Defence Science Journal, Vol. 56, No. 1, January 2006, pp. 31-38

Software Quality Assurance–Challenges in Launch Vehicle Projects

Poofa Gopalan, S.S. Uma Sankari, D. Mohan Kumar, and R. Vikraman Nair
Vikram Sarabhai Space Centre, Thiruvananthapuram–695 022

ABSTRACT

Launch vehicle projects now depend on software, more than ever before, to ensure safety and efficiency. Such critical software systems, which can lead to injury, destruction or loss of vital equipment, human lives, and damage to environment, must be developed and verified with high level of quality and reliability. An overview of current quality practices pursued in launch vehicle projects is presented in this paper. These practices have played a vital role in the successful launch vehicle missions of Indian Space Research Organisation. As complexity of software increases, the activity that gets affected is nothing but, software quality assurance (SQA). The SQA team is facing a lot of challenges in current practices. This paper brings out such challenges in different phases of software life cycle. A set of key points to some techniques and tools, that could contribute to meet the software quality assurance challenges in launch vehicle projects, are also discussed.

Keywords: Software quality assurance, launch vehicle, testing, simulation, verification, validation, software fault injection

1. INTRODUCTION

Indian Space Research Organisation has a long history of producing reliable launch vehicles and achieving repeated success in vehicle lanch missions. Key to the success of these missions is the quality assurance practices employed to ensure robust, fault-tolerant designs, to ensure a final product that faithfully embodies those designs, and to physically verify that launch and environmental stresses will indeed be well-tolerated by the systems.

1.1 Criticality of Software

Space industry around the world has witnessed a number of software-related mission failures. In 1993, the first flight of polar satellite launch vehicle (PSLV) ended in a failure during the transition from the second stage to the third stage due to control system error exceeding the full-scale value, which resulted in a software overflow. In 1996, Ariane501 mission failed due to an unprotected conversion from a 64 bit floating to a 16 bit signed integer value[1]. The piece of software used for the conversion was a reused component of the previous flight, which was not adequately tested for the present mission conditions.

The next major mission failure due to software error occurred in mars pathfinder in 1997. It experienced an unexpected system reset, which resulted due to a priority inversion bug while simultaneously executing different processes. In 1998, two other missions to the Mars were also failed due to software-related reasons. Mars climate orbiter burnt up due to a navigation problem, which occurred because of the

Received 04 October 2005

difference in unit of engine thrust used in the ground control system software and onboard software. The problem encountered in the Mars Polar Lander was the unexpected setting of a software variable by the touchdown sensors, resulting in premature shutdown of the descent engine.

These failures have taught the following lessons to the quality assurance community[2].

- Although software reuse is a mean to reduce the coding effort and costs, it has to be handled with utmost care, because the software, which works adequately in one context, can fail in another context.
- The redundant software package is exactly the duplicate of the prime system. The redundant system also fails to handle software design errors.
- Code optimisation may sometimes affect the correctness of the software.

2. CHALLENGES FOR SQA TEAM

The requirement for quality assurance in software-intensive systems developed at the ISRO has increased significantly over the last decade. The software quality assurance team is facing a number of challenges throughout the software development life cycle. Since quality assurance activities cover both the development phase, and the verification and validation phase, the challenges in quality software realisation are also spread over these two phases.

2.1 Development Phase Challenges

2.1.1 *Identification of Software Functions*

Navigation, guidance, and control (NGC) systems in the launch vehicle play a predominant role in achieving the objectives of a mission. The navigation, guidance and control system requirements are realised in hardware and software. Major challenge lies in the apportionment of system requirements into hardware and software functions. A joint team consisting of hardware and software experts performs detailed analysis of the system requirements and brings out a set of software functions. Quality asurance tasks in this phase ensure that all required studies are considered for decision-making, and all recommendations of the review board are implemented.

The navigation, guidance and control subsystem realised in the avionics system of an ISRO launch vehicle is the fault-tolerant subsystem. This is achieved through both hardware and software. In a particular mission, fault detection is realised in software by monitoring health status of computing elements and fault avoidance is done through hardware switchover from faulty links or packages to healthy ones.

2.1.2 *Choice of Right Algorithm*

Once the software requirements of the navigation, guidance, and control system are finalised, the next challenge is in choosing the right navigation, guidance and control algorithm from the available ones. Feasibility studies are conducted in simulation testbeds to evaluate the proposed algorithm. Reviews are conducted for selection of the right algorithm. The flight-proven algorithms are given priority.

In a mission software, algorithms like sensor-error modelling in navigation software, closed-loop guidance scheme in guidance software, control law in control software, etc, are finalised based on a number of simulations and trade-off studies. Ensuring the exhaustiveness of the algorithm studies, impact analysis, effectiveness of trade-off studies, incorporation of all review recommendations, etc, are the major quality assurance functions in this development phase.

2.1.3 *Requirements & Design*

Domain-specific requirements and interfaces must be checked to ensure the integrity of complex software systems. Today, requirements analysis is time-intensive and expensive because it is done manually. Major commercial tools are neither powerful nor customisable enough to check complicated interface rules. There arises the challenge for an intuitive approach for checking domain-specific requirements.

Proving the correctness of the developed specifications wrt the requirements is one of the most important and difficult tasks performed by the verification and validation teams of ISRO.

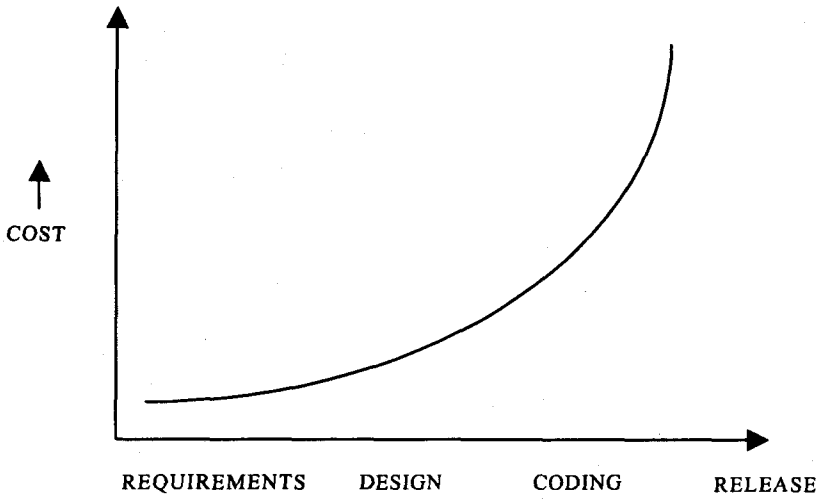


Figure 1. Cost of fixing bugs in different phases of the software development life cycle

With the validated requirements, the software development cost can be reduced to a large extent in terms of manpower and delays. Figure 1 shows a comparison of the cost of fixing bugs in different phases of the software development life cycle.

Detection of missing requirements is another difficult and challenging problem to overcome. These are not found in the specification, so these are often overlooked by reviewers during verification. The following schemes are generally adopted to trace the missing requirements in launch vehicle projects[3] to:

- Ensure that traceability is established between mission requirements and software requirements.
- Ensure that non-functional requirements such as quality attributes, performance goals, constraints, external interface requirements, have been specified.
- Represent requirements information in an alternate way (like structured text or graphical format) and establish to consistency between the two representations.
- Create a checklist of typical functional categories and to check if requirements are present in all the pertinent categories.
- Examine similar and competing applications for additional functionalities.

In navigation, guidance, and control software projects, requirements and design-level challenges are overcome through peer reviews by a team of domain experts. IEEE Std 1028:1997 defines technical review as a systematic evaluation of a software product by a team of qualified personnel that examines the suitability of the software product for its intended use and identifies discrepancies from specifications and standards[4]. Technical reviews may also provide recommendations of alternatives. In addition, availability of design guidelines, pseudo language for design, etc, help in overcoming design-level challenges in operational missions.

2.1.4 *Coding*

In the coding phase, there are a number of challenges. Foremost comes the choice of implementation language. Low-level assembly language or a high-level language like Ada is used, depending on the mission and type of the navigation, guidance and control system selected. Ada is selected as the language for the forthcoming missions, by considering a number of aspects like identified processors, tool support, efficiency of language, usage by other international space agencies, etc. The next task is interfacing of modules in the software using appropriate data structures. A quite complicated job is the

coordination of inter-task communications. Choice of data type of arithmetic variables, use of error-prone language constructs, etc, are also challenges in this process.

Adherence to coding guidelines helps both designers and quality assurance engineers to combat the coding phase challenges easily. The error-prone nature of the advanced language features gets resolved using a standardised safe subset of the language features. An Ada subset for flight applications has been defined for this purpose. Until recently, ISRO Software Engineering Standard:92 (ISES:92) was being followed in ISRO software projects. Vikram Sarabhai Space Centre Software Engineering Standard was framed in 2004 and is being adopted. This standard is a tailored version of IEEE 12207 Std. It specifies all required guidelines for software processes.

Software systems continue to suffer from symptoms of aging, as these are adapted to volatile and changing requirements. Software development process should support software evolution[5]. Formal methods have been advocated as a means to improve software development with an emphasis on software specification and verification. Currently, even if small-localised changes are made to the specification of a program, the entire program needs to be verified again. This makes the cost of verification of changes proportional to the size of the program. Formal methods need to embrace change and evolution to serve as practical tools for software engineers.

In recent ISRO projects, formal methods are being adapted to prove the correctness of synchronisation logic, error-handling logic, and timing properties of the onboard software. Design methodologies like object-oriented approach, UML-based design, etc, are also experimented in the software projects. The challenges of inducting these schemes in the software engineering process and subsequent transition to related methods, are also to be met in the forthcoming years.

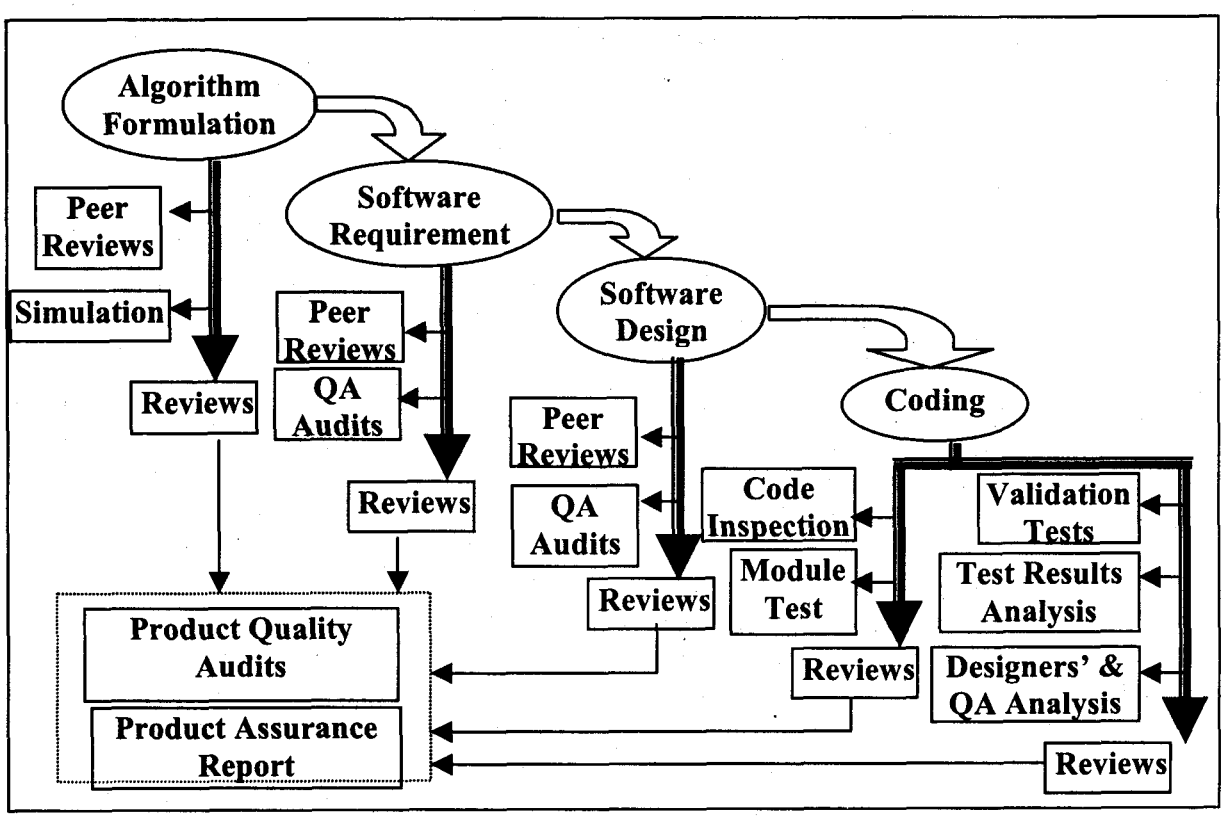


Figure 2. Verification and validation in software development life cycle for onboard software

2.2 Verification & Validation Phase Challenges

Verification and validation activities followed throughout the software development life cycle are depicted in Fig. 2.

2.2.1 *Code Inspection*

Code inspection is the visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications. The software design document (SDD) is taken as the reference for this process. On completion of the inspection, a report is prepared on the bugs found.

The observations recorded in code inspection of a particular application software in a specific mission are classified as shown in Table 1.

Table 1. Code inspection observations

Category of observations/ recommendations	No. of observations/ recommendations
Wrong code	1
Missing code	2
Extra code	9
Requirements specification error	4
SDD error	0
Violation of guidelines	1
Document related	8
Suggestion for improvement	7
Total	32

Several challenges are being faced by the teams performing code inspection[6]. A few are:

- Total reliance on expertise of the tester
- Subjection of code to human error
- Number of person-hours is proportional to the complexity of the code.

To cope up with the abovementioned threats, the process of code inspection has been augmented with automated source code analysis tools. The tool 'understand for Ada' is extensively used for static analysis before initiating code inspection. Tailor-made tools are also under development to meet the challenges of future missions. These tools are customised to incorporate user-defined rules/ guidelines.

2.2.2 *Software Testing*

Software testing is essential at the application level as functional testing. It is necessary to be carried out at the unit level. Unit testing is the lowest level of testing performed during software development, where individual units of software are tested in isolation from other parts of a program[7].

Presence of logically and computationally complex software cannot be avoided in many ISRO software projects. Complete testing coverage of such software is a major task. Especially the subtle interactions between multiple processes and different subsystems are difficult to test. Standard software testing cannot test all the combinations of paths/input variables in a program. New techniques are needed by the verification and validation teams to verify the fault tolerance of complex software systems.

The software fault injection (SFI) is a new technique developed in the last decade. The SFI input errors into the software at various locations and verifies that the program responds in an acceptable manner. This technique is used in the process of unit testing performed by the verfication and validation teams. Model-based testing is an emerging technology that can be used to detect critical software errors. This method uses state-space exploration to evaluate a number of potential program behaviours. Activities have already been initiated to implement this upcoming technology in the future missions.

Solutions for online software failures are difficult for the upcoming exploratory missions because failures are hard to pinpoint and contingencies may be too complex. Diagnostic techniques have to be experimented to identify failures. Integration of software-analysis tools and defect-tracking tools has to be established to create a link between a software feature and a failure. Then, by applying

machine learning to the results, predictors leading to software faults can be obtained.

Analysis of the most severe anomalies that occur during an operation is an important mean of improving quality of the current and the future software[8]. Mining anomaly reports serve to reuse knowledge regarding one system on the other similar systems. Anomaly analysis can explicitly warn similar vulnerabilities on the future systems. Such feed-forward references need to be captured for inclusion in inspections, reviews, and test cases of subsequent similar systems. Thus, anomaly analysis can be a valuable asset.

2.2.3 *Simulation*

Validating the system in near-flight environment is done through simulations. Modelling of vehicle characteristics in flight, like propulsion, aerodynamics, atmospheric, wind, control power plant, etc, is a great challenge in conducting simulation studies. Choice of simulation schemes is another intricate task. technical reviews aid in the formulation of the simulation models. Test results of simulations with actual hardware and software are compared with the results from simulation studies. Experience gained from previous missions plays a vital role in realising the task of simulation.

Redundancy management is an essential part of a mission-critical software. A fault-tolerant design is also mandatory in the navigation, guidance and control system. Additionally, error-handling features have to be incorporated in the onboard software to tackle the errors that may occur during the flight. Validating all these features through simulation testbeds is a tedious task. In ISRO, failure-mode studies and integrated system testing help to ease this task.

Test cases are specified for each simulation testbed to validate the system performance. These test cases should be capable of covering anomalous conditions and 3-sigma vehicle dispersions for the stress testing of the flight software[9]. Arriving at a decision about the actual number of test cases needed for complete validation is a tricky task. Complexity measures and code coverage metrics are helpful here to come out with sufficient test cases.

2.2.4 *Effectiveness of Measures Adopted to Overcome Challenges*

It is through simulations that the effectiveness of all the measures taken to overcome challenges in the software development process gets validated. Some of the challenges and the types of simulations carried out to validate the measures adopted, are given in Table 2.

Continuous success of launch vehicle missions of ISRO demonstrates the effectiveness of software quality assurance measures adopted. An elaborate post-flight analysis is performed after every launch. No software anomalies were observed in any mission except for the failed PSLV-D1 mission. Moreover, the flight path was seen to be very close to the pre-flight predicted trajectory.

3. CONCLUSION

A brief description of some of the challenges in the software development process and the present schemes to overcome these in ISRO launch vehicle

Table 2. Types of simulations for validation

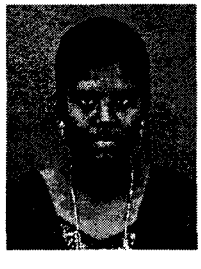
Type of challenge	Type of simulations for validation
Apportioning software & hardware functions	Integrated closed-loop simulations with sensors/actuators in loop
NGC algorithm selection	Mission simulation studies in closed-loop system simulations
Software requirements and design	closed-loop system simulations under nominal, off-nominal, and stressed conditions
Robustness and fault tolerance	Open-loop system simulations under identified test cases and failure modes
Open-loop performance of the embedded system	Performance studies in open-loop simulations testbed
Closed-loop performance of the integrated system	Performance simulations (nominal and off-nominal) in closed-loop system simulations testbed

projects is attempted in this paper. Software evolution is yet another challenge faced by the software quality assurance team. Developing and verifying safety-critical software for future ISRO missions, like reusable launch vehicles (RLVs), mission to moon (*Chandrayaan*), etc, is a big challenge for the software community. Software development teams of ISRO have already initiated efforts to tackle the challenges in the upcoming launch vehicle projects.

REFERENCES

1. Jones, Michael. Software engineering: Are we getting better at it? Mission Data Systems Division. ESA Directorate of Operations, Germany. *ESA Bulletin*, February 2005.
2. Lowry, Michael R. Software construction and analysis tools for future space missions. Computational Sciences Division, NASA Ames Research Centre, 2001.
3. Wiegers, Karl E. Peer reviews in software–a practical guide. Addison-Wesley Information Technology Series, 2002.
4. Radice, Ronald A. Software inspections. Tata McGraw Hill Publishing Co Ltd, 2003.
5. Mens, Tom. Challenges in software evolution. ECRIM-ESF Workshop ChaSE 2005.
6. Duncan, Brent. Cleanscape director, stopping bugs before they kill your software organisations. 2001.
7. Gopalan, Poofa & Uma Sankari, S.S. Software testing–a roadmap for Hi-REL software. *In* Proceedings of the National Conference on New Horizons in Computing -FICOM 05, February 2005.
8. Lutz, Robert R. & Mikulski, Ines Carman. Empirical analysis of safety-critical anomalies during opearations. *IEEE Trans. Software Engg.*, March 2004,
9. Strickland, Edward A. An object-oriented design of a launch vehicle simulator-case study–2000. Analex corporation, Colorado.

Contributors



Ms Poofa Gopalan completed her graduation in Electronics and Communication Engineering in 2002, and postgraduation in Computer Science and Information Technology in 2004, both from the Manonmaniam Sundaranar University. She joined the Quality Assurance and Reliability for Software and Mission Group at the Vikram Sarabhai Space Centre (VSSC) in 2004. Since then, she has been involved in independent software inspections and testing of embedded software for launch vehicle missions.

Ms S.S. Uma Sankari obtained her BE (Computer Science) from the Madurai Kamaraj University in 1997, and postgraduation in Computer and Information Science from the Cochin University of Science and Technology in 2002. She joined the Quality Assurance and Reliability for Software and Mission Group at the VSSC in 2002. Since then, she has been involved in independent inspections and testing of simulation and checkout software for launch vehicle missions.

Mr D. Mohan Kumar obtained his postgraduation in Mathematics from the Kerala University in 1981, and joined VSSC in the same year. He worked in system programming and mission simulations, and is currently involved in quality assurance of flight software.

Dr R. Vikraman Nair did his BE (Electronics and Communication Engg) from the Kerala University in 1971, received PhD in Information Technology from the University of Paris (France) in 1980. Since 1971, he has been involved in launch vehicle integration, design and development of checkout systems for flight operations, and quality assurance of mission software at the VSSC. Currently, he is heading the Quality Assurance and Reliability for Software and Mission Group at the VSSC.