SMARTBEAR

Meeting Federal Regulations for Airborne Software Certification with Code Review

A SmartBear White Paper

Introduction

The V-22 Osprey helicopter uses a tilting wing and rotor system in order to fly like an airplane and land like a helicopter. In one test flight, the hydraulic system failed just as the pilot was tilting the wing to land. A previously undiscovered error in the aircraft's control software caused it to decelerate in response to each of the pilot's eight attempts to reset the software. The aircraft had a built-in back-up system to handle such failures, but it had not been tested under those precise conditions, and the defect in the back-up system's software had not been found. The uncontrollable aircraft fell 1,600 feet (490 m) and crashed in a forest. For the second time that year, the V-22 was grounded.[1]

The Problem is Complexity

As systems become more capable, it becomes harder to test all of the ways they will be used in advance. Once you test software and fix all of the problems found, the software will always work *under the conditions for which it was tested*. The reason there are not more software tragedies is that testers have been able to exercise these systems in most of the ways they will typically be used. But all it takes is one software failure and a subsequent lawsuit to seriously damage a company's reputation.

Test-and-fix approaches are vital dynamic testing approaches. Whether performed on individual units or the entire system, these dynamic approaches share one common shortcoming: they all rely on test cases.

Test case scenarios are constructed from the same source documents that developers use, such as requirements and specification documents. These documents are much more comprehensive at defining what the finished product should do, rather than what it shouldn't do. Developers inject about 100 defects into every 1,000 lines of the code they write.[2] Many of these defects will have no impact on the test case scenarios designed for testing. Yet, they could have devastating, unforseen effects in the future.

If quality cannot be tested in, then what?

Software product quality assurance in the aerospace industry has been intertwined with process assurance for over 30 years. At NASA's Goddard Space Flight Center, software quality assurance (SQA) is assigend to the office of Systems Safety and Mission Assurance. In 1997, the SSMA office created a list of tasks to be performed at each of 6 phases of the software development life cycle, from concept and requirements gathering through design, development, testing and maintenance. Within the development phase, the very first requirement of SSMA is "perform code walk-throughs and peer reviews."[3]

DO-178C, "Software Considerations in Airborne Systems and Equipment Certification" is the title of the published document by which the certification authorities such as FAA, EASA and Transport Canada will approve all commercial software-based aerospace systems. DO-178C was published in January 2012 and replaces DO-178B which was last revised in 1992.

Chapter 6.1 defines the purpose for the software verification process. The more recent DO-178C adds the following statement about the Executable Object Code: "The Executable Object Code is robust with respect to the software requirements that it can respond correctly to abnormal inputs and conditions." Compare that to the earlier version of this statement in DO-178B: "The Executable Object Code satisfies the software requirements (that is, intended function) and provides confidence in the absence of intended functionality." This addition points out the importance of identifying and rectifying unintended consequences of the code. (DO-178B and DO-178C)

The guidelines use the terms "verification" and "validation" (also referred to as "V&V") to encompass software quality process requirements. While the terms are sometimes interchanged, validation generally refers to traditional, dynamic testing of the "by objective evidence." Verification, on the other hand, refers to confirmation by examination.

In a software development environment, software verification is confirmation that the output of a particular phase of development meets all of the input requirements for that phase. Software testing is one of several verification activities intended to confirm that the software development output meets its input requirements. Other verification activities specifically listed include:

- Walk-throughs
- Various static and dynamic analyses
- Code and document inspections
- Module level testing
- Integration testing

As Capers Jones points out, "A synergistic combination of formal inspections, static analysis and formal testing can achieve combined defect removal efficiency levels of 99%." Where tool assisted peer review stands out is in code and document inspections as well as providing a central location for reviewing test cases, plans and the results of static analysis tools.

1"Saving The Pentagon's Killer Chopper-Plane," Wired, July 2005

2 "The Software Quality Challenge," Watts S. Humphrey, CrossTalk, June 2008

3 Software Quality Assurance Testing At NASA," Linda Rosenberg, PhD, Goddard Spaceflight Center, NASA

4 Capers Jones, Combining Inspections, Static Analysis and Testing to Achieve Defect Removal Efficiency Above 95%, January 2012.

While some believe static analysis of the code is best done by automated tools, code reviews are actually more effective at finding errors than automated tools. Most forms of testing average only about 30% to 35% in defect removal efficiency levels and seldom top 50%. Formal design and code inspections, on the other hand, can acheive 95% in defect removal efficiency.[4]

There are some verification requriements that can only be satisfited by code review. "While analysis may be used to verify that all requirements are traced, only review can determine the correctness of the trace between requirements because human interpretation is required to understand the implications of any given requirement.The implications must be considered not only for the directly traced requirements but also for the untraced but applicable requirements. Human review techniques are better suited to such qualitative judgments than are analyses."[5]

The software verification process is aimed at showing the correctness of the software. It consists of requirement reviews, code reviews, analyses, and testing. Reviews are to be regularly conducted throughout the software development process to ensure that the Software Development Plan is being followed. All steps in the decomposition of high-level system requirements to object code are considered in this process. DO-178B and DO-178C require examination of the output of all processes to check for software correctness and to find errors. DO-178C (section 6) requires that:

1. The high–level software requirements are correctly and completely formed from the system requirements
2. The high-level requirements are complete and consistent
3. The software architecture correctly and completely meets all high-level requirements
4. The low-level requirements correctly and completely fulfill the software architecture
5. The low- level requirements are consistent and correct
6. The software code correctly satisfies all low-level requirements
7. All code is traceable to one or more low-level requirements
8. The object code correctly implements the software on the target computer, and it is traceable and complies with all low-level and high-level requirements[6]

The guidance is clear that all code reviews must be in writing. Otherwise, there is no proof it has been performed. Statements about the code in general, specific lines, and specific issues, must all be tied to the person, time and date of their identification. If needed, this data should be presented as both comments and metrics to allow an accounting of the development process.

Firms may perform, manage and document the process manually, as long as they use "appropriate controls to ensure consistency and independence." Source code evaluations should be extended to verification of internal linkages between modules and layers (horizontal and vertical interfaces), and compliance with their design specifications. Documentation of the procedures used and the results of source code evaluations should be maintained as part of design verification.[7]

The DO-178C does not go into detail as to how code reviews and evaluations should be performed. While thousands of organizations have successfully implemented and defended peer code reviews successfully, many have failed. The difference most often comes down to poor implementation strategies that can be readily addressed:

- Reviews are too long. After just a few hours, attention wanders and effectiveness decreases. All-day code reviews can seem almost painful. Keep reviews short, no more than one or two hours per day. In that time, developers will be able to review between 150 and 300 lines of code, depending on complexity. Not surprising, this rate of review also provides the highest rate of defects identified per line of code (defects / LOC).
- Reviews are seen as an additional task. It is especially true when a review backlog builds up. Rather than let them become a bottleneck, make reviews a daily activity or take them as they come in. Let the artifact review serve as a break from a hard problem or a way to transition between tasks
- Comments are seen as subjective. It is easy to discount a colleague's comments as just their opinion. Make it easy for reviewers to annotate the specific code or file in question and to get other reviewers to weigh in.
- Remote reviews can be challenging. Distributed teams are a given, and bringing teams together for reviews is at odds with the need for regular, brief reviews. Instead, facilitate remote reviews with tools designed for remote collaboration in general and peer code review, specifically.
- Documentation is not automated. The administrative burden of documenting, archiving and distributing this living document can be overwhelming. Use tools that make compliance documentation an automatic by-product of the review.

One of the most important contributions a company can make to successful adoption of peer reviews are the tools it provides its teams. The right tool set will enable each development team to

4 "Measuring Defect Potentials And Defect Removal Efficiency," Capers Jones, CrossTalk, June 2008
5"An Analysis of Current Guidance in the Certification of Airborne Software", Ryan Erwin Berk, MIT, 2009
6 Certification of Safety-Critical Software Under DO-178C and DO-278A, Stephen A. Jacklin, NASA Ames Research Center

7 General Principles of Software Validation; Final Guidance for Industry and FDA Staff, January 11, 2002

find its own best way to do code reviews, enabling a bottom-up approach to code review design and ensuring fuller achievement of potential gains and regulatory compliance.

Some characteristics of a code review tool set to look for include:

1. Supports team-designed rules and processes. Teams should be able to determine review intervals, workflows and specific tasks to be accomplished during the review while the tool supports and manages adherence.
2. Supports each team's preferred mode of interaction. Whether side-by-side, remote real-time or asynchronous, or a combination, the team should decide. The tool should support before and after views of code and document changes and threaded contextual chat with references to files and line numbers.
3. Provides seamless integration with SCM systems. To start reviews easily and expedite them, developers should be able to point to the code that needs review and have those files extracted automatically. Tools add tangible value to this process by automating the collection and distribution of these files.
4. Ensures that documents are integrated within the review process. A standardized peer review process enables all project-related documents (e.g. PDF, MS Office, HTML, images, schematics, intranet and web-based document management system) to be reviewed the same way, making document reviews less frustrating for developers.
5. Enables accurate reporting. Meaningful metrics play a critical role in the reporting process to indicate progress and current status. Useful metrics used in meeting review milestones and audit requirements include man-hours spent in review, defect data, and lines of code inspected, as well as review approval and electronic signature status.

It should be noted that this paper has steered away from discussing any particular software development methodology. A peer code review process can be implemented within waterfall, Agile and other methodologies with equal success. The point to focus on is that not only will implementing peer code and docuemnt reviews make the products your company produces better, it will make the processes and the people that produce them better as well.

Peer reviews are a powerful tool for eliminating defects, but achieving compliance can be burdensome. Even in organizations where peer reviews have been "adopted," they are skipped as much as 30% of the time, primarily because they are inadequately supported.[8]

Too often, organizations believe they can have ad-hoc development processes, and then use an inspection process at the end to remove all defects. It just will not happen. Industry statistics indicate that for every four errors pulled out, one new error is injected. Therefore, only portions of defects are actually removed if the attempt is applied only to the end of the implementation process. To approach zero defects, inspection must be an iterative process.[9]

For years, it was believed that the value of inspections is in finding and fixing defects. However in examining code inspection data, it becomes clear that inspections are beneficial for an additional reason. They make the code easier to understand and change. An analysis of data from a recent code inspection experiment shows that 60% of all issues raised in the code inspections are not problems that could have been uncovered by latter phases of testing or field usage because they have little or nothing to do with the visible execution behavior of the software. Rather they improve the maintainability of the code by making the code conform to coding standards, minimizing redundancies, improving language proficiency, improving safety and portability, and raising the quality of the documentation — benefits which are not possible from automated testing.[10]

Conclusion

Peer reviews create an environment of shared understanding and collaboration. As teams review and comment on each other's files, whether in real-time or asynchronously, they all get better. In the end, the peer review provides a platform for continuous process improvement, leading to improved standards, better developers, better efficiency, a higher quality finished product, and the peace of mind that comes from knowing the organization can prove compliance.

9 Quality ProcessesYield Quality Products," Thomas D. Neff, CrossTalk, June 2008

10 "Does the Modern Code Review Have Value?" H. Siy, Software Maintenance 2009

8 Mario Bernhart, Andreas Mauczka, Thomas Grechenig Research Group for Industrial Software (INSO) Vienna University of Technology, Austria, 2010

SMARTBEAR

About SmartBear Software

SmartBear is the choice of more than four million software professionals and over 25,000 organizations in 90 countries that use its products to build and deliver the world's best software applications. SmartBear's user-centric application management solutions support key software delivery processes of development, testing, API readiness, and application performance management across desktop, web, and mobile platforms. Get started at www.smartbear.com.

SmartBear Software 450 Artisan Way, Somerville, MA 02145 +1 617.684.2600
www.smartbear.com

WP-COL-071113-WEB