective of this work is to enable, through a proposed assessment process, a detailed analysis of the defects and identification of their sources (common root causes) in order to: (i) avoid their introduction (by tackling the main deficiencies in software engineering); and (ii) allow a more efficient detection of the remaining defects during the software development lifecycle (by identifying appropriate V&V methods and techniques). To support our work, the results of the enhanced ODC taxonomy proposed in Silva and Vieira [13] are used as input and analysed in detail to support the root cause analysis.

4.2 Background

This section presents some background concepts, namely in what concerns the Orthogonal Defect Classification (ODC), ISVV, and previous relevant works.

4.2.1 Orthogonal Defect Classification

The ODC, originally proposed by IBM (Chillarege et al. [11]), is one of the most used defects classification approaches. It is intended to be generic and applicable to different technology domains, but it is mostly oriented to design, code and testing defects. ODC defines eight attributes for defects classification, divided into two main groups: (i) opener, and (ii) closer. Three attributes (Activity, Trigger, and Impact) classify the defect when it has been discovered and so they are part of the opener group. The other five attributes (Target, Type, Qualifier, Age, and Source) are used when the defect is resolved, being thus part of the closer group. The full taxonomies for each attribute can be obtained from the ODC v5.2 specification and are not included here for brevity. Nevertheless, a description of ODC attributes is summarized in Table 4.1.

In addition to ODC, several other classification taxonomies exist, including Beizer's [14], and IEEE Standard Classification for Software Anomalies [15]. Although ODC comprises some questionable attributes (8 dimensions), making it also somehow complex to classify, we have selected this taxonomy due to its generic nature, its orthogonality, its comprehensiveness and the level of usage in industry that seemed higher than for all the others. Also, it is important to emphasize that ODC has been used in the past as a starting point for developing new and focused defect taxonomies

Table 4.1 Orthogonal defect classification attributes description

ODC Attribute	Description
Activity	The actual activity that was being performed at the time the defect was discovered. The main activities applicable to this work are: Requirements verification, design verification, code verification, test verification and test execution.
Trigger	A trigger represents the environment or condition that had to exist for the defect to surface.
Impact	The impact is the effect that the team who is classifying the defect thinks it would have on the system if not corrected.
Target	Represents the high level identity of the entity that was fixed.
Type	The defect type is defined according to the fix that is necessary to remove it from the system. For that reason, it is best classified by a team/person who applied the fix to the defect.
Qualifier	Captures the element of a non-existent, wrong or irrelevant implementation.
Age	Categorizes the age of the defect, whether if it is new or surfaced from a previous defect.
Source	Describes the source of the defect in terms of its developmental history.

for different domains. A few examples were presented by Leszak et al. [16] and Lopes Margarido et al. [17], which used ODC for studying, building and validating defect categorization schemes. In practice, the focus of ODC is to support the analysis and feedback of defect data targeting quality issues from different phases of the engineering lifecycle.

4.2.2 Independent Software Verification and Validation (ISVV)

Independent Software Verification and Validation is a set of structured engineering activities and tools that allow independent analysts to evaluate the quality of the software engineering artifacts produced at each phase of the development lifecycle. ISVV is performed on mature artifacts, which follow a strict engineering standard and that have been previously verified and validated as part of the development process. It provides an additional layer of confidence and is not expected to find a large number of severe defects.

Independent Software Verification and Validation produces evidences that support measuring the quality of the software and related processes and is referenced in several international standards: (i) ISVV guide from the European Space Agency (ESA) [18]; (ii) ISO Software Lifecycle Processes (ISO/IEC 12207) [19]; and (iii) IEEE Software V&V (IEEE 1012) [20].

Independent Software Verification and Validation includes six phases (Figure 4.1) that can be executed sequentially or selected/adapted as the result of a tailoring process based on a criticality analysis [18].

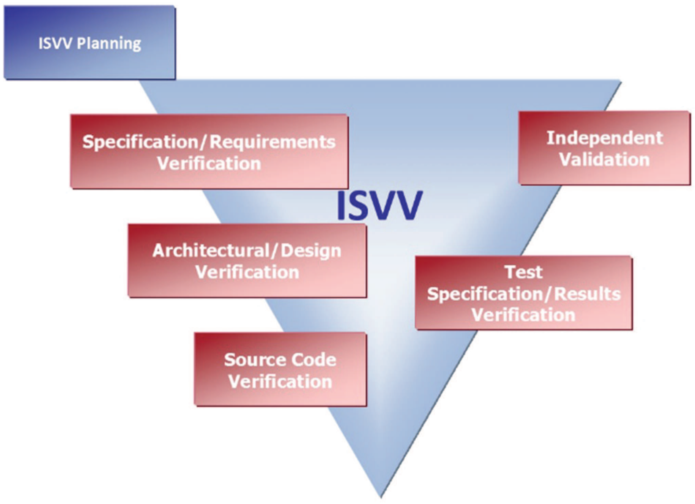
ISVV Planning
Specification/Requirements Verification
ISVV
Independent Validation
Architectural/Design Verification
Test Specification/Results Verification
Source Code Verification

Figure 4.1 ISVV phases.

According to the ESA ISVV Guide [18], ISVV engineers classify defects considering three severity levels: (i) Major (defect with a significant impact in the system dependability, quality or safety); (ii) Minor (defect with a minimum impact on the artifacts quality but not in the end system); and (iii) Comment (an improvement suggestion). Each ISVV defect is also classified according to an ISVV defect type (e.g., External Consistency, Internal Consistency, Correctness, Technical Feasibility, Completeness, Readability, and Maintainability).

4.2.3 Related Work

Some studies in the literature have analysed metrics, efficiency and efficacy of the techniques used within ISVV to identify the defects in critical projects [5–8]. However, none of these studies considered their observations and results to classify the defects and improve the development processes, techniques, tools, or standards. Furthermore, we could not find in the research literature any complete study focused on defects in mission- and safety-critical systems, nor an extensive and complete classification or root cause analysis that relates the results of ISVV with the development lifecycle parameters of the systems under study. For space systems, Jones [21] has provided a small study about space failures in the frame of the European Space Agency missions, but simply concluded that the main cause for all the accidents was lack of testing. A more in-depth analysis is necessary as testing is not the cause but one of the detection methods.

Several researchers have looked into the analysis of failures in safety-critical systems during different life-cycle phases (from requirements to operations) and performed empirical studies and root cause analysis [22–24]. For example, Seaman et al. [25] used historical datasets with defects data from reviews and inspections and applied different categorization schemes to the defects. However, none of the mentioned studies covers all the life-cycle phases for the used defects dataset, nor bases the root cause analysis and the defects avoidance measures in a sound orthogonal classification of the defects.

Regarding the root cause analysis topic, it is worth mentioning some works that relate and somehow present results that are connected to the work presented in this chapter. Neufelder [26] collects data from field defects since 1993 and correlates that data to find the process properties that generate more defects; however, she is not focusing on CSs or systems developed under strict requirements and standards. Rao [27] has made an industry study about root cause defect classification for documentation defects, analysing only a few dozen defects on a monthly basis. Kumaresh et al. [28] conducted a study with data from a few hundreds of collected defects, where these defects have been classified and the corresponding root causes have been proposed to the learning of the projects as preventive ideas. No work has been performed for CSs nor with such a complete assessment and coverage of so many defect types (as shown by the ODC defect type results), as we did in our work.

4.3 Defects Assessment Process

Based on the analysis that we conducted and the lessons learned, we propose a general approach for root cause analysis of critical software, enabling the continuous improvement of implementation and V&V at all levels (processes, techniques, tools, personnel, application of standards, organization, and so on). Although our dataset and our experience are mainly from space software, we believe that this generalization is able to support the evaluation and root cause analysis of any critical system, independently from the domain. Figure 4.2 shows the general approach of a defects assessment procedure, which includes a root-cause analysis and a continuous improvement procedure, described hereafter.

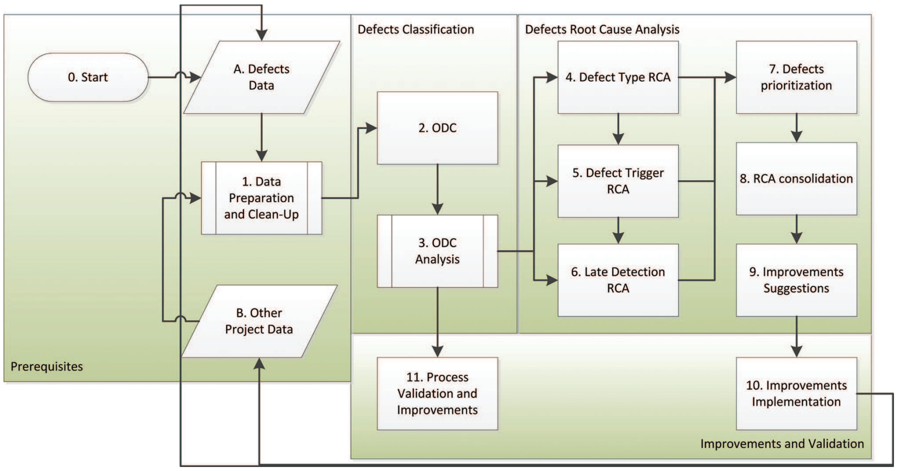
Defects Classification
Defects Root Cause Analysis
0. Start
A. Defects Data
1. Data Preparation and Clean-Up
B. Other Project Data
Prerequisites
2. ODC
3. ODC Analysis
4. Defect Type RCA
5. Defect Trigger RCA
6. Late Detection RCA
7. Defects prioritization
8. RCA consolidation
9. Improvements Suggestions
10. Improvements Implementation
11. Process Validation and Improvements
Improvements and Validation

Figure 4.2 Generalized defect assessment procedure.

4.3.1 Procedure Prerequisites

The approach is based on data analysis and software engineering knowledge that require some prerequisites to be fulfilled for the correct application of the process:

0. Start:
In order to successfully perform the defects analysis, it is necessary that the collected data (A. Defects Data and B. Other Project Data) contain the necessary information. This includes basic requirements such as: (i) detailed information about each defect and its fix; (ii) knowledge of defect environment conditions, such as tools, personnel and constraints; (iii) engineers' assessment of the defect causes; and (iv) phase when the defect was introduced and when it was detected.

Some prerequisites are necessary to successfully apply the process. The first one includes training on the involved techniques, such as defects classification (e.g. ODC) and root cause analysis. The second includes rules and guidelines (or a template) for defects description or defect data collection.

1. Data preparation and clean-up:
Once we have the necessary data it is important to organize it and perform some anonymization if required. Data organization is essential for the next steps, since it is important to have the data in a searchable and manageable manner.

4.3.2 Defects Classification

In order to efficiently and concretely tackle the important problems of critical software engineering, the first set of activities shall focus on an orthogonal classification of the sets of defects:

2. ODC:
Perform the ODC classification on the organized dataset. Enhancements and adaptations to the ODC taxonomy can be useful depending on the nature of the defects and the domain; however, these enhancements should be quite precise. For examples, see Silva and Vieira [12, 13].

3. ODC Analysis:
Provide a summary of the ODC analysis. This information gives the first hints about the quality of the dataset, which can provide some feedback to the implementation and V&V teams. For examples, Silva and Vieira [12, 13].

4.3.3 Defects Root Cause Analysis

The root cause analysis is composed by several steps that include analysis of the defects types, the triggers allowing defect detection, the defects that could have been detected earlier, and then prioritization and consolidation of these root causes leading to concrete proposed improvements:

4. Defect Type RCA:
Based on the different defect types, identify the possible generic root causes. This list of causes shall come from experience and expert judgement or a dedicated database where defect types are mapped to root causes. The list of root causes might be reduced or harmonized in step (8) below.

5. Defect Trigger RCA:
Based on the classified defect triggers identify the causes and V&V techniques (or triggers) that allowed the defects detection at the current defect detection stage. This list of causes shall come from experience and expert judgement or a dedicated database where defect triggers are mapped to root causes. The list of root causes might be reduced or harmonized in step (8) below.

6. Late Detection RCA:
With the list of defects that have slipped more than one lifecycle phase milestone identify the causes of the failures in the V&V and ISVV techniques that allowed the defects to propagate until a later stage in the development lifecycle. This list of causes shall be added/harmonized with the list from step (5) Defect trigger RCA.

7. Defects prioritization:
If required (for example to tackle the defects with high impact on the system, or due to the large amount of defects and respective causes) list of defect types and triggers can be prioritized according to a defined severity (for example, based on the main impact of those defects) and the respective root causes can be filtered according to the prioritized type and trigger.

8. RCA consolidation:
The list of root causes obtained in the previous steps (4–6) is consolidated according to the prioritization done in step (7). This consolidation can also contribute to reduce to an essential and more concrete list of causes.

9. Improvements Suggestions:
For all the root causes, define solutions or modifications to the processes, techniques, tools, training, resources, environment or application of standards. The solutions must cover the development activities to avoid the creation of defects and also the defect detection activities in order to identify the defects as soon as possible.

4.3.4 Improvements and Validation

The suggested improvements might be difficult to implement, and their effectiveness can vary from team to team. They shall contribute to improve the software quality and reduce the amount of defects, different defects can then surface, and this is why this process shall have a consistent process improvement in place:

10. Improvements Implementation:
The development and V&V teams must be informed about the required changes or adjustments (9. Improvements Suggestions), and the organization, management and quality planning shall decide on the improvements to implement for future projects.

11. Process Validation and Improvements:
At every step, it is possible to derive improvements to the process. Such improvements can be set to adjust to the company culture, to the project environment, to the customer requirements, etc. However, it is essential to measure the effectiveness of the implementation of the results (9. Improvements Suggestions and 10. Improvements Implementation) once the suggestions have been implemented and new defects (or no defects) have been collected. Note that Improvement can and shall also be about the current process, the defects classification scheme, the root cause analysis techniques and so on. The presented process shall be able to adapt and help in improving itself and the related techniques that compose it.

4.4 Results

This section presents the dataset case studies description and the results of application of the process described in the Section 4.3.

4.4.1 Characterization of the Systems

Our analysis is based on a set of real defects from ISVV activities in space projects. The projects include subsystems that compose satellite systems for three different domains (i) scientific exploration; (ii) earth observation; and (iii) telecommunications; covering different types of software, such as start-up or boot software, on-board application software, command and control units, payload software, and attitude and orbit control units. The engineering processes used in the selected missions were driven by the ECSS standards, namely the space engineering standard E-ST-40 [3] and the quality standard Q-ST-80 [4] which has a comparable lifecycle and similar strict requirements imposed by the European Space Agency.

The subsystems were developed according to functional and non-functional requirements mandated from ECSS and mission specifics. They were characterized by the following needs/objectives, which are common to space CSs, that were collected from the ECCS standards [3, 4] and the corresponding engineering interpretations of the specification documents from several missions:

- No crash or hang shall happen at any time;
- No dynamic memory allocation is allowed;
- Communication – Telemetry (TM)/Telecommands (TC) – must always be possible between ground control and the satellite;
- The system must implement a Safe Mode (with basic communications, patch and dump functionalities);
- Most systems shall have a very simple and stable start-up software (also called boot software);
- There must be a watchdog (Hardware and/or Software) or an alive signal;
- Systems are built with redundancy (at least Hardware);
- Most systems must include FDIR (Fault Detection Isolation and Recovery) functionalities to account for the environment and external faults;
- The systems must have high autonomy and some self-correction procedures;
- Systems are categorized with a criticality level related to the impact or consequences of system failures (in this case, the ECSS defined levels are: Catastrophic, Critical, Major and Minor or Negligible).

The projects are also characterized by:

- Requirements written in natural language (structured), highly based on documentation and non-formal processes and languages;
- Documentation in UML/SysML and PDF, with limited possibilities of automated verification and formal analysis;
- Programming languages such as C, Ada and Assembly, that are quite mature and low level languages;
- Unit tests performed in commercial tools (e.g. Cantata++, VectorCast, LDRA), commonly developed and adapted for the specific projects embedded systems and environments;
- Integration and system testing performed in specific validation environment (Software Validation Facility – SVF) developed for this purpose on a case by case situation, with HW emulation and HW in-the-loop, simulated instruments, etc.

4.4.2 Defects in the Dataset

Table 4.2 summarizes the 1070 defects in the dataset, divided by severity (having a major or minor impact in the system, or just being comments to improve the engineering) and considering the ISVV activities in which they were found. The defects have been originated from the analysis of more than 10,000 software requirements, more than 1 million lines of code (mostly C, Ada95 and some Assembly), and over 3,000 tests[1] (some unit tests, some integration tests). In practice, the objective of ISVV was to find issues in the project artifacts, report and classify them in a clear and consistent way for the customer to act upon.

4.4.3 Enhanced ODC Results

The results of the application of the enhanced ODC for space defects are summarized in Table 4.2 showing the five top types, triggers and impacts cover about 90% of the issues analysed. This observation suggests that actions can be taken to quickly improve the quality of systems, by tackling a limited amount of properties.

[1]The 3,000 tests correspond to only part of the requirements and code referred, as not all ISVV activities cover the full set of artifacts, e.g., for some projects only source code analysis was performed, no tests related to that specific codehave been assessed.

Table 4.2 Enhanced ODC classification results

Defect Type	Qty	%	Defect Trigger	Qty	%	Defect Impact	Qty	%
Documentation	515	48.1%	Traceability/Compatibility	309	28.9%	Capability	308	28.8%
Function/Class/Object	203	19.0%	Test Coverage	227	21.2%	Maintenance	264	24.7%
Algorithm/Method	96	9.0%	Consistency/Completeness	206	19.3%	Reliability	252	23.6%
Checking	69	6.4%	Logic/Flow	119	11.1%	Documentation	157	14.7%
Interface	56	5.2%	Design Conformance	119	11.1%	Performance	39	3.6%
Build/Package/Environment	52	4.9%	Rare Situation	26	2.4%	Usability	28	2.6%
Assignment/Initialization	46	4.3%	Test Sequencing	16	1.5%	Requirements	9	0.8%
Timing/Serialization	33	3.1%	Standards Conformance	14	1.3%	Migration	8	0.7%
			HW/SW Configuration	13	1.2%	Standards	4	0.4%
			Recovery/Exception	10	0.9%	Installability	1	0.1%
			Other Triggers	11	1.0%			
Total	1070	100%	Total	1070	100%	Total	1070	100%

The '*Documentation*' defect type represents almost half of the defects and '*Function/Class/Object*' represents almost 20% of the defects. This can be justified by the fact that CSs highly depend on documentation and documented evidences to prove the accomplishment of requirements and standards and to ensure qualification/certification of the systems by external entities. '*Function/Class/Object*' identifies functionality implementation deficiencies, especially at implementation level.

'*Traceability/Compatibility*' is the most frequent trigger, although '*Test Coverage*' and '*Consistency/Completeness*' are quite frequent. This suggests that the most efficient defect triggers are the simplest and most logical ones, namely those related to traceability, reviews and testing activities. This is due to the nature of the artifacts under analysis that require extensive documentation and creation of evidences that are developed over lifecycle phases depending on the previous phases artifacts.

In terms of the impacts, four of them are very important, namely: '*Capability*', '*Maintenance*', '*Reliability*' and '*Documentation*' (in this order). It is normal that Capability (i.e. functionality) is the most affected property but, in such space CSs, maintenance has a significant importance as well as the reliability requirements (see Section 4.4.1 regarding the needs/objectives of the target systems).

4.4.4 Enhanced ODC Defect Impact Analysis

The ODC Impact analysis can be used to prioritize the defect types/triggers to identify the development and V&V activities that might conduct to the defects with a high impact in the system. As "high impact", we consider equally the impacts in Capability, Reliability, and Maintenance, as they are the most severe since they represent three essential requirements of critical space systems: functional quality, non-functional reliability assurance, and maintainability. Though, for the purpose of this work, we have considered the importance of impact as the frequency that the defects affect system capability, reliability, or maintenance.

The following graphs in this section represent the defect impacts as they have been originated by specific defect types, and also as they have been uncovered by specific defect triggers. The graphs provide an idea of the importance of defect types (related to root causes) and how defects that lead to specific impacts have been detected with specific triggers.

4.4.4.1 Type vs. Impact

Figure 4.3 shows the defect types that have a high impact in the system (affecting Capability, Reliability and Maintenance). Defects with impact in Capability (blue dashed line) are mainly related with Function/Class/Object, Documentation and Algorithm/Method types, confirming that the functionality specification/implementation, the documented artifacts and the design decision in what concerns algorithms and methods to apply are the main contributors to defects that influence the system capability.

Defects with impact in Reliability (orange dotted line) are originated from Documentation, Checking, Function/Class/Object and also Algorithm/Method defect types. In this case, there is a new defect type that contributes significantly to reliability issues: Checking. It is clear that reliability (including redundancy, fault detection/monitoring, isolation and recovery) is often implemented with checks and verifications and so the importance of avoiding this type of defects to guarantee higher reliability.

Defects with impact in Maintenance (gray line) originate essentially from Documentation defect type. This is an expected result due to the fact that maintenance depends on documented artifacts that include installation and download instructions, user and developer manuals, and maintenance procedures.

The prioritization related with the three impacts is presented in the Section 4.4.5, namely in Table 4.3.

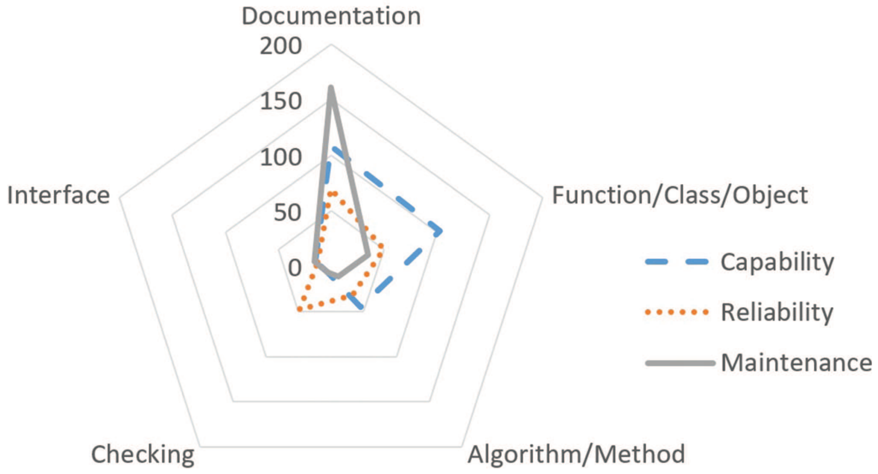

Figure 4.3 Defect type vs. defect impact.

Table 4.3 Summary of root causes for main defect types

Root Cause	Defect Types
Inefficient/insufficient reviews	Documentation; Function/Class/Object; Algorithm/Method; Checking; Interface
Ambiguous/missing/incorrect artifacts (documentation, requirements, design, tests)	Function/Class/Object; Algorithm/Method; Checking; Interface
Insufficient/Wrong tests (unit, integration, system, fault injection)	Function/Class/Object; Algorithm/Method; Checking; Interface
Limitations of the tools or toolsets that deal with documentation	Documentation
Lack of Completeness and consistency of system level (or previous phases) documentation	Documentation; Function/Class/Object; Algorithm/Method
Oversimplified documentation planning procedures	Documentation
Lack of time to produce, review and accept documentation artifacts	
Lack of importance given to some documentation artifacts	
Simplification of the product assurance processes related to documentation artifacts	
Limited engineers domain knowledge – lack of appropriate skills	Function/Class/Object; Algorithm/Method
Incomplete specifications in what concerns FDIR and erroneous situations	Checking
Lack of reliability and safety culture	Checking
Incomplete specifications in what concerns interfaces, environment and communications	Interface
Limited definition of the operation, usability, maintainability requirements	Interface
Lack of tools knowledge, programming languages, design languages	Function/Class/Object; Algorithm/Method
Version and configuration management procedures inappropriately implemented	Build/Package/Environment

4.4.4.2 Trigger vs. Impact

Figure 4.4 shows the defect triggers that allow detection of the defects with a high impact. The graph reinforces the importance of the 3 main triggers: a) Consistency/Completeness, b) Test Coverage, and c) Traceability/Compatibility as the most important (frequent) triggers (overall they allowed the detection of 77.0% of the issues). For this particular case, Reliability can

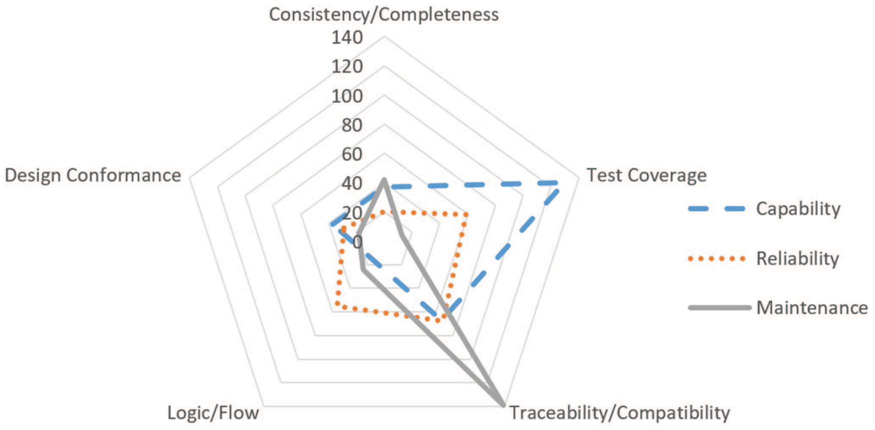

Figure 4.4 Defect trigger vs. defect impact.

be ensured with better Traceability/Compatibility analysis, Test Coverage and Logic/Flow analysis. Capability shall be assessed more efficiently with Test Coverage, Traceability/Compatibility assessment and Design Conformance Analysis. Maintenance defect impact can be mitigated with Traceability/ Compatibility and Consistency/Completeness analysis.

The results of the prioritization related with these three impacts are presented in Section 4.4.5, Table 4.4.

4.4.5 Consolidation of the Root Cause Analysis and Proposed Improvements

The defects with impact on capability, reliability, and maintenance, identified in Section 4.4, represent 77% of the total dataset. From these, we considered the top 6 defect types and the top 5 defect triggers (Table 4.2) because they account for more than 90% of the defects with high impact. Then we were able to identify the main root causes for the most important defect types (Table 4.3) and the most important defect triggers (Table 4.4).

This analysis results on a list of the most important causes of the defects identified during ISVV, and for the most important causes of failure in the verification and validation activities during the development lifecycle. For high defects with impact, the listed causes show that software engineering processes, methods and tools require some adjustments in order to become more efficient to produce more dependable and safe systems. The identified root causes are all related to existing development and V&V activities that require more careful application, especially in what concerns schedule

Table 4.4 Summary of root causes for main defect triggers

Root Cause	Defect Trigger
Lack of traceability verification culture	Traceability/Compatibility
Lack or inefficient usage of tools that support traceability across lifecycle phases	
Lack of appropriate test planning and test strategy definition	Test Coverage
Lack or inefficient testing tool and testing environment support	
Incomplete tests specification and execution	
Review process related root causes	Document Consistency/ Completeness (Internal Document)
Documentation related root causes	Document Consistency/ Completeness (Internal Document)
Deficient usage of tools and applicable processes	Document Consistency/ Completeness (Internal Document)
Unclear or missing/incomplete specifications	Document Consistency/ Completeness (Internal Document); Logic/Flow
Ambiguous or unclear architecture definition	Logic/Flow
Lack of usage of tools that support data and control flow analysis	Logic/Flow
Inappropriate architecture support tools or tool usage	Design Conformance
Deficient specification or design artifacts	Design Conformance

and planning pressures (or we can call it strategies as well), rigor and caution on the application of engineering processes, and V&V activities importance. The quality/product assurance strategies and the guidance from applicable processes and required standards are essential to ensure that these root causes are minimized.

The root causes presented (in Tables 4.3 and 4.4) have been ordered according to expert knowledge and experience applicable to the high impact defects, and intend to provide a preliminary ordering in what concerns their contribution to the high defect impacts.

The identified root causes for defect triggers indicate that improvements to the current processes, both development (to avoid the introduction of defects) and V&V (to detect the defects within the phase they are introduced) might be possible. At a higher level, the leading safety standards might require additional guidance to support development and V&V in order to reinforce that the product/quality assurance (PA/QA), and safety and dependability assessments should be properly realized, reducing the amount of defects caught by ISVV. The proposed improvements are guidelines derived directly from the root causes summarized in Tables 4.3 and 4.4 and from domain and expert knowledge of the authors and industrial contributors to this work. Their intent is to fulfil the needs of the development and V&V processes in order to avoid the most important and more frequent defects as those in our dataset.

From the development perspective, based on Table 4.3, the following measures should be considered:

- Define/redefine appropriate review methods, processes and tools and enforce their application at every stage of the SDP;
- Implement automated documentation generation processes and tools to avoid inconsistencies between artifacts/lifecycle phases;
- Use tools that integrate and manage all the phases of the lifecycle, such as concept specifications, requirements, architecture, source code, tests, etc.;
- Introduce/use tools with automatic validations (documentation completeness, design consistency, code analysis, control and data flow analysis);
- Provide training to the engineering teams, to improve the domain knowledge, the system or interfacing systems knowledge, standards knowledge and techniques and tools practice;
- Promote workshops or meetings to present the specifications/requirements, to discuss and clarify them before advancing to the following phase;
- Introduce additional guidelines or even specific requirements (e.g., by defining and specifying the reasoning behind the standards requirements and how to achieve them in full conformance) in the applicable standards (PA/QA, version and configuration control and development).

From the V&V perspective, based on the results in Table 4.4, the following measures should be considered:

- Define appropriate test plans and strategies, especially unit and integration tests. The soundness of the test plans and strategies will reflect in the success of the validation;
- Ensure appropriate (or automated) traceability analysis at every stage of the development lifecycle;

- Improve the testing completeness, coverage and reviews;
- Implement non-functional tests (fault detection, fault injection, redundancy, etc.);
- Apply or develop tools to verify and validate the implementation and design compliance.

4.5 Conclusions

This chapter presented a defects assessment process based on a field study on root cause analysis of 1070 defects in space software projects.

We proposed a general procedure to derive improvement suggestions for the systems and the analysis process itself applying an improved ODC taxonomy and examining the defect types, triggers and impacts. We have also prioritized the root causes based on their importance by considering the impact of the defects on capability, reliability, and maintainability, and proposed generic solutions to implementation (to prevent defects) and V&V (to effectively detect defects) in order to avoid these defects in future projects.

The outcomes of the field study show that, although CSs are already guided by appropriate development and V&V techniques and processes, most of the defects are caused by an inefficient usage or implementation of these techniques and processes. Appropriate guidance, additional requirements and constraints, better test strategies and tools that are able to help in the application of the techniques and processes would be essential to obtain better results (less defects). ISVV was originally able to detect the 1070 defects but could still be enriched by applying the proposed V&V actions in order to avoid defects slippage.

References

[1] RTCA DO-178B. (1992). *(EUROCAE ED-12B), Software Considerations in Airborne Systems and Equipment Certification*. RTCA Inc., Washington, DC.

[2] Sai Global. (2011). CENELEC EN 50128: Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems.

[3] ECSS. (2009). *ECSS-E-ST-40C, Space engineering – Software*.

[4] ECSS. (2009). *ECSS-Q-ST-80, Space Product Assurance – Software Product Assurance*.

[5] Silva, N., Lopes, R. (2012). “Overview of 10 Years of ISVV Findings in Safety-Critical Systems,” in *2012 IEEE 23rd International Symposium on Software Reliability Engineering Work-shops (ISSREW)* (New York, NY: IEEE), 83.

[6] Silva, N., Lopes, R. (2012). “Independent Assessment of Safety-Critical Systems: We Bring Data!” in *IEEE 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW)* (New York, NY: IEEE), 84.

[7] Silva, N., Lopes, R. (2012).“10 Years of ISVV: What’s Next?” in *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW)*(New York, NY: IEEE), 361–366.

[8] Silva, N., Lopes, R. (2011). “Independent Test Verification: What Metrics Have a Word to Say”, in *1st International Workshop on Software Certification (WoSoCER)*, ISSRE, Hiroshima, Japan (New York, NY: IEEE).

[9] Silva, N., Lopes, R., Esper, A., Barbosa, R. (2013). “Results from an independent view on the validation of safety critical space system,” in DASIA 2013, 14–16 May, Oporto, Portugal.

[10] EasterBrook, S. (1996). “The Role of Independent V&V in Upstream Software Development Processes”, in *Proceedings of 2nd World Conference on Integrated Design and Process Technology (IDPT)*, Austin, Texas, December 1–4.

[11] Chillarege et al. (2013). *Orthogonal Defect Classification v 5.2 for Software Design and Code* IBM.

[12] Silva, N., and Vieira, M. (2014). “Towards Making Safety-Critical Systems Safer: Learning from Mistakes,” in *ISSRE2014*, Naples, Italy.

[13] Silva, N., and Vieira, M. (2016). “Software for Embedded Systems: A Quality Assessment based on improved ODC taxonomy,” in SAC 2016, Pisa, Italy.

[14] Copeland, L. “Software Defect Taxonomies”. Available at: http://flylib.com/books/en/2.156.1.108/1/

[15] IEEE. (2010). *IEEE 1044-2009 Standard Classification for Software Anomalies*. Institute of Electrical and Electronics Engineers, New York, NY.

[16] Leszak, M., Perry, D. E., and Stoll, D. (2002). Classification and evaluation of defects in a project retrospective. *J. Syst. Softw*. 61, 173–187.

[17] Margarido, I. L., Faria, J. P., Vidal, R. M., Vieira M. (2011). “Classification of Defect Types in Requirements Specifications: Literature Review, Proposal and Assessment.” 2011 6th Iberian Conference on Information

Systems and Technologies (CISTI) (New York, NY: IEEE), 1–6. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5974237

[18] ESA ISVV Guide, issue 2.0, 29/12/2008, European Space Agency.

[19] ISO/IEC 12207:2008 *Systems and software engineering – Software life cycle processes*.

[20] IEEE 1012-2004 – *IEEE Standard for Software Verification and Validation. IEEE Computer Society*.

[21] Jones, M. (2005). *Software Engineering: Are we getting better at it?* ESA Bulletin 121, 52–57.

[22] Leszak, M., Perry, D. E., Stoll, D. (2002). "A Case Study in Root Cause Defect Analysis," in *Proceedings of 22nd Intl Conf SW Eng (ICSE'OO)* (New York, NY: IEEE), IEEE CS Press, Los Alamitos, CA, 428–437.

[23] Lutz, R. (1993). "Analyzing Software Requirements Errors in Safety-Critical," in *Embedded Systems, Proc IEEE Intl Symp Req Eng* (New York, NY: IEEE CS Press), 126–133.

[24] Weiss, K. A., Leveson, N., Lundqvist, K., Farid, N., Stringfellow, M. (2001) "An Analysis of Causation in Aerospace Accidents," in Digital Avionics Systems, 2001. DASC. 20th Conference (New York, NY: IEEE).

[25] Seaman, C. B., Shull, F., Regardie, M., Elbert, D., Feldmann, R. L., Guo, Y., and Godfrey, S. (2008). "Defect Categorization: Making Use of a Decade of Widely Varying Historical Data," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY: ACM), 149–57. Available at: http://dl.acm.org/citation.cfm?id=1414030

[26] Neufelder, A. M. (2012). *The top ten things that have been proven to impact software reliability*. Available at: http://www.softrel.com/downloads/TopTen.pdf

[27] Rao, R. (2014). *Root Cause Defect Classification (RCDC) for Documentation Defects*. Available at: http://www.stc-india.org/conferences/2014/presentations/Root%20Cause%20and%20Defect%20Classification%20for%20Documentation%20Bugs%20-%20Ramaa%20Rao.pdf

[28] Kumaresh, S. and Baskaran, R. (2010). Defect Analysis and Prevention for Software Process Quality Improvement. *Int J. Comput. Appl.* (0975–8887), 8.

[29] Silva, N., Vieira, M., Ricci, D., Cotroneo, D. (2015). "Assessment of Defect Type influence in Complex and Integrated Space Systems: Analysis Based on ODC and ISVV Issues," in *2015 IEEE International Conference on Dependable Systems and Networks Workshops (DSN-W)* (New York, NY: IEEE), 63–68.