American Journal of Science, Engineering and Technology
2021; 6(2): 27-33
http://www.sciencepublishinggroup.com/j/ajset
doi: 10.11648/j.ajset.20210602.13
ISSN: 2578-8345 (Print); ISSN: 2578-8353 (Online)

A Software Verification Approach That Complies with DO-178B Certification Rules on UAV's Flight Control Computer

Oğuzhan Demir[*], İbrahim Seyfullah Babaarslan

Turkish Aerospace, Ankara, Turkey

Email address:

ioabolude@gmail.com (İ. O. Abolude)

[*]Corresponding author

To cite this article:

Oğuzhan Demir, İbrahim Seyfullah Babaarslan. A Software Verification Approach That Complies with DO-178B Certification Rules on UAV's Flight Control Computer. *American Journal of Science, Engineering and Technology*. Vol. 6, No. 2, 2021, pp. 27-33.
doi: 10.11648/j.ajset.20210602.13

Received: June 12, 2020; **Accepted**: July 24, 2020; **Published**: June 9, 2021

Abstract: In this paper, the verification approach developed in accordance with the DO-178B certification requirements of the software of the Unmanned Aerial Vehicle's (UAV) Flight Control Computer (FCC) and the lessons learned from this approach are presented. The software verification process is a process that is used to verify how the aircraft's flight control computer behaves according to specified requirements and is used to verify that it does not produce unexpected results. The paper will first describe the software architecture, and then the types of tests developed in accordance with the software architecture. Then, test levels will be compared according to different testing parameters. Afterwards, the information regarding the management of test cases will be reviewed in detail with their different scenarios. The traceability controls and the importance of using traceability while writing the test cases and how to blend a traceability inside a test case will be explained. The studies on structural coverage analysis will be covered in a different section. This whole process can be made automated. To help automate the process, various tools are used. These tools also need to be tested, meaning they need to be qualified. Section 8 talks about this. Finally, lessons learned from the DO-178B certification process will be presented at the end of the paper.

Keywords: Do-178B, Software Verification, Software Testing, Unmanned Aerial Vehicle

1. Introduction

The level of development of software and complex hardware that must be complied with according to STANAG 4671 applicable to Fixed Wing Military UAVs is given in Table 1: STANAG 4671 Development Assurance Level Targets [1]. According to the SAE-ARP-4761 safety analysis that references the article "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment" in of related software and units, the software is assigned a target development level [2]. Flight Control Computer Software is in the "Catastrophic" category according to the safety analysis and therefore needs to demonstrate the highest level of Development DAL-B (Assurance Level-B) compliance for UAVs [3]. DO-178B standard describes the actions required in case an error happens in the system. The levels explained in this standard are classified regarding errors' casualties. It can vary between causing a loss of life and not affecting anything or anybody [4].

Table 1. *STANAG 4671 Target Development Assurance Levels.*

Setting a Development Assurance Level for each and every System and Architecture		**Necessary Development Assurance Level**
Level of Importance	Catastrophic	DAL B
	Hazardous	DAL C
	Major	DAL D
	Minor	DAL E
	No Effect	DAL E

The requirements needed for each level are stated in Table 2.

Table 2. *DO-178B Requirement Table According to the Importance Level.*

DO-178 Feature	Level A	Level B	Level C	Level D
Independence Level	High	Mid	Low	Very Low
Software Plans	Yes	Yes	Yes	Yes
Software Standards	Yes	Yes	Yes	No
Structural Statement Coverage	Yes	Yes	Yes	No
Structural Decision Coverage	Yes	Yes	No	No
Modified Condition Decision Structural Coverage	Yes	No	No	No
Verifiable High Level Requirement	Yes	Yes	Yes	No
Verifiable Low Level Requirement / Code	Yes	Yes	No	No
High Level- Low Level and Low Level-Code Traceability	Yes	Yes	Yes	No
Low Level Requirement Test Coverage	Yes	Yes	Yes	No
Code Review	Yes	Yes	Yes	No
Configuration Management	High	High	Mid	Low
Software Quality Assurance Transition Criteria	Yes	Yes	Yes	No
Architecture, Algorithm Verification	Yes	Yes	Yes	No

As the criticality level of a software increases, so do the detailed documentation, tests and analysis of the software [5]. As a result, the cost increases. A more controlled structure has been established and higher quality products have emerged with the involvement of independence and reviews [19]. Although DO-178B Independence Level criterion is requested as Medium according to the Table of Requirements according to Critical Levels given in Table 2, this level is provided as “High” for the UAV we developed [6]. Having completely separate teams of developers and testers enabled cross-validation of changes by both of those teams. The criterion for software plans to be in process is provided by the fact that the software plan documents of all processes in the software development life cycle are present and accessible to all. These are; plan for software Aspects of Certification (PSAC), software quality assurance plan, software configuration management plan, software development plan and software verification plan. Software requirements standard, software design standard, software coding standard, etc. are examples to software standards. For structural coverage analysis, “Structural Decision Coverage” is used as the criterion which is sufficient for Level B. The software includes verifiable high-level requirements and low-level requirements associated with these requirements. Traceability is established to bond high level requirements with low level requirements and to bond low level requirements with code. Test cases also include traceability for the requirement that they verify [16]. After the codes are developed, they are reviewed by a person other than the developer according to the code development control table. All software products (code, test, plan documents, requirements, etc.) are kept in configuration management tools [7]. To prevent unauthorized access, modify, read, etc. for each product, authorizations are defined. Architecture and algorithm verifications are provided by integrated and partition based tests and code reviews.

The software architecture of the UAV Flight Control Computer will be explained in this paper. Then the partition based and integrated tests of aircraft software will be mentioned. The comparison, use and the contribution of the partition based and integrated test methods applied in the software verification process will be given. Next, the method of managing a test case that is developed or updated for the first time will be described. We will see how the defect and change requests are managed. Structural coverage analysis studies which is one of the most important efficiency parameters of software test case will be explained. We will talk about the qualification process of the tools used to help verify the software. Finally, the lessons gained from the DO-178B certification process will be presented [8, 9].

2. Software Architecture

First, confirm that you have the correct template for your paper size. This template has been tailored for output on the A4 paper size. Flight Control Computer Software (FCCS) is a software designed to work on real-time operating systems. The software has partitions that are programmed to do specific tasks and those partitions work on definite time interval and on a specific address. The FCCS consists of the Application Layer, where the functional functions are managed, the Input / Output Layer, which enables communication with other equipment / subsystems in the Aircraft and the Control Station, and the Management Layer, which monitors the health information and operational performance of the layers. The illustration of the software architecture is given in Figure 1.

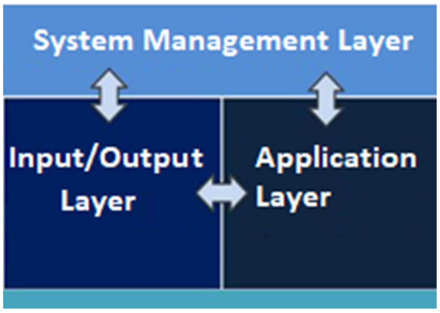

Figure 1. *Flight Control Computer Software Architecture.*

Each layer is divided into different sub-layers according to its function. In the Application Layer, codes are generated using model based software development tool. Input / Output and Management Layer is coded manually.

3. Methods

3.1. Tests

3.1.1. Partition-Based Tests

For each partition, there are software requirements that describe the function of that partition. These software requirements are written only in terms of the input and output of that partition, ignoring the rest of the partitions. Partition-based tests, just like their requirements, are based on just checking the inputs and outputs of that partition, ignoring partitions outside of that partition. For this reason, a test partition is used in partition based tests. With the help of the test partition, inputs are provided to the shared memory areas of the tested partition, and the outputs generated by the tested partition in response to these inputs are controlled [10]. The partition-based test approach is shown in Figure 2.

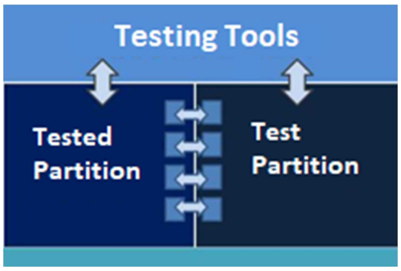

Figure 2. *Partition-Based Test Approach.*

3.1.2. Integration Tests

In addition to the requirements written on the basis of the partition, there are requirements written on an integrated level that will allow all partitions to work together [11]. Together with the testing of these requirements, it is verified that all partitions work together correctly and produce the expected results. In integration tests, inputs are provided to the software by using test and simulation tools, and the outputs produced by the software are collected by the same tools and passed / failed decisions are made. An automation tool developed by the software verification team [17] is used to provide input to the simulation tool to work with the integrated software and to compare the outputs. The integration test approach is shown in Figure 3.

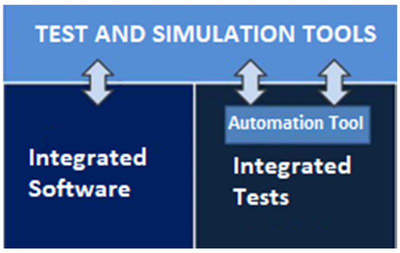

Figure 3. *Integration Test Approach.*

3.1.3. Comparison Between Partition-Based Tests and Integration Tests

In the developed test environments, the tests of the partitions that provide modularity to the software development environment and the integrated tests that verify the functionality of all partitions working together are performed. When a test case scenario development time is compared, it is seen that the development time of integrated level tests is less than that of partition based tests. This is due to the fact that the requirements that can be explained in each partition are explained at only a single level. Similarly, there are fewer requirements and number of test steps in integrated tests than partition-based tests, so updating time and effort is shorter. Since partition based tests, simulation environments, hardware cards in real environment, etc. are not affected by external factors, it is more convenient to develop automated test cases, therefore partition based tests usually run faster than integrated tests.

Since partition-based tests are designed for a specific unit under test, the error found directly focuses on the subunit tested, other subunits associated with that unit, and on the partition where the tests are run. This makes it easier to analyze the error and to find out which part of the software has it. In integrated tests, in case of an error, from which partition is the error originated is hard to tell. It can also be the simulation tool that causes the error. It is a process that takes a long time to analyze whether environmental problems cause this error. In this context, it can be said that partition-based tests are better than integrated tests to focus on an error.

Table 3. *Advantages and Disadvantages of Different Tests.*

Partition Based Tests
Advantages;
-Easy to automate
-Takes less time to run thanks to automation.
-Easy to find the source of a defect
Disadvantages;
- Takes more time to develop
- Verifies each partition, although it does not validate the intended functions of each partition.
-Takes more time to update and to maintain
Integration Tests
Advantages;
- Takes less time to develop
- Helps figure out design errors because it is closer to the user requirement level.
-Takes less time to update less effort to maintain
Disadvantages;
- Hard to automate
- Takes more time to run the test cases because of lesser automation.
- Takes deep analysis to find the source of a defect

Integrated tests are more successful in locating discernible errors in partition-based design because they are closer to user requirements. Our system includes both partition-based tests of partition-based requirements that explain the functionality of each partition, as well as integrated tests of the integrated requirements, where all partitions work together. With partition-based tests, the components that make up the software are tested one by one and at the level of detail and are free from hidden errors. In integrated tests, it is checked that matured components work as expected as a whole by passing through partition based tests. The findings obtained from the tests are summarized in Table 3 to show the advantages and disadvantages of each approach.

3.2 Test Case Management

The requirements written by the software development team at the integrated level that describe the behavior of a partition or all software partitions working together are input to the software testing process. The test case is developed according to the requirement level and the developed test cases are inputs to the peer review activity and are reviewed according to the test case checklists [14]. After the comments given to the relevant test case are re solved, the test case is released and the change must be made with the software change request at every subsequent update. The test report is created by providing the released test case in the target environment, and after the analysis of the report, software change requests related to the code, requirement or test cases are opened and the defect / request management tool is followed [15]. The steps for managing a test case developed for the first time are shown in Figure 4.

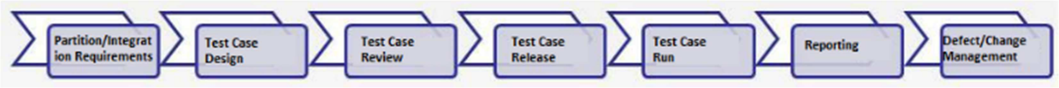

Figure 4. *The steps for managing a test case developed for the first time.*

The software change request is opened for the test cases that need to be changed as a result of an update to the software requirements and the defect / request management tool enables the change to be checked by someone other than the person who made the change. The steps involved in managing a released test case are shown in Figure 5.

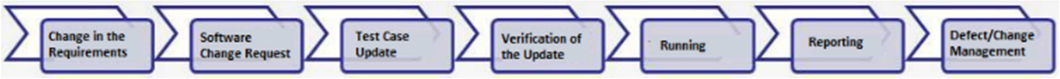

Figure 5. *Managing a Released Test Case.*

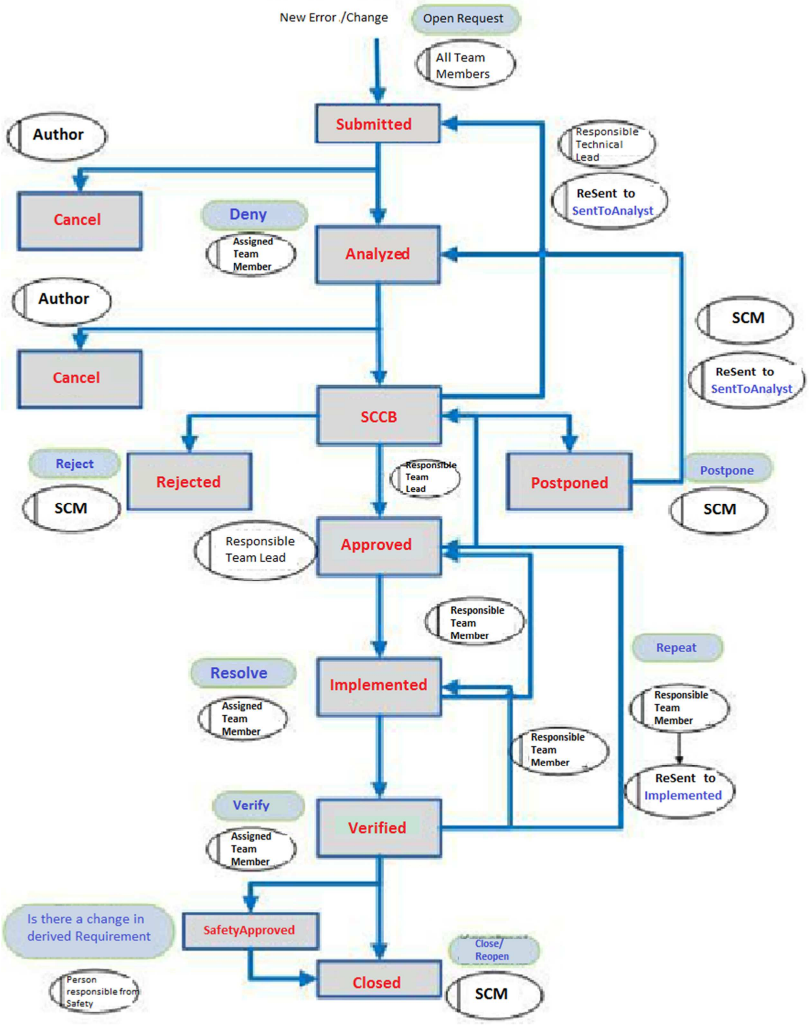

Figure 6. *Change / Defect Life Cycle.*

3.3. Change and Defect Management

As the system requirements change, the software requirements and code that are written to the corresponding system requirements also change [16, 20]. As software requirements change, there arises a need to update software test cases. When a change is required to any part of the software (code, test case, requirement, etc.), a software change request is created for that part. The software change request is analyzed by the person who will make the change after it is created and sent to the software configuration control board (SCCB) or cancellation decision is taken. The cancellation decision is usually made if the person that requested a change opens an incorrect request or reopens a previously opened request. The necessity of making changes in the software configuration control board, what it is based on and issues alike are evaluated and decided to accept, reject or postpone the amendment. Decisions to be accepted, rejected or postponed are made by the representatives of each team in the SCCB by evaluating the cost, risk and effects of the change on the project calendar. When the postponed change request is due, it is analyzed again and sent to the SCCB. A change request accepted in the SCCB is processed by the responsible person and assigned to a person other than the person who has performed this process for verification. After checking that the change is made correctly, the process is checked by the software quality assurance managers and the request is closed. In order to control changes made to a released product, the modification is only permitted if there is an accepted change request on the product. This control is provided by the integration of the configuration management tool and the change / error management tool. If the change request causes a change in the derived requirements, the change is assigned to the safety team for safety assessment. Figure 6 illustrates the situations in which the change or defect has gone through its life cycle.

3.4. Traceability Control

An external traceability tool is used to check that all of the software requirements are tested, to detect incorrect traceability between test cases and software requirements, to track the test case that is linked to the requirement that it verifies and vice versa. Software requirements and tests are added to this tool to establish desired traceability. In the event that software requirements are not covered or incorrect traceability is established, a software change request is opened [12]. This is done to add a test case for the relevant software requirement or to correct incorrect traceability. The structure of the traceability control is shown in Figure 7.

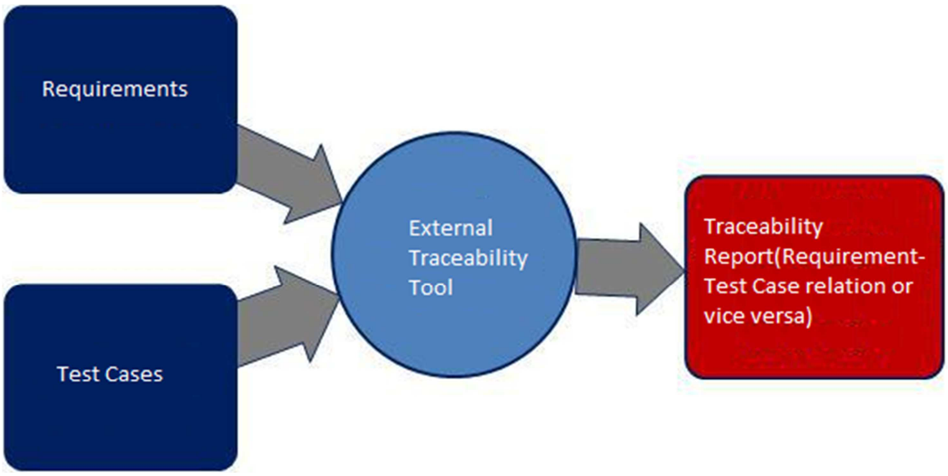
Requirements
Test Cases
External Traceability Tool
Traceability Report(Requirement-Test Case relation or vice versa)

Figure 7. *Traceability Control.*

3.5. Structural Coverage Analysis

One of the most important measures of the effectiveness of software test cases is the results of structural coverage analysis [13]. All software tests are run on the instrumented version of the code with an external structural coverage analysis tool. Decision coverage analysis is used for software tests. The report produced by the external structural coverage analysis tool is examined and the lines of code that are not covered during the running of the software tests are determined and these lines are analyzed. The illustration for generating the structural coverage report is shown in Figure 8.

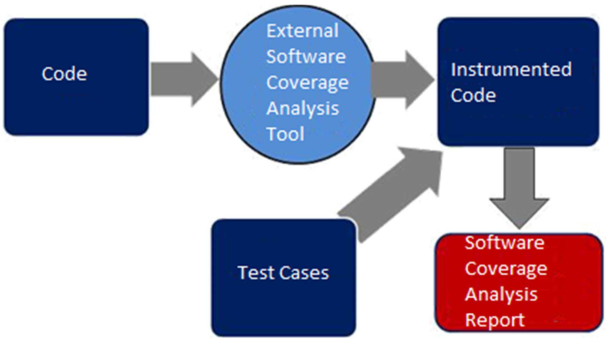
Code
External Software Coverage Analysis Tool
Instrumented Code
Test Cases
Software Coverage Analysis Report

Figure 8. *Structural Coverage Analysis.*

If the functionality of the uncovered lines is not specified by the software requirements, this indicates a lack of software requirements and a software change request is opened for the software requirements. If the functionality of the uncovered lines is specified in the software requirements but there is no test to test these requirements, it indicates a deficiency in the software tests causing a software change request to be opened to test out the relevant requirement. Parts that are never likely to run in the code are called dead code, and a software change request is opened to the code for removal. Tests, which are considered to be affected by the removal of the relevant code piece, may be repeated. In addition to these situations, inactive code, defensive code and code that can be covered by analysis / inspection are also interpreted and reported. Inactive code is the code that will be activated when the software is configured to do so. Defensive code is a code piece that prevents the system from crashing, is written for protection and does not reflect the actual functionality. The code that can be covered by analysis / inspection is the code that is tested by manual analysis scenarios in software modules with complex algorithms. Inactive code, defensive code and the code that can be included in the analysis / inspection are reported together with their reasons and left untouched. When there are no lines of code that are not covered or unexplained after all change requests have been resolved, the structural coverage analysis ends for the software version in which it was made.[18]

3.6. Tool Qualification

Although an error in the tools used for software verification does not inject an error into the software, it may prevent an error in the software from being found [6]. Therefore, it is necessary to check that any auxiliary tool used for software verification is working correctly. This process is called tool qualification. In this process, the requirements of the tool to be used to verify the software are written and tested. Errors are reported and fixes are made. At the end of this process, it is ensured that the tool used to verify the software works as expected and is considered by the authority as a tool that can be used for software verification.

4. Result

In this paper it is aimed to give an inside look of how a Critical System that complies with DO-178B should be verified. Different methods that are used in the verification process are mentioned in the order of relevance and application in the Methods section. These methods, if applied in the suggested order, would give a strict development and verification process that follows the specifications of a standard, that is DO-178. These application of methods and the development and verification process can be tightened or relaxed given the severity of the aircraft, resulting in a different standard compliance. Our aircraft is designed to fit the DO-178B.

Not only the process of development and verification, but also software development and verification team need to follow each and every instruction without skipping any steps in between. This would not guarantee but could drastically reduce the risk of losing an aircraft, or even giving casualties. Errors in software world are inevitable, but making them less serious and reducing their number is possible.

5. Discussion and Conclusion

Although the Independence Level criterion on the Table of Requirements according to DO-178B Critical Levels Table 2 is requested as “Medium” this level is provided as “High” for our UAV, so that errors can be detected earlier and an independent perspective can be provided while reviewing the products of other individuals [22]. Thanks to the complete separation of development and verification teams, cross-validation of changes is achieved by both developers and validators. The criterion for software plans to be in process is provided by the fact that the software plan documents of all processes in the software development life cycle are written before the start of the development and are accessible to all. These plans are; software certification liaison process plan for managing the process between software certification authority and project managers, software quality assurance plan for how to check the compliance of software processes according to plans, software configuration management plan for how to manage each configuration part and software development plan for how to develop software and lastly software verification plan for software verification activities. Thanks to these plans, it was possible for each newcomer joining to the team to learn the workflow and a standardization was provided to them on how each process would proceed. Many of the software development, configuration and verification standards have helped to prevent a mistake, and a certain standardization has been achieved. For structural coverage analysis, “Decision Structural Coverage Criterion” is used as the criterion that seems sufficient for Level B. This ensures that all software requirements are covered by test cases and actions are taken to correct any inconvenience. The software includes verifiable high-level requirements and verifiable low-level requirements associated with these requirements [21]. The traceability of the higher level requirements with the lower level requirements as well as the lower level requirements with the code has been established, thus checking that the expected functionality in the software flows to the lower level requirements correctly. Traceability of lower-level requirements was established through comments in the tests to verify that each requirement was tested. After the codes are developed, they are reviewed by a person other than the developer according to the code development control table. All software-related products (code, test, plan documents, requirements, etc.) are stored in configuration management tools for each product to prevent unauthorized access. Authorization conditions are defined for each individual. The transition criteria required for the initiation and completion of a process are defined and applied in order to ensure the continuation of the interconnected processes. Architecture and algorithm verifications are provided by integrated and partition-level tests and code reviews.

References

[1] NATO, STANAG 4671 Unmanned Aircraft Systems Airworthiness Requirements (USAR) (2017).

[2] SAE, ARP 4761 - Guidelines and Methods for conducting the Safety Assessment Process on Civil Airborne Systems and Equipment (1996).

[3] RTCA, DO-178B, Software Considerations in Airborne Systems ans Equipment Certification (1992).

[4] Software Testing Material Site, https://www.softwaretestingmaterial.com/integrationtesting/#What-is-Bottom-Up-Approach, last accessed 10/08/2019.

[5] Herrmann, D. S. (1999). Software safety and reliability: Techniques, approaches, and standards of key industrial sectors. Los Alamitos, CA: IEEE Computer Society.

[6] Alspaugh, T. A., S. R. Faulk, K. H. Britton, R. A. Parker, D. L. Parnas, and J. E. Shore 1992 "Software requirements for the A-7E aircraft," Tech. Rep. NRL/FR/5546-92- 9194, Naval Research Lab., Washington DC.

[7] Busser, R. D., M. R. Blackburn, A. M. Nauman 2001 Automated Model Analysis and Test Generation for Flight Guidance Mode Logic, Digital Avionics System Conference.

[8] Hayhurst, Kelly J., C. Michael Holloway, Cheryl A. Dorsey, John C. Knight, Nancy G. Leveson, G. Frank McCormick, and Jeffery C. Yang 1998 Streamlining Software Aspects of Certification: Technical Team Report on the First Industry Workshop. NASA/TM-1998-207648, April.

[9] Johnson, Leslie A. (Schad) 1998 DO-178B,"Software Considerations in Airborne Systems and Equipment Certification" STSC Crosstalk, October. http://www.stsc.hill.af.mil/crosstalk/1998/10/.

[10] Salmon, David 1993 Assemblers And Loaders, Ellis Horwood Ltd (Ellis Horwood Series in Computers and Their Applications) Market Cross House, Cooper Street, Chichester, PO19 1EB, West Sussex, UK. ISBN 0130525642.

[11] Rierson, L. (2013). Developing Safety-Critical Software. Abingdon, United Kingdom: Taylor & Francis.

[12] Patel, Parnasi & Bhatt, Chintan. (2019). Structural Coverage Analysis Methods. 10.4018/978-1-5225-7455-2.ch002.

[13] H. D. Desai, "Test Case Management System (TCMS)," 1994 IEEE GLOBECOM. Communications: The Global Bridge, San Francisco, CA, USA, 1994, pp. 1581-1585 vol. 3, doi: 10.1109/GLOCOM.1994.513041.

[14] Mandava, R. B., & Arcand, J. F. (2007). U.S. Patent No. 7, 203, 928. Washington, DC: U.S. Patent and Trademark Office.

[15] Defect Management Process: How to Manage a Defect Effectively. (2020, April 16). Retrieved from https://www.softwaretestinghelp.com/defect-management-process/.

[16] Jacklin, Stephen & Lowry, Michael & Schumann, Johann & Gupta, Pramod & Bosworth, John & Zavala, Eddie & Kelly, John & Hayhurst, Kelly & Belcastro, Celeste & Belcastro, Christine. (2004). Verification Validation and Certification Challenges for Adaptive Flight-Critical Control System Software. Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference. 4. 10.2514/6.2004-5258.

[17] Escudero, César & Delmas, Rémi & Bochot, Thomas & David, Matthieu & Wiels, Virginie. (2018). Automatic Generation of DO-178 Test Procedures. 10.1007/978-3-319-77935-5_27.

[18] Cook, Stephen & Haverkamp, Glenn. (2020). Challenges and Opportunities for Software Development and Verification on Military Aircraft Systems. 10.2514/6.2020-0238.

[19] Jasim, Omar & Veres, Sandor. (2020). Verification Framework for Control System Functionality of Unmanned Aerial Vehicles.

[20] Seabridge, Allan & Moir, Ian. (2019). Verification of System Requirements. 10.1002/9781119611479.ch7.

[21] O'Regan, Gerard. (2019). Verification of Safety-Critical Systems. 10.1007/978-3-030-28494-7_13.

[22] Yu, Junchong. (2020). Test Verification. 10.1007/978-981-15-2894-1_16.