SMARTBEAR Collaborator

Whitepaper

Tackling ISO 26262 with Collaborative Code and Document Reviews

SMARTBEAR

One death, two injuries, and 1.25 million pickup trucks recalled. Just this past May, Fiat Chrysler had to face the tragic ramifications of an error in their software. In rollovers, the side air bags and seat belt pretensioners in a number of their Ram pickups were malfunctioning. This problem only came to light after a suit was filed at the end of 2016, regarding the failed air bag deployment in the rollover crash of a 2014 Ram 1500. [1.1] With the recall already underway, the focus turns to the process at Fiat Chrysler and the other automotive companies involved.

Last September, General Motors also had to issue a recall of nearly 4.3 million vehicles because of software defects with their air bags. [1.2] They only discovered the issue after a crash killed one person and injured three. Were these crashes that spurred massive recalls the kind of scenarios that should have reasonably been anticipated and tested? Probably, but even if they absolutely were, a disconnect between their code review and testing processes was evident and dangerous.

The Role of Safety Standards: ISO 26262 and Automotive SPICE

Of course, functional safety standards exist to attempt to prevent these kinds of accidents. Automotive Original Equipment Manufacturers (OEMs) in the US have to comply with industry standards like ISO 26262 by incorporating quality management features throughout their product development process. Other companies, depending on their location, may also need to include considerations for the similar but distinct standard, Automotive SPICE®. These standards have quality and risk management expectations for each step in the development lifecycle, from initial product design through software development and production. The initial version of ISO 26262 was published in 2011; and next year, edition 2 of ISO 26262 will be released. [1.3] In order to streamline some compliance considerations, it is being reported that edition 2 will group process-related requirements into one section, require a communication channel between functional safety and related disciplines, and expand its scope to cover motorcycles, trucks & buses, autonomous systems, and semiconductors.

Because the standards focus so much on risk assessment and quality management, what the part is doesn't matter as much as how the part is developed. Still, as new technologies hit the market, the how is changing too.

For example, with the Internet of Cars on the horizon, what additional quality management features will the industry need to incorporate in order to address network security concerns? Today, every car is comprised of hundreds of programmable computing elements and millions of lines of code.[1.4] If the existing standards can't prevent software bugs from creating air bag malfunctions, what chance do they have at mitigating risk when human drivers take a literal backseat? Yes, the standards are changing, but maybe not fast enough.

Now, automotive companies will need to step up more when it comes to managing the quality of their software. Even the best developers will create bugs after coding a certain number of lines. With more source code being created in the automotive industry than ever before, the risk has never been higher. If you start with the reasonable assumption that defects are inevitable, then the question really becomes how you can identify and fix them early enough that they don't reveal themselves in a tragic accident and expensive recall.

The Functional Safety Standards Pillars

Software product quality assurance in the automotive industry has been intertwined with process assurance for more than 40 years. ISO 26262 provides the guidelines for safety assurance in new product planning, from concept through decommissioning. The core pillars of this standard are Automotive Safety Integrity Levels (ASILs) ratings, verification, and validation.

The ASIL Rating System

Specific to the automotive industry, ASIL ratings are a means of prioritizing areas of risk mitigation. The risk levels are ranked highest to lowest as ASIL D, ASIL C, ASIL B, ASIL A, and QM (Quality Managed). While risk level is important for determining the overall ASIL Hazard and process impact to your organization, the same outcome of hazard elimination is expected to be consistent across the scale. Before you can start building the Safety Case for your development process, you need to first determine the dependability requirements for your system, based on the following criteria. [1.5]

The three ASIL dimensions:

1. **The probability of exposure to harm should the system fail.**

2. **The controllability of the situation upon exposure.**

3. **The severity of the resulting harm should the situation not be controlled.**

Moving forward, controllability will be an interesting area to watch as assisted and self-driving capabilities become mass market technologies. With no driver, controllability decreases dramatically and increases ASIL risk levels across the board.

Verification & Validation

Verification is the demonstration that the designed model or code satisfies specifications and requirement, while also not including any unintentional functionality.

In practice, this process is often a combination of reviews, static analyses, and comprehensive functional testing at the model level.

Software testing is one of several verification activities intended to confirm that the software development output meets its input requirements. 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%." [1.6] 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.

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 achieve 95% in defect removal efficiency. [1.7]

There are even some verification requirements that can only be satisfied 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." [1.8]

In addition to code reviews, document reviews are needed throughout the software development process to ensure that the Software Development Plan is being followed. When choosing a peer review tool, make sure that the solution you select can handle both code and document reviews.

The Problem is Complexity

As systems become more capable, it becomes harder to test all the ways they will be used in advance. Once you test software and fix all 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. [1.6] Many of these defects will have no impact on the test case scenarios designed for testing. Yet, they could have devastating, unforeseen effects in the future.

ISO 26262 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. Allday 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 code 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 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 code reviews are the tools it provides its teams. The right tool set will enable each development team to 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.
6. **Keep Review checklists (if used) short –** contain no items that are obvious or can be detected via automation, and should focus on things that are easy to forget (e.g. "Are all errors handled correctly everywhere?").

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 reviews make the products your company produces better, it will make the processes and the people that produce them better as well. Code reviews are a powerful tool eliminating defects, but achieving compliance can be burdensome. Even in organizations where code reviews have been "adopted," they are skipped as much as 30% of the time, primarily because they are inadequately supported. [1.9]

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. [1.10]

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. [1.11]

In Conclusion

Peer reviews create an environment of shared understanding and collaboration. As developers review and comment on each other’s code, whether in real-time or asynchronously, they all get better. In the end, the code 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.

Endnotes

1. “Fiat Chrysler recalls 1.25 million trucks over software error,” David Shepardson & Nick Carey, Reuters, May 2017.

2. “GM recalls 4.3 million vehicles over air bag-related defect,” David Shepardson, Reuters, September 2016.

3. “Keeping up with design: ISO 26262 – time for an update,” Peter Els, Automotive IQ, March 2017.

4. “The future of the modern car is actually digital,” Bob O’Donnell, Recode, May 2017.

5. “Understanding ISO 26262 ASILs,” Chris Hobbs & Patrick Lee, Electronic Design, July 2013.

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

7. “Measuring Defect Potentials and Defect Removal Efficiency,” Capers Jones, CrossTalk, June 2008.

8. An Analysis of Current Guidance in the Certification of Airborne Software”, Ryan Erwin Berk, MIT, 2009.

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

10. Quality Processes Yield Quality Products, ”Thomas D. Neff, Cross Talk, June 2008.

11. “Does the Modern Code Review Have Value? ”H. Siy, Software Maintenance 2009.

SMARTBEAR
Collaborator
Reach your teams’ CMMI goals faster.
Start Your Free Trial Today