Independent Verification and Validation of Aero Engine Propulsion System Software

Sonal Shekhawat, Arshad Iqbal, Usha Srinivasan, and Sreelal Sreedhar

Abstract With the evolving technology and extensive software usage, aircrafts have become a software embedded flying contrivance. Navigation system, landing gear system and propulsion system are some of the major subsystems of the aircraft. Propulsion system is one of the vital sub-systems with demarcated purpose to propel the aircraft. Earlier, the control unit of the engine was completely controlled by mechanical means but with the technical advancements it has been automated by software embedded control unit. Software has become so important these days that its safety, complications and risks cannot be ignored. The embedded software in digital engine control unit is a safety critical software as its failure can lead to hazardous state that can cause loss of property, damage to environment and even loss of human life. Therefore, intensive care needs to be taken while ensuring the safety and reliability of such software. The traditional testing approach needs to be fortified with more firm and rigid standardized methodology in order to diminish the probability of failure of the system. This paper throws light on the Independent Verification and Validation process followed to ensure safety, reliability and robustness of aero engine propulsion system software.

Keywords Verification and Validation · DO-178B · Traceability · Static testing · Dynamic testing

1 Introduction

Over recent years, software has gained its importance in more or less every field of engineering. Across all the disciplines including aviation systems, engineers rely on software these days for seamless interaction with the hardware. Various subsystems such as aircraft propulsion system, aircraft structural system, air data and flight instrumentation, navigation system and communication system constitute the complex machinery known as aircraft [1].

S. Shekhawat (✉) · A. Iqbal · U. Srinivasan · S. Sreedhar
Gas Turbine Research Establishment, DRDO, Bangalore, India
e-mail: sonalshekhawat209@gmail.com

G. Sivaramakrishna et al. (eds.), *Proceedings of the National Aerospace Propulsion Conference*, Lecture Notes in Mechanical Engineering,
https://doi.org/10.1007/978-981-19-2378-4_13

Aircraft propulsion system is one of the vital subsystems to propel the aircraft. Propulsion means to push or to drive forward. It works on Newton's 3rd law of motion, which states that for every action there is an equal and opposite reaction. To drive the aircraft in the preferred direction, adequate amount of thrust is required to be generated in the opposite direction which is attained by an engine.

As per the various implementations and principles followed, different kinds of engines can be used for aerospace propulsion system such as piston engine and gas turbine engine. The piston engine works on the principle of converting the pressure into rotating motion using pistons, whereas the gas turbine engine uses the pressure generated from fuel ignition to produce thrust. The gas turbine engines have got quite a few advantages over piston engine such as very high power to weight ratio and much lighter weight [2]. It follows the principle of Brayton's cycle and has got three main sections, namely compressor, combustor and turbine. Each of these sections has a well-delineated purpose such as compressing the incoming air, air/fuel mixture, ignition timing, idle speed and energy extraction from the heated air to turn the compressor [3].

Before the introduction of electronic control units, each of the above-mentioned functionality was accomplished with mechanical coupling and control. However, as the avionics system evolved, the usage of software has secured a vital footing by enhancing system's reliability and performance.

In today's aircraft design, Digital Electronic Control Unit (DECU) is designed with the embedded software to autonomously control the engine all through its operating range in both normal and off-normal conditions. In case of mechanical equipments, each operation needs to be executed mechanically by the pilot, which might result in distraction and lesser attention toward another prominent task.

During a risky situation at 35,000 feet, pilot cannot be expected to take the complete charge of the mechanical control system. For that matter, DECU decreases pilot's workload by providing self-operating, self-monitoring, redundant and failsafe setup. With all these advancements in the airborne systems technology, the aircraft has become a flying machine, controlled and monitored by complex software. Avoiding aircraft accidents and providing air safety is the need of the hour of aviation system development. In this context, the safety critical software needs to have a very high assurance of the quality with respect to system's safety, reliability and security which is ensured by Independent Verification and Validation (IV and V).

The Patriot missile system shutdown, Ariane five rocket explosion, Ethiopian Airlines Flight 302 crash and Lion Air Flight 619 crash are some of the known examples from the past where software errors have certainly led to loss of irrecoverable human life and critical missions. This paper talks about the IV and V process followed to ensure the exactitude of safety critical embedded software which in turn assures its quality, reliability and safety.

2 Why IV and V?

Undeniably, software is one of the most intricate man-made piece. Unlike hardware, software errors are not realistically visible. Software does not follow any proved laws of physics, so as to predict the type and the consequences of the error. Some bugs still remain in the software even after meticulous and rigorous testing. Verification and Validation is often mistaken as testing. Testing is only a part of V and V process and not the V and V as a whole.

The development team also tests the software before delivering it, then, what is the need of an Independent V and V activity? Developer works with constructive mind-set, whereas the verifier works with a destructive mind-set. While developer needs out-of-the-box thinking and creates lens in finding better ways to optimize the solution, testing needs patience, discipline and relentlessness in doing repetitive work.

Every human being thinks that he is right and developer is no exception to that. With a default attitude to authenticate the efforts, the developer articulates the test cases which are adequate to demonstrate the intended functionality. On the contrary, the independent verifier creates a wave of negativity to disagree with the developer's testing regarding the correctness and completeness of the envisaged objectives.

What is IV and V? IV and V stands for Independent Verification and Validation. Verification ascertains the correctness of the software in terms of the process followed to ensure the intended functionality. Validation guarantees the mapping of the software functionality to the user requirements and assures the completeness of the software [4]. As per Boehm: Verification: "*Are we building the product right?*" Validation: "*Are we building the right product?*" [5].

Moreover, an independent test resource would rule out any misunderstanding in the requirement by reviewing, analyzing and testing the software without any preconceived notion about the software, thereby enhancing the confidence in the delivered product.

3 Do-178b

To ensure the global acceptance of any product, the development process should adhere to some standard guidelines which are accepted worldwide. A standard is a commonly agreed upon technical document which is formulated to provide uniform guidelines. All the stakeholders of a specific process, product or service come together to form a standard [6].

Accomplishment of quality goals, improved software management, overcoming the schedule and budget constraints are some of the potential benefits attained by the usage of standards. There are various standards laid out for different safety critical system software such as automotive standard—ISO 26262, railway standard—CENELECEN 50,126, nuclear standard—IEC 60,880 and medical standard—IEC

Table 1 Categorization of software as per criticality levels

Criticality level	Design assurance levels	Examples
When software failure results in catastrophe events	Level A	Flight and engine control system
When software failure leads to hazardous or severe major failure conditions of aircraft	Level B	Flight management system
When software failure leads to major failure condition of the aircraft	Level C	Collection of internal built-in test equipment (BITE) faults
When software failure leads to minor failure conditions of the aircraft	Level D	Flight history, keyboard monitor command
When software failure does not hamper the operational capability of aircraft or pilot workload	Level E	Entertainment equipment

62,304. Similarly, DO-178B is an aerospace standard for airborne systems and software considerations. It categorizes the software in different levels based on the failure conditions and the criticality levels of the software as given in Table 1.

Each level of the software has its own specific set of objectives defined. The focus of this paper is toward the level A objectives mandated for aero engine propulsion system software.

4 Software Verification

Software verification is an integral process that is applied throughout the entire software life cycle. It starts in the planning phase and goes all the way through product release and even into maintenance [7]. As per the glossary of DO-178B, *"Verification is the evaluation of the results of a process to ensure correctness and consistency with respect to the inputs and standards provided to that process"*[8].

It ensures that uncovered and unidentified errors do not propagate to the next step of the activity, thereby reducing the amount of work that the developers need to redo in case the error propagates to the next level. A combination of reviews, analysis and thorough testing satisfies the objectives of software verification which will be elaborated further.

4.1 Review

Review is a holistic process which scrutinizes the activity through a different perspective and provides a qualitative assessment of the correctness [8]. It helps uncovering the defects and errors at early stage. Reviewing an activity is carried out at every stage of the software development life cycle, beginning from requirement gathering and all the way through design, development and testing. The cost incurred to rectify the defect is inversely proportional to the software development detection phase. After evaluating multiple projects, Barry Boehm [5] approximated the cost impact analysis as depicted in Fig. 1 [9].

The considerable increase in the cost encountered to detect and resolve the bugs during the requirements phase to testing phase can heftily increase the software expenditure.

The robust review process across the software development life cycle (SDLC) enhances efficiency and confidence in the product being delivered. There are various review activities which are carried out at different stages such as code review, design review and requirements review to ensure that the developed software meet its requirements and has been represented as per the agreed standards.

4.1.1 Document Review

It involves reviewing all the SDLC artifacts such as software requirements document (SRD) and software design document (SDD). Document review ensures readability, understandability, completeness and traceability with respect to its SDLC counterparts. Apart from ensuring the bidirectional traceability across SDLC, review also handles the assessment of compliance to its corresponding standards.

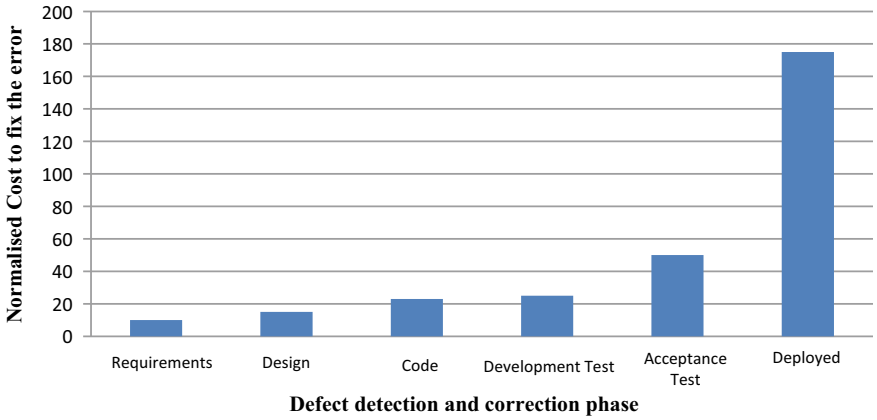

Fig. 1 Increase in cost to fix during SDLC

4.1.2 Code Review

In order to assure the completeness and accuracy of the software, the source code is reviewed with respect to the low-level requirements in code review. It makes sure that the coding standards followed in the code are in compliance with software design plan (SDP). Reviewing the code also ensures the bidirectional traceability between the low-level requirements and code. After each review, problem reports are generated by incorporating all the review results and shared with the developer team for rectification. This iterative process continues until the code becomes bug-free.

4.2 Analysis

As per DO-178B, analysis concentrates on the obtained results of the software development and software verification. It provides repeatable evidences of correctness [8]. There are various types of analysis performed during the safety critical software life cycle. Analysis not only inspects the completeness and the intended task of the software but also scrutinizes its association to the other components in the aero engine system [8].

In order to report repeatable evidences of the correctness, analysis should be perfectly documented and maintained. The procedure followed to carry out the analysis should have well-defined purpose, criteria and all the other related requirements to identify the analysis methodology and data items to be analyzed. Detailed instruction set is required to perform analysis. The artifacts generated as a result of analysis procedure are further scrutinized and corrective actions are suggested. There are various kinds of analysis which are performed as per the requirements [7]:

(1) Worst case execution analysis,
(2) Memory margin analysis,
(3) Link and memory map analysis,
(4) Load analysis,
(5) Interrupt analysis,
(6) Math analysis,
(7) Errors and warning analysis,
(8) Partitioning analysis.

4.3 Testing

4.3.1 Static Testing

Static testing is a testing technique which detects the defects without having the code executed. This testing is also known as non-executable testing as code execution is

not required for this testing. Static testing comprises of software inspection, code walkthrough and static analysis.

IEEE defines the software inspection technique as: a visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications. Inspections are peer examinations led by impartial facilitators who are trained in inspection techniques. The goal of software inspection is to detect the software flaws by scrupulous peer examination, whereas code walkthrough aims at evaluating the source code file, detecting bugs, omissions and discrepancies. The output of code walkthrough process documents the anomalies and corresponding action items [10].

Static analysis is a procedural analysis of detecting anomalies, checking code complexity and analyzing data flow in the source code. It is usually performed by using tools like data flow analyzers, rule checkers and complexity analysis tools. It analyzes the complexity of the code by computing a variety of metrics which can be used to enforce appropriate standards. Data flow analysis analyzes the action on the variables in the source files and reports any kind of problem with the usage.

4.3.2 Dynamic Testing

Dynamic testing is a testing approach which focuses on testing the dynamic behavior of the software. This testing needs the code to be executed; hence, it is termed as executable testing. Dynamic testing being implemented at a later phase of the SDLC, its associated cost impact on error addressal is relatively high. In this methodology, the source code is probed by providing various inputs to the software to ensure the reliability and robustness of the software.

4.3.3 Unit Testing

In unit testing, individual modules of the source code are tested at component level. Test Cases are designed and executed to ensure the accurate functionality of each module. It tests each module independently against the expected results as per the code and ensures that the generated results; i.e., the actual results are identical to the expected results. Hundred percent structural coverage, dead code elimination and deactivated code analysis are accomplished by means of unit testing.

4.3.4 Software–Software Integration Testing

In Software–software integration (SSI) testing, independent modules are integrated and tested for the apt functionality. It uncovers all the errors which have been introduced as a result of the integration of the modules. It aims at testing the interface between the modules. To perform the integration testing, a well laid-out plan should be in place which includes the designing of test cases, test scenarios and test scripts

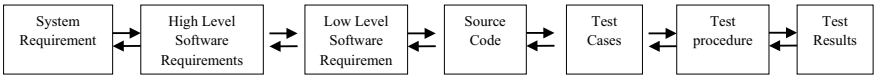

Fig. 2 Bidirectional traceability

followed by the execution of test cases. The detected defects are further traced and documented, and this continues till the source code is completely tested.

4.3.5 Hardware–Software Integration Testing

Once the application software is ready, it is imported on the hardware. Hardware–software integration (HSI) testing aims to expose the errors which occur when software is executed in the target operating environment. Different kinds of tests are executed to certain the robustness and reliability of the system. Various dynamic errors such as control loop behavior, interrupt handling, timing, memory faults are uncovered during HSI testing [7].

5 Traceability

As per DO-178B, traceability is the evidence of the association between items, such as between process outputs, between an output and its originating process or between a requirement and its implementation [8].

Establishing traceability across SDLC is crucial in order to guarantee that no unwanted functionality is introduced in the code. Traceability ensures that every low-level software requirement is traceable to some system requirement. Traceability verification at each stage starting from system requirements to test results assures that only the intended requirements have been implemented. Bidirectional traceability as shown in Fig. 2 assures that no unwanted task has been coded in the software. Traceability, once established, gives the confidence that each and every requirement specified in the SRD has been implemented in the source code. If some changes take place, traceability helps to detect impacted modules or data that need to be verified again.

6 Conclusion

The invention of aircraft with demarcated subsystems, such as landing gear, propulsion system, navigation system and telemetry system, is an extremely intricate research that humans have ever attempted. The convoluted functionality of the aircraft propulsion system depends on the complex gas turbine engine controlled by crucial

software embedded DECU which is the concealed brain behind the safe travels. Ensuring the quality, reliability, impregnability and security of such safety critical software is mandatory. Propulsion system, being one of the most intricate functionalities of the aircraft system, contributes to the criticality and complexity of the embedded software. In this software reliant era, performing IV and V activity has become a must to muddle through the exponential increase in the complexity and criticality of the source code and rapid technology changes to accommodate the capability enhancement. Would any one risk the human life to a complex machine after knowing that it could have been subjected to another perspective before flying but hadn't? [11].

IV and V is that another perspective which brings the destructive view into the picture to track down the design shortcomings and code bugs by diligently reviewing and analyzing the results. IV and V gives the confidence about the correctness of the software being deployed and makes sure that the final software is meeting user requirements [12]. It ensures that the software is reliable and serves no extra functionality, thereby eliminating the risk of failure. It is an efficacious risk alleviation strategy that effectively exposes the faults and looks out for the opportunities for improvement throughout the SDLC [13].

IV and V also results in significant reduction in overall cost of the project savings by augmenting the odds of exposing the high-risk errors early in SDLC [14].

An oversight is an open invitation to safety risks, vulnerable data, security issues and fatal errors that are sufficient enough to break the system down causing financial loss, mission failure, loss of irrecoverable human life and atrocious destruction. A well-defined IV and V when in place serves as a mitigation strategy to avoid these disastrous scenarios and to confidently deploy the safety critical software in the airborne systems.

Acknowledgements The author expresses the gratitude to Director GTRE for his consent to demonstrate this work. The author would like to thank Mrs. Pratibha Menon Sc "D," for her continuous moral support and encouragement in pursuing this activity.

References

1. http://assets.press.princeton.edu/chapters/s9497.pdf
2. https://www.differencebetween.com/difference-between-gas-turbine-engine-and-vs-reciprocating-engine-piston-engine/
3. https://www.cast-safety.org/pdf/3_engine_fundamentals.pdf
4. Pressman RS, Software engineering, a Practitioner's approach
5. Boehm B (1981) Software engineering economics. Prentice Hall, pp 463–365
6. https://www.cen.eu/work/endev/whatisen/pages/default.aspx
7. Rierson L, Developing safety-critical software: a practical guide for aviation software and DO-178C
8. DO-178B Document/RTCA DO-178B (EUROCAE ED-12B)
9. https://pdfs.semanticscholar.org/d236/2d97f419c29c5ad78df4f79d4e7061c19155.pdf
10. Khurana H (2016) Software testing. Pearson India Education Services Pvt. Ltd., pp 14–24

11. Lewis RO (1992) Independent verification and validation—a lifecycle engineering process for quality software. A—Wiley Interscience Publication, pp 3–10
12. https://www.nasa.gov/centers/ivv/services/whativv.html
13. https://panorama-consulting.com/wp-content/uploads/2016/07/Independent-Validation-Verification.pdf
14. https://www.belmero.com/2017/06/the-benefits-of-ivv/