18th International Conference on Space Operations, Montreal, Canada, 26 - 30 May 2025.

SpaceOps-2025, ID # 79

Quality Assurance for Ground Segments Development

Ghania Fau[1], **Louis Martin**[2]

Centre National d'Etudes Spatiales (CNES), *ghania.fau@cnes.fr*

Abstract

The development of ground segments is a critical step in the successful operation of space missions. This is true regardless of whether the ground segment is intended for use in a control center, mission center, or data processing center. At CNES (Centre National d'Etudes Spatiales), Quality Assurance is a key function in ensuring that ground systems meet the specified requirements for performance, reliability, maintainability, and security. It is therefore essential that the requirements are analysed for clarity, completeness and testability before the development process begins. Test plans should be created including test cases and test approach. Subsequently, verification and validation protocols (unit testing, integration testing, system testing, etc.) must be carried out to assess the software's performance. In this regard, it is essential to ensure that CNES coding standards are met. Consequently, integrating automated testing, including quality assurance tools, into a CI/CD pipeline could prove invaluable in ensuring the desired quality level at each stage of development.

The use of Software Factories is becoming more prevalent for CNES developments. They are employed to improve the efficiency and quality of the software being developed, as well as to enhance collaboration within the team. The use of a software factory in ground segment development offers several significant benefits, including automation, reusability, maintainability, quality assurance, and cost efficiency. These benefits are particularly crucial in light of the fact that the control and mission centers are now based on standardized and generic product lines, including LdP ISIS for control centers, LdP SIRIUS for Flight Dynamic Systems, and SPIRIT for mission centers.

This abstract outlines the Quality Assurance processes that have been implemented throughout the ground segment development process, from the initial requirements definition stage to system integration and validation (AIV). It also highlights the significant capital gains that can be achieved by integrating quality assurance and other disciplines within the CNES Software Factory.

Keywords: Ground Segments, Quality Assurance, processes, Software Factory.

Acronyms/Abbreviations

AIV: Assembly Integration Validation
AR: Acceptance Review
CDR: Critical Design Review
CI/CD: Continuous Integration / Continuous Delivery
CNES: Centre National d'Etudes Spatiales (French National Space Agency)
DDR: Detailed Design Review
ECSS: European Cooperation for Space Standardization)
ISIS: Initiative for Space Innovative Standards
PA/QA: Product Assurance/Quality Assurance
PDR: Preliminary Design Review
QR: Qualification Review
SAALTO: Segment Sol multi-missions ALTimétrie, Orbitographie et localisation précise
SWOT: Surface Water Ocean Topography
TC/TM: Telecommand/Telemetry
TRB: Test Review Board
TRR: Test Readiness Review

1. Introduction

The development of ground segments is a critical step in the successful operation of space missions, whether scientific, military or dual-use, consisting of one or several satellites. CNES Ground segments include ground stations that ensure communication, tracking and orbit management, control centers that are responsible for commanding, monitoring and controlling the satellite, and mission centers whose role is to plan and execute the mission, as well as to process and distribute the mission data.

For several years now, CNES missions have been using product lines such ISIS (Initiative for Space Innovative Standards), a generic Command and Control Center, and multi-missions center as SSALTO (Segment Sol multi-missions ALTimétrie, Orbitographie et localisation précise) used for Altimetry missions. The development of these product lines is undertaken by subcontractors in collaboration with the CNES teams. In-house developments, executed by the CNES teams themselves, are carried out to supplement these developments with libraries, procedures, playbooks, configurations, workflows…

The development of these ground segments is subject to a multi-phase process including requirements definition, design, production, verification and validation.

The paper first presents the Quality Assurance processes applied to ensure that ground systems meet the specified requirements for performance, reliability, maintainability, and security. The second section defines the CNES Software Factory, describes how it is used and the benefits to be gained by using it. The following section describes each of the Software Factory's tool and the link with the Engineering and Quality Assurance processes will be highlighted. The final section presents a series of examples of code quality analysis using the CNES Software Factory.

2. Quality Assurance role

The development of CNES ground segments heavily relies on the ECSS (European Cooperation for Space Standardization) standards, which are employed to ensure consistency, quality, and reliability across the various phases of the development process and throughout the system's lifetime. Figure 1 illustrates the main standards applicable to ground segments development, and here we will be focusing on the ECSS-E-ST-40 "Space Engineering Software" [1] and ECSS-Q-ST-80 "Software Product Assurance" [2] standards. The implementation of these two standards is subject to the classification of software criticality, as defined by the ECSS-Q-ST-30C standard [3].

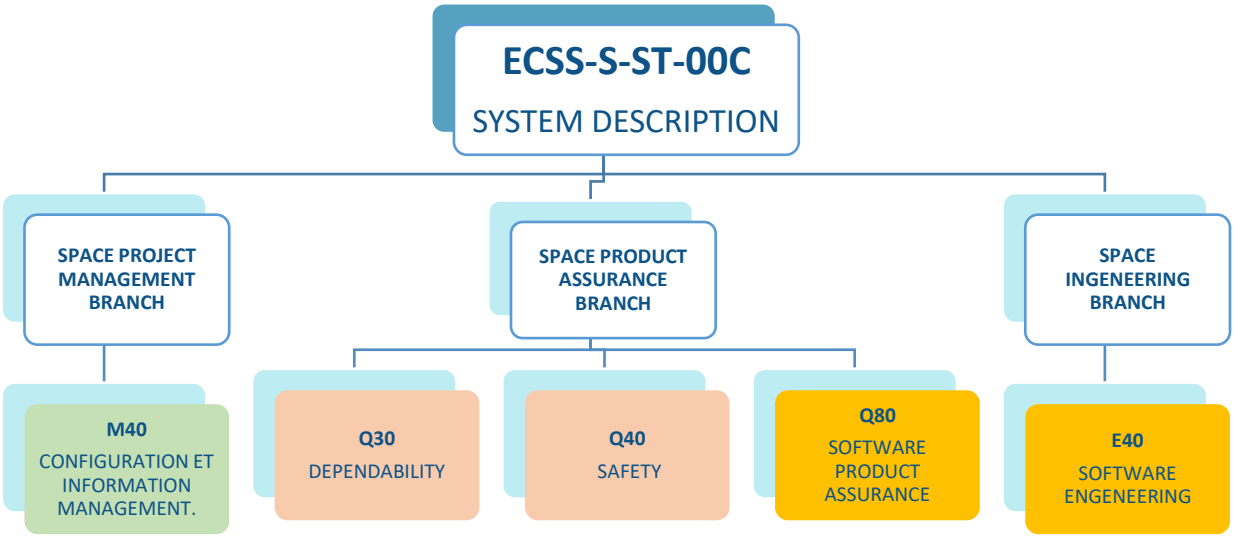

Figure 1 : Main ECSS standards applicable to ground segments development

Therefore, for the development and maintenance of CNES ground segments, PA/QA activities are implemented for each development to ensure that the products meet the quality requirements derived from the quality models, and which are established according to the technical specification. In order to avoid redundancy with the ECSS-Q-ST-80

standard, the present article will not provide a comprehensive overview of all the processes involved, but rather will focus exclusively on those that are deemed to be pertinent to the subject matter.

Software management process:
Software life cycle including phases, their inputs and outputs, and joint reviews, in accordance with the overall project constraints and with ECSS-M-ST-10 shall be defined. Technical reviews as TRR, TRB, DDR, and project, such as PDR, CDR, QR, AR, are to be conducted throughout the project phases.

Software requirements and architecture design:
This process consists of defining the technical specification including functional and non-functional requirements, in order to satisfy performance, safety, reliability, robustness, quality, maintainability, configuration management, security, privacy, metrication, and verification and validation requirements. Architectural design shall also be defined. The corresponding outputs shall be documented.

Safety and dependability: CNES products are subject to rigourous safety and dependability analyses in order to determine their criticality, as defined by the severity of the product's failure consequences as illustrated in Table 1 :

Software criticality categories applied at CNES :

Category	Severity	Dependability	Safety
A	Catastrophic	Failures propagation	Loss of life, life-threatening or permanently disabling injury or occupational illness; Loss of system; Loss of an interfacing manned flight system; Loss of launch site facilities; Severe detrimental environmental effects.
B	Critical	Loss of mission	Temporarily disabling but not life-threatening injury, or temporary occupational illness; Major damage to interfacing flight system; Major damage to ground facilities; Major damage to public or private property; Major detrimental environmental effects.
C	Major	Major mission degradation	–
D	Minor or Negligible	Minor mission degradation or any other effect	–

Table 1 : Software criticality categories applied at CNES [3].

At CNES, the following criticalities are employed in the majority of cases:

Criticality	B	C	D	NOS
System	Flight Software	TC/TM software	Operational component which doesn't cause major mission degradation	Non operational Software

It is important to note that a ground segment can be composed of sub-systems with different criticalities.

Coding: coding standards, measurements, criteria and tools should be identified before coding activities start.
For a CNES development, the following handbooks are used in order to define measurements and criteria which are checked in parallel with the coding to ensure direct feedback and efficiency:

- ECSS-Q-HB-80-04A (Software metrication programme definition and implementation)
- RNC-CNES-Q-HB-80-501 (Common rules for the use of programming languages)
- RNC-CNES-Q-HB-80-527 (Coding rules for JAVA language) and handbooks for each language.

Integration and testing : The integration and testing strategy is to be defined in an AIV plan, including responsibilities, schedule and testing approach. Test procedures and data shall be provided. At CNES, unit tests, integration tests, and system tests are performed. In the event of any change, regression tests are conducted: configuration, platform, data…
The test coverage goals for each testing level are identified according to the criticality of the software. The conformance of the test coverage is then checked against the requirements.

Validation: At CNES, validation is performed against the technical specification using mission's data and scenario, in order to ensure that the product will perform successfully in a representative operational environment
A Test Readiness Review is held before testing and a Test Readiness Board is held after.

Configuration management and control: the code, data used for validation, procedures…are put under configuration control after unit tests and integration tests are successfully performed. At CNES, Git, SVN, CVS tools are used to manage the code sources.

Delivery and acceptance: In the event of the product being developed and delivered by an industrial company, CNES is responsible for conducting site acceptance tests in order to validate the product in the most representative environment possible.

Operation process and maintenance: availability, operability and maintainability of the software should be guaranteed to the software users.

3. CNES Software Factory

The evolution of DevOps with its CI/CD practice and the desire to break down the silos between development and operational teams, have had a significant influence on the creation of CNES Software Factory. This has been achieved by promoting automation, collaboration, and efficiency in the software development lifecycle.

A Software Factory is defined as a structured environment that optimises software production. Its overarching objective is to standardise, automate and industrialize software development processes, thereby enhancing quality and productivity, while concurrently accelerating software product delivery.

The following fundamental principles underpin the concept of a Software Factory:

- Standardisation, which aims to standardize development methods, tools and practices with a viewto guaranteeing consistent quality.
- Automation, which aims to reduce manual and repetitive tasks to minimise human errors and increase efficiency (Continuous Integration (CI) and Continuous Delivery (CD)).
- Reuse of components is encouraged to maximise the use of existing, proven software building blocks in order to accelerate development and reduce costs (e.g. shared libraries, code templates, tooling, graphic guidelines).
- Versioning enables source code modifications to be historically recorded (backup, traceability) and managed. It enables flexible collaborative working, allowing several developers to work on the same source code using commit and merge mechanisms.

- The integration of quality control mechanisms throughout the development cycle (code review, automated testing, quality metrics) is paramount to ensure the consistent delivery of high-quality software.

As presented in Figure 2, CNES Software Factory implements ECSS-Q-ST-80 processes and partially ECSS-E-ST-40 processes. It covers the following functionalities:

- Source code configuration management
- Application build with
  - Application security analysis
  - and code quality analysis
- Application and code testing
- Management of application binaries (storage, versioning and distribution)

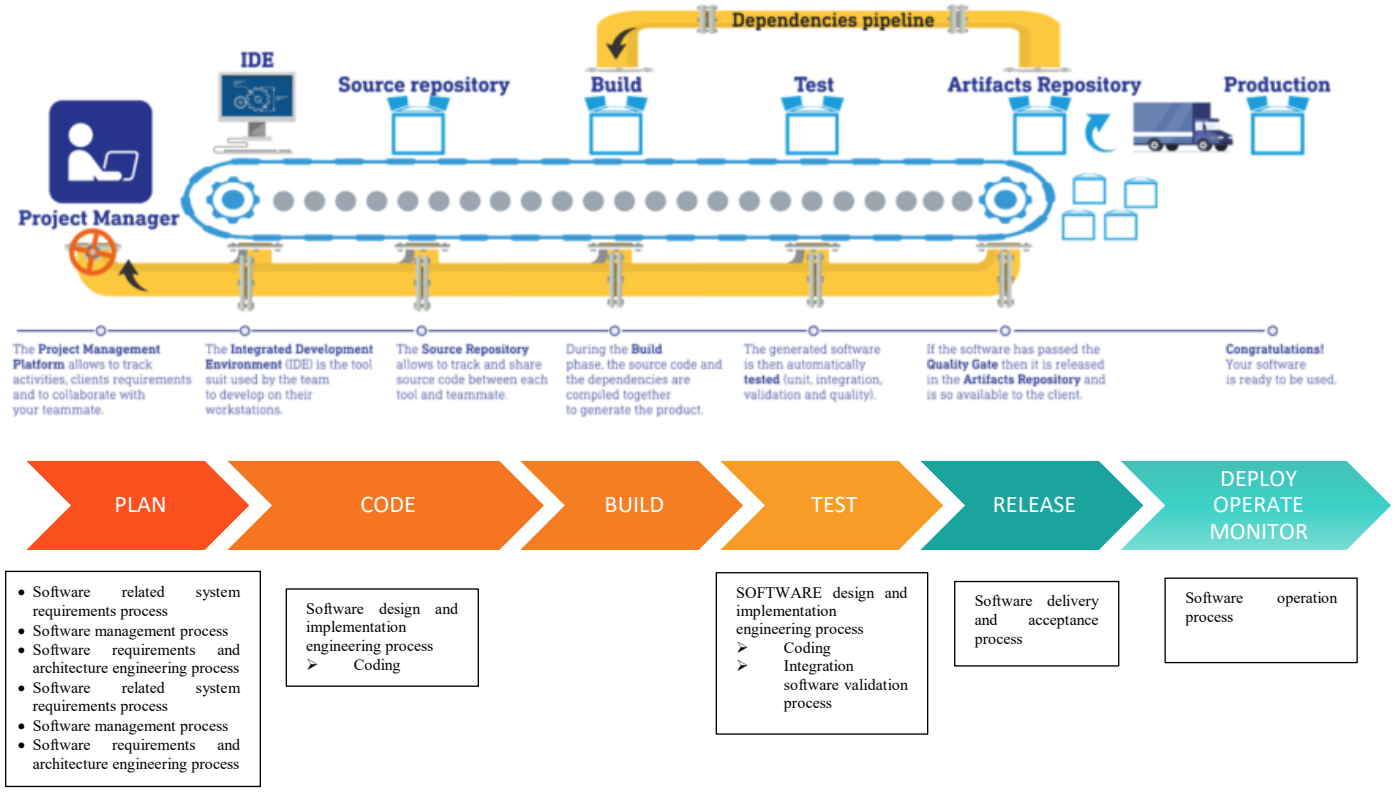

Figure 2 : CNES Software Factory and its link to the ECSS processes

4. CNES Software Factory tools

The CNES Software Factory tools were implemented in such a way to cover the different phases of CI/CD and DEVOPS approach.

PHASE	TOOL	DETAILS
Code	Git & Gitlab	For configuration management of application source code
Build	Jenkins or Gitlab-CI	For the definition and parameterization of Jobs (steps) for the construction of the project deliverable
Test (partially)	Jenkins ou Gitlab-CI, SonarQube, JFrog XRay	For setting up launch Jobs of : • Unit tests • code quality control with SonarQube (coding rules, code coverage, etc.) • security control with JFrog XRay (requires you to have already pushed a version of the binary into JFrog Artifactory)
Release	Jenkins or Gitlab-CI, JFrog Artifactory	For storing deliverables and making them available to the various project stakeholders (internal or external)

Table 2 : *The CNES Software Factory tools and their link with the CI/CD phases [4].*

The following is a concise definition of the tools presented in Table 2 :

- Git: Git is a distributed version control system used primarily for tracking changes in source code during software development. It allows multiple developers to work on the same project simultaneously without interfering with each other's contributions. It is notable for its ability to offer flexibility and efficiency in the management and collaboration on code. Notable features include version control, the ability to work with branches, commit history and collaboration.

- Gitlab: Gitlab is a powerful Git repository manager. It facilitates creation, cloning, pushing, and pulling repositories, thereby providing developers with version control capabilities that ensure changes to code are tracked, managed, and merged efficiently. Additionally, Gitlab can be employed for issue tracking, further augmenting its functionality.

- Jenkins/Gitlab-CI: Jenkins and Gitlab-CI are used for the automation of specific software development processes, including building, testing, and deploying applications. They easily integrate with version control systems such as Git, enabling the automatic triggering of builds upon the occurrence of changes within a version control system. Furthermore, Jenkins and Gitlab-CI can also integrate with static code analysis tools, such as SonarQube (described further), to ensure continuous monitoring of the quality of the developed product.

- Artifactory: Artifactory is a repository manager that is utilised for the management and storage of dependencies and binary artefacts throughout the software development lifecycle. These artefacts include, but are not limited to, such items as compiled code, libraries, configuration files and container images.

- JFRog XRay: JFRog XRay is a security and compliance tool designed to scan, analyse, and monitor software artefacts for vulnerabilities and licensing issues. Indeed, JFRog XRay integrates advanced security scanning, automation, and knowledge directly into the DevSecOps workflow. It is notable fot

its capacity to analyse code and binaries in order to provide accurate vulnerability context, thereby avoiding the occurrence of false positives, and thus resulting in time saving, and agility increase.

- Sonarqube: SonarQube is used for continuous inspection of code quality, incorporating the subsequent characteristics:
    - Code Analysis: It performs static code analysis to detect a wide range of issues, from syntax errors to security vulnerabilities.
    - Multi-language Support: SonarQube supports a variety of languages like Java, Python, JavaScript, C#, C++ and more.
    - Quality Gates: It allows teams to define quality gates (specific thresholds for code quality) which must be passed before code is merged or deployed.
    - Code Coverage: It integrates with testing frameworks to measure the percentage of code covered by tests, helping ensure adequate test coverage.
    - Security Vulnerabilities: SonarQube can highlight security risks in the code, including potential vulnerabilities that could be exploited.
    - Technical Debt: It tracks "technical debt," which is essentially the amount of effort needed to bring code quality up to acceptable levels.
    - Integration: SonarQube integrates with tools like GitLab, Jenkins and Gitlab-CI to provide feedback on code quality in the context of a CI/CD pipeline.
    - Reports & Dashboards: SonarQube provides detailed reports and visual dashboards to track code quality metrics, trends, and improvements over time and development lifecycle.

5. Software Product Assurance / Quality Assurance role in relation to the Software Factory

In the context of a Software Factory, the role of the PA/QA is important to insure that the software produced meets the required standards of quality, functionality, performance and security. This role can be distilled into several key activities:

1. Definition of Quality Standards and metrics: the QA manager sets the standards according to the customer specifications including coding standards, change management, testing strategy and quality metrics. Quality profile and quality gate are both defined jointly with the development team.
2. Test strategy: the QA manager ensures that all components of the software go through rigorous testing, such as unit tests, integration tests, regression tests, and acceptance tests, to identify and resolve defects early in the development process. In addition, the conformance of the test coverage is checked against the requirements.
3. Change management: the QA manager ensures that configuration management and version control processes are correctly implemented in the Software Factory and that changes on the product go through a rigorous testing process before deployment.
4. Quality control: in order to provide insights into the health of the software, the QA manager tracks the metrics defined in section 5.1. Such as code coverage, complexity, code size, fault density and failure intensity, number of failures, comment rate…This helps to detect issues early in the development cycle. SonarQube tool automatically analyses the code and detect issues such as bugs, code smells, security vulnerabilities, dead code, and adherence to predefined coding standards.
5. Risk management: the QA manager identifies throughout the development risks such as technical debt, scalability concerns, or security vulnerabilities. Moreover, he works with the teams in order to set up a risk mitigation strategy ensuring the software product is resilient and reliable.

6. Documentation and reporting: the QA manager ensures that specifications, tests cases, tests results, product quality status…are documented.
7. Continuous improvement: one of the key activities PA/QA manager is to continuously monitor and improve the processes on one hand, and to work with the development team to assess and improve the software development lifecycle process. This includes suggesting best practices, optimizing workflows, and adopting new tools or methodologies (e.g., Agile, DevOps).

6. Code quality analysis

As stated in the previous chapter, the QA manager performs using tools a static code analysis in order to control the code quality and to ensure that it complies with the established coding standards.

This chapter provides two examples of code analyses that were conducted in the CNES Software Factory using the SonarQube tool.

Analysis #1: The following features were selected for illustration in this example:

- Analysed product: a component of the X satellite Mission Center
- Number of the code lines: 6400 lines
- Coding languages: Python & Shell
- Quality profile: a Quality profile is defined as a set of rules that, when violated, results in issues being raised within the codebase. In the CNES Software Factory, a number of Quality profiles are defined, the criticality of the software dictating the applicable profile.
- Quality gate: a Quality gate is an indicator of whether the code meets the minimum level of quality required. It consists of a set of conditions that are applied to the results of each analysis. The Quality gate for the present example is as follow:

Conditions on New Code

Metric	Operator	Value
Coverage	is less than	0.0%
Duplicated Lines (%)	is greater than	5.0%
Line Coverage	is less than	100%
Maintainability Rating	is worse than	A (Technical debt ratio is less than 0.0%)
Blocker Issues	is greater than	0
Critical Issues	is greater than	0
Major Issues	is greater than	0
Reliability Rating	is worse than	A (No bugs)
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A (No vulnerabilities)

Figure 3 : X Quality gate

In order to facilitate the comprehension and interpretation of the following analysis, the key definitions are given below:

Metric	Definition
Bug	A reliability related issue that represents a coding mistake. It can lead to an error or unexpected behavior at runtime.
Code smell	A maintainability related issue in the code that makes the code confusing and difficult to maintain.
Vulnerability	A security related issue that represents a security weakness.
Security hotspot	Security sensitive pieces of code that need to be manually reviewed. Upon review, one will find either that there is no threat or that there is vulnerable code that needs to be fixed.
Technical debt	The estimated time required to fix all maintainability issues and code smells.
Code coverage	The percentage of the codebase exercised by automated tests.

Table 3 : definition of SonarQube metrics

Figure 4 and Figure 5 show two dashboards: one of the new code analysis and one of the overall code analysis. These dashboards integrate the metrics defined below, as well as the Quality Gate. These metrics are tracked over time in order to assess the code quality and check its conformance against quality standards.

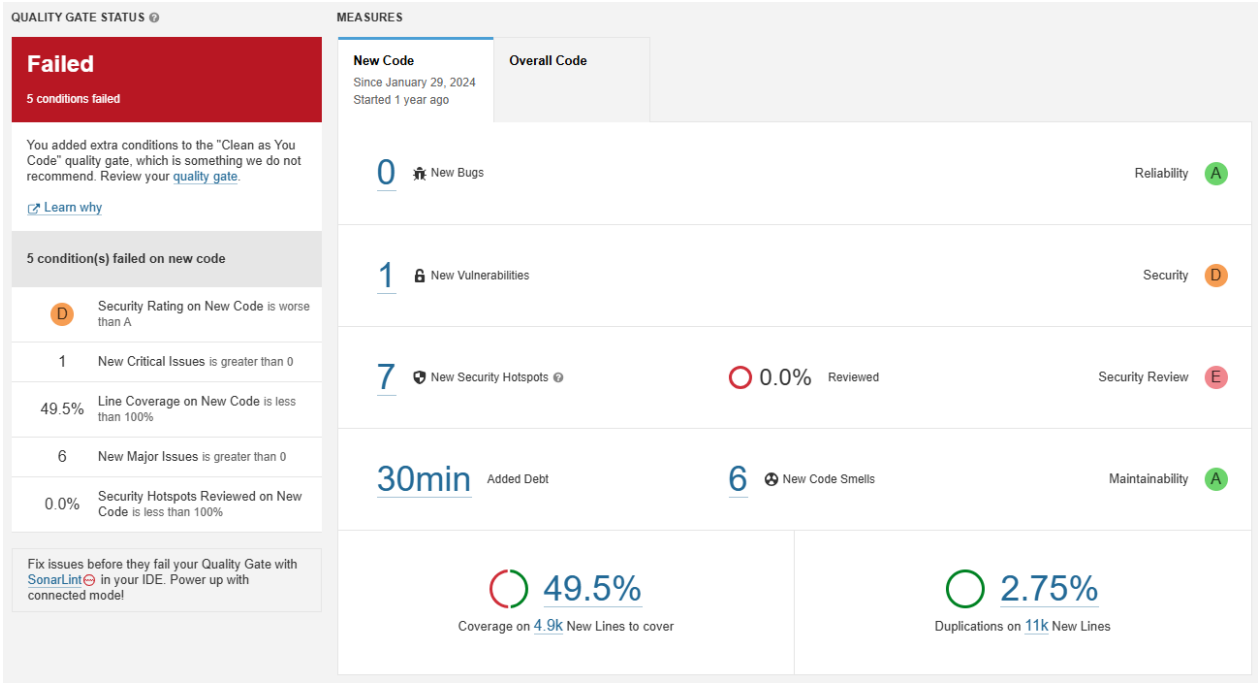

Figure 4 : X New code metrics

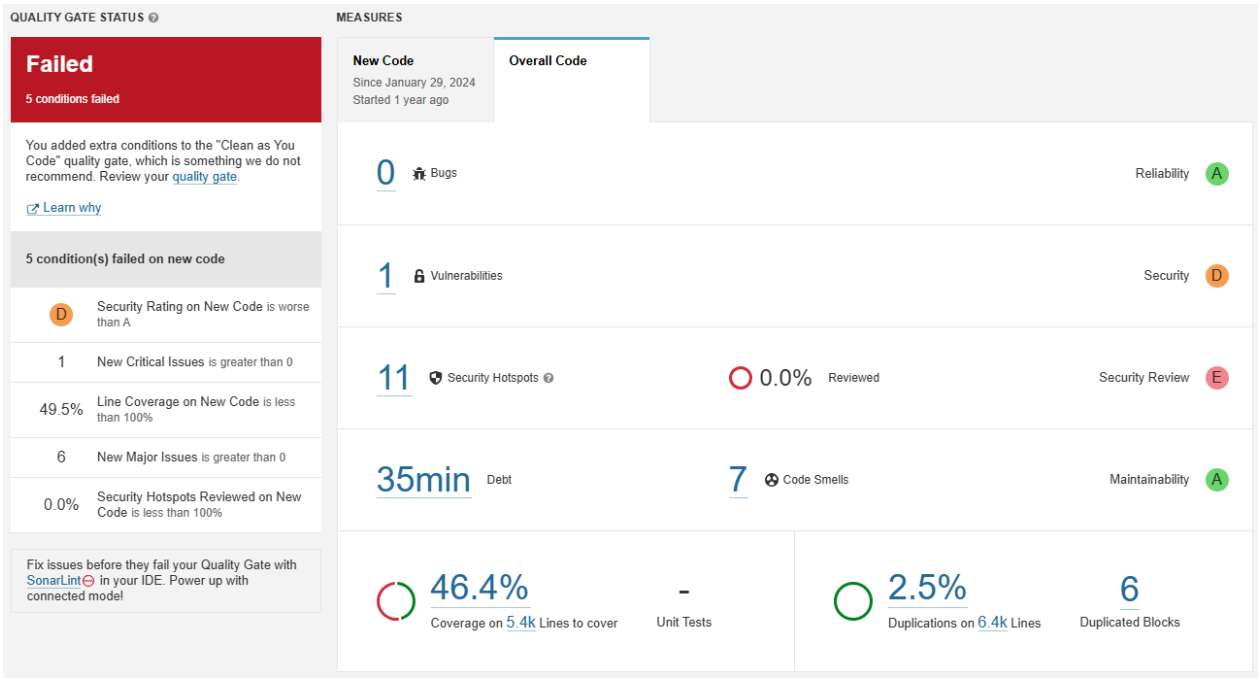

Figure 5 : X Overall code metrics

A detail of the issues is given below together with number of Minor, Major, Critical and Blocker issues:

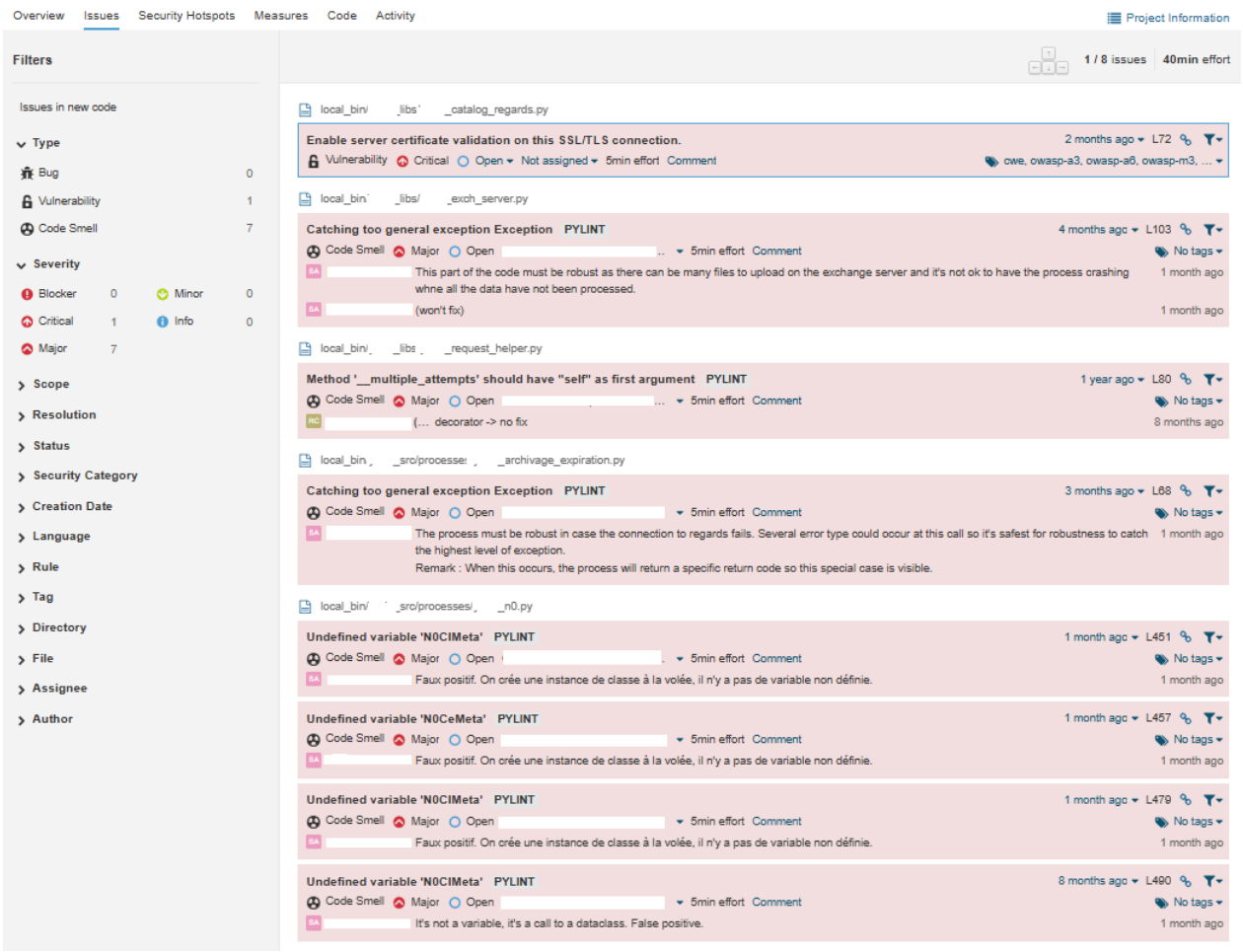

Figure 6 : X Quality analysis issues

ISSUE SEVERITY	DEFINITION
Info	Neither a bug nor a quality flaw, just an info.
Minor	A quality flaw that can slightly impact the developer's productivity. For example, "lines should not be too long" is considered minor issues.
Major	A quality flaw that can highly impact the developer's productivity. A duplicated blocks, or unused parameters are examples of major issues.
Critical	Either a bug with a low probability to impact the behavior of the application in production or an issue that represents a security flaw. An empty catch block or SQL injection would be a critical issue. **The code must be reviewed immediately**.
Blocker	Bug with a high probability to impact the behavior of the application in production. For example, a memory leak, is considered as blocker issue and **must be fixed immediately.**

Table 4 : definition of issues severities in SonarQube

The interpretation of the SonarQube analysis is presented below:

Quality gate	The analysis focuses on new (or modified) code only. This allows to be resilient to any changes in the analysis tool, and to update coding rules if standards evolve during the development, without impacting the code already written at a given moment. Since all new code is "clean", technical debt does not increase, and will even decrease over time. This is also the simplest and least costly case to manage, focusing solely on the code currently being written.
Issues	• There are 0 "bugs": No code that could lead to a bug is detected by a SonarQube rules. • There is 1 "vulnerability" classified as critical and 11 "security hotspots". The vulnerability is either to be corrected, or an analysis by a security manager is required if it is decided not to correct it, in which case a waiver will probably be required. • For the 11 "security hotspots": there aren't necessarily any vulnerabilities, but points of attention, an analysis is required. • There are 7 "code smells": these errors don't necessarily lead to a bug, but they can lead to a discrepancy between what we understand the code to do and what it actually does. o As a general rule, Blocker and Critical should always be corrected; here, no errors fall into this category. o On this particular sonarQube analysis, each error was justified and explicitly marked as "not to be corrected". - 5 false positives, i.e. a detection that has no reason to exist - 2 using Python's "Exception" class to detect a runtime error. In fact, in Python, it's generally preferable to target a specific error such as "ImportError", "MemoryError" or "ZeroDivisionError". There are, however, cases where using a broad exception is relevant, so be careful to keep this practice limited.
Test coverage	Test coverage is around 50%, which is low. The "top 70%" is often considered inexpensive to cover by testing, and we recommend coverage of at least 80% or even 90%. Coverage of 100% is sometimes costly, but not always necessary if the project is non-critical.
Duplication rate	The duplication rate here is very low, so no advice is required. However, be careful not to increase it.
Overall opinion	The quality of this project is good, but beware of security issues. Compliance with the quality gate in the past has enabled us to achieve a good level of quality. Only testing is lacking, as coverage is too low.
Recommended actions	- Fix the vulnerability - Improve test coverage or ensure code is tested differently (integration testing) - Security analysis of the 11 security hotspots to ensure that no security vulnerabilities are present. - No errors to correct. Continue to respect Quality Gate in the future.

Analysis #2: The following features were selected for illustration in this example:

- Analysed product: a component of the SWOT CNES Mission Center
- Number of the code lines: 580 lines
- Coding languages: JAVA & YAML
- Quality profile: RNC Criticality D (Référentiel Normatif du CNES derived from the ECSS standards)
- Quality gate :

Conditions on New Code

Conditions on New Code apply to all branches and to Pull Requests.

Metric	Operator	Value
Coverage	is less than	0.0%
Duplicated Lines (%)	is greater than	5.0%
Maintainability Rating	is worse than	A (Technical debt ratio is less than 5.0%)
Blocker Issues	is greater than	0
Critical Issues	is greater than	0
Major Issues	is greater than	0
Reliability Rating	is worse than	A (No bugs)
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A (No vulnerabilities)

Figure 7: SWOT CNES Mission Center Quality Gate

Figure 8 : Example of SWOT CNES Center Mission Center New code metricsshows the new code analysis metrics:

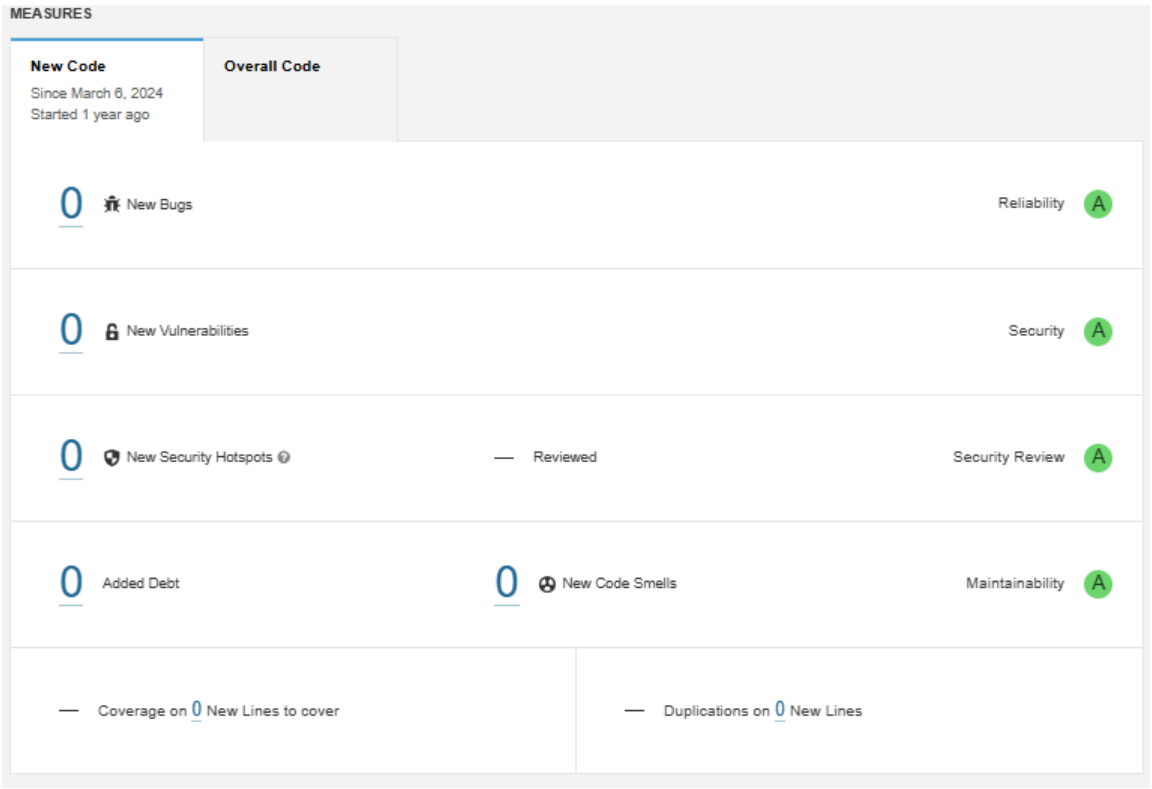

Figure 8 : Example of SWOT CNES Center Mission Center New code metrics

As the analysis includes many violations, not all are presented here. This figure highlights the amount of Minor, Major, Critical and Blocker issues :

Figure 9 : Extract of SWOT CNES mission center quality analysis issues

The interpretation of the SonarQube analysis is presented below:

Quality gate	The analysis focuses on new (or modified) code only. This allows to be resilient to any changes in the analysis tool, and to update coding rules if standards evolve during the development, without impacting the code already written at a given moment. Since all new code is "clean", technical debt does not increase, and will even decrease over time. This is also the simplest and least costly case to manage, focusing solely on the code currently being written.

Issues	• There are 8 "bugs": These errors need to be carefully analysed, as they can directly cause a malfunction or even a program shutdown.
	• In this case, the errors are specifically linked to the JAVA language, not to an "algorithmic" problem (such as division by zero).
	• There are 0 "vulnerabilities.
	  ○ SonarQube detects no vulnerabilities, but this does not mean that the code is free of security flaws. SonarQube only checks for the most common flaws. Additional analysis of JAVA dependencies may be necessary, depending on the level of confidentiality of the project. In any case, it's a good idea to have an SSI manager on hand to help with these issues.
	  ○ There are 166 "Code smells": These errors do not necessarily lead to a bug, but can lead to errors between what we understand the code to do and what it actually does.
	  ○ There are 0 "blocker" and 7 "critical" errors. In this case, the errors are defects in the factorization of the code. Variables are duplicated, which can lead to code that is not very scalable and difficult to evolve (or to correct if a bug occurs in these portions of the code).
	  ○ There are 106 "Major" errors, but despite their initially alarming nature, correction is not always mandatory, depending on the project's resources (although it is recommended in all cases). To avoid excessive use of human resources on this task, it is advisable to correct these errors as the code is edited.
	  ▪ On this analysis, many of the errors are "magic numbers", i.e. numbers that would be better placed in a variable to better understand what they represent.

	○ There are 46 "minor" errors. This is the type of error you need to deal with to go from good code to perfect code. ○ There are 7 "Info" errors, which are only of limited interest for formatting purposes.
Test coverage	Test coverage is not performed. We have to make sure that the analysis is performed by another mean.
Duplication rate	The duplication rate is null, so the code is correctly factorised.
Overall opinion	The quality of this project is good, despite a few areas for improvement. In the development phase, the quality is quite satisfactory. In the finalization/delivery phase, it could be worthwhile correcting residual errors. However, it is essential to know the test coverage by then. The earlier this is implemented, the lower the risk of delays.
Recommended actions	- Deal with critical code smells - Implement test coverage - Correct the 8 bugs with the help of someone experienced in JAVA - Depending on project resources and deadlines: reduce the number of "Major" code smells

7. Conclusion

In order to ensure the operation of complex systems, many ground segments developments are carried out at CNES, which are subject to a multi-stage process from requirements definition to operation. In order to operate these complex systems, the products developed must pass through several stages during the development process. Quality Assurance plays a key role in ensuring that these developments meet the defined specifications in term of performance, reliability, maintainability and security.

The use of the Software Factory for developments at CNES has demonstrated its ability to facilitate teams collaboration, automation, efficiency and high quality. Its various tools can be used to manage several functions: code, build, test, deploy and release.

In addition to conventional QA/PA methodologies and processes, the implementation of the SonarQube tool within the Software Factory has engendered a paradigm shift in the realm of code analysis, signifying an enhancement in terms of quality, efficiency, and continuous enhancement.

References

[1] ESA Requirements and Standards Division, "Space engineering - Software", *European Coordination for Space Standardization*, ECSS-E-ST-40C, URL: http://ecss.nl/standard/ecss-e-st-40c-software-general-requirements/

[2] ESA Requirements and Standards Division, "Space product assurance - Software product assurance", *European Coordination for Space Standardization*, ECSS-Q-ST-80C Rev.1, URL: http://ecss.nl/standard/ecss-q-st-80c-rev-1-software-product-assurance-15-february-2017/

[3] ESA Requirements and Standards Division, "Space product assurance – Dependability", *European Coordination for Space Standardization*, ECSS-Q-ST-30C Rev.1, URL: http://ecss.nl/standard/ecss-q-st-30c-rev-1-space-product-assurance-dependability-15-february-2017/

[4] CNES Sofwatre Factory URL: Usine Logiciel - Process Métier - Usine Logiciel - Process Métier - Confluence