GRAMMATECH

MAKING SAFETY-CRITICAL SOFTWARE DEVELOPMENT AFFORDABLE WITH STATIC ANALYSIS

TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

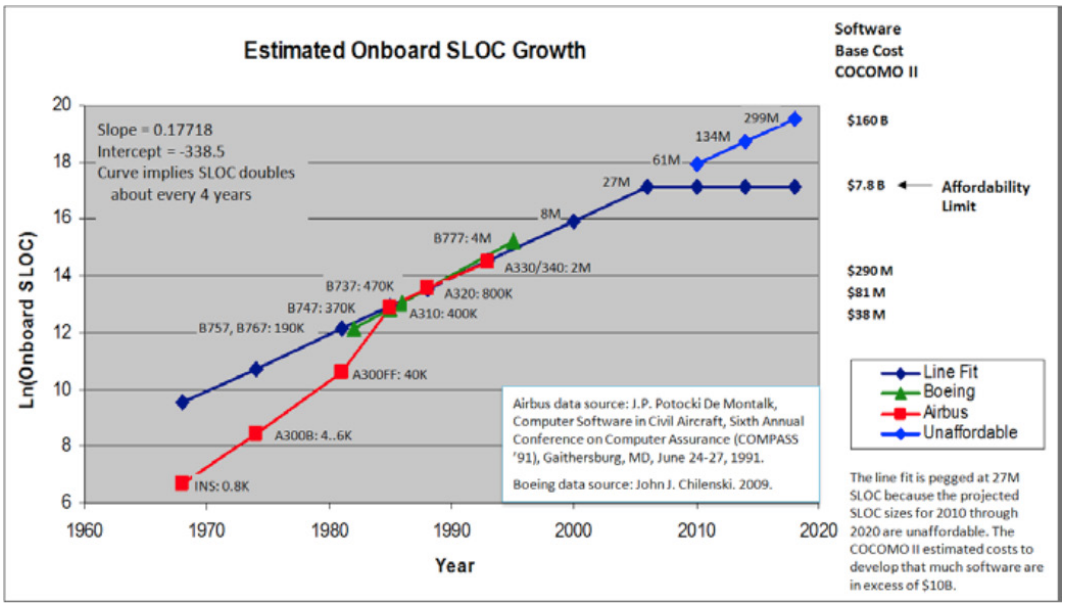
WWW.GRAMMATECH.COM

BACKGROUND

Safety-critical software has hit the "affordability" wall due to increasing complexity and growing reliance on software to perform mission-critical functions (Redman et. al. 2010). Software developer productivity on safety-critical systems hasn't really changed from 5 lines of code (LOC) a day and roughly 1000 LOC per year (O. Benediktsson 2000). However, with the growing reliance on software, the code size for safety-critical software has skyrocketed. Software affordability is a real problem, and static analysis tools are recommended by various standards and experts in the safety-critical software field as an essential tool for tackling it.

THE SAFETY-CRITICAL SOFTWARE AFFORDABILITY WALL

Software has become the leading cost of safety-critical systems. One third of new airplane costs now resides in software and software development. In automobiles, 25% of the capital costs of new vehicles are electronics (and software). Software has afforded amazing new capabilities, but its exponential growth and associated costs have made it effectively unaffordable:



Source: SEI, "Virtual Integration for Improved System Design", Redman et. al, 2010

Clearly the in-step increase in development costs for larger systems has to change, and for this various processes and techniques have been proposed. Amongst these is the use of tools – specifically static analysis – to improve test coverage and to detect defects that traditional testing cannot. In fact, both SEI and NASA recommend static analysis as an indispensable tool in safety-critical software development.

THE EXPONENTIAL COST OF FAILURE

Safety-critical software is expensive to develop and static analysis tools are highly recommended by both certification standards and practitioners in the field. Even more expensive is the result

of software failures in manufactured and shipped products, from recalls to litigation to damaged reputation. Toyota's unintended acceleration flaw, for example, in its drive-by-wire throttle system, cost the company up to $5 billion dollars in damages and lost revenues – a significant loss and important lesson in software safety. Although the Toyota example might seem extreme, a significant software failure can have almost unbounded financial impact – studies have shown that defects cost 100 times more to fix in production than in early phases of development.

As pointed out by Capers Jones (2009), looking at a cost-per-defect metric alone is misleading since it doesn't factor in the volume of defects and the fact that the cost to find and repair a defect is often the same over time (something developers are quick to point out). For safety-critical embedded systems, however, the cost of repair is higher than other industries. If a safety-critical defect isn't fixed on time (or worse, purposely hidden), the financial impact can escalate to legal liability and impact future revenue.

Considering the typical software development lifecycle illustrated by the V-model below, we can consider the relative benefits of static analysis at each phase of development. The V-model is a good example here, being featured in many of the safety-critical software certification standards (e.g. IEC 61508 and ISO 26262).

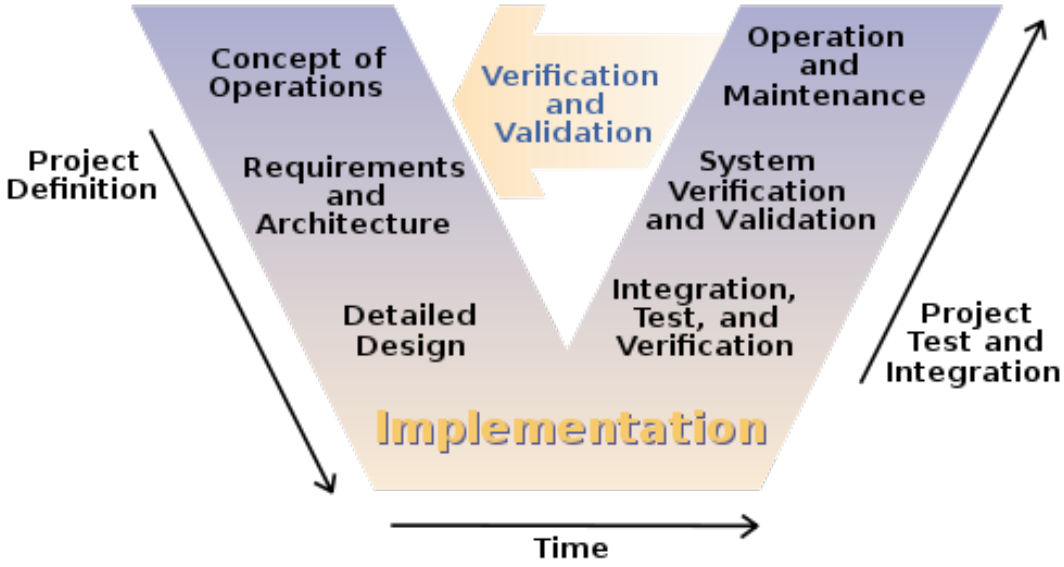


Figure 1: The V-model of system development, often referenced in safety-critical standards, i.e. IEC 61508 and ISO 26262.

Rather than looking at the traditional cost-per-defect over time or per phase, which Jones (2009) argues is true mathematically but doesn't reflect what is seen in practice, the more revealing data is the cost-per-defect per relative volume of defects (as volume of defects decreases over time and each phase of development), shown on the next page.

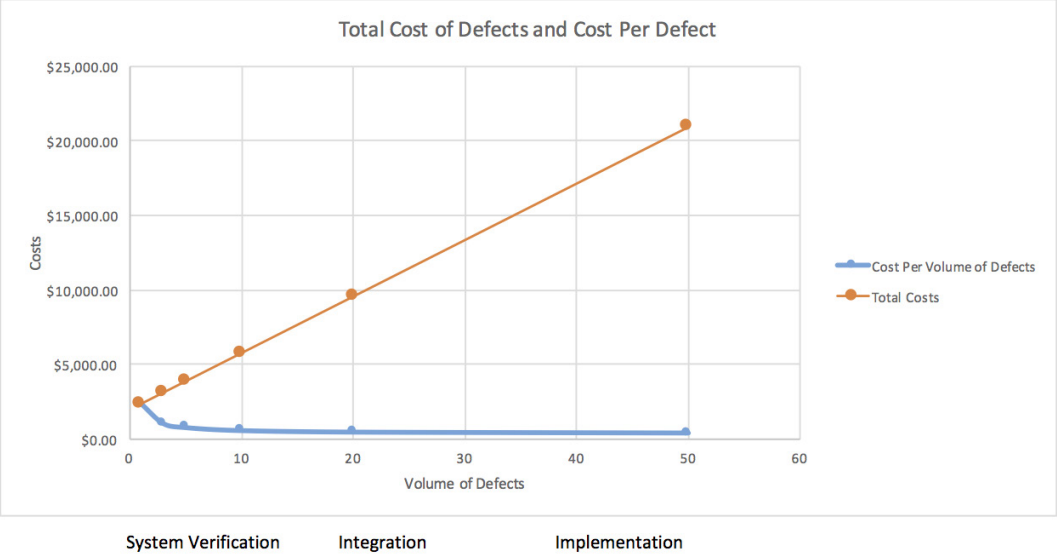


Figure 2: The total cost and cost per defect as the volume of defects diminishes. Source: Capers Jones (2009).

What is interesting about Figure 2 (note the development phase order), is that the cost-per-defect at each phase does go up as expected, but total costs are going down due to the decreased volume of defects. In practice, it doesn't take longer to find and fix bugs at each phase, but the costs are still there despite diminished volume. It's worth noting, also, that as a product matures into operation and maintenance (not covered in the chart), cost-per-defect is much higher due to the impact of servicing a fielded product. The other intangible costs, such as damage to brand and loss of future customers and income, are still factors to consider.

FUNCTIONAL SAFETY STANDARDS

Functional safety is end-to-end in scope, meaning that safety of a component or subsystem is evaluated in terms of the entire system (including both software and hardware). This is an important concept given that software was once considered independent, or insignificant in a system. The famous Therac-25 incident proved that idea false in a tragic way. Aerospace led the way with standards like DO-178 due to both the complexity of the software and the criticality of the systems being developed. Industrial, transportation, rail, and automotive followed over the years, usually as derived standards from IEC 61508.

What has also benefitted from the focus on functional safety is better focus on the impact of software control on the entire system and the risk analysis and management that is used at the system level. The focus on repeatable, documented processes and rigorous testing has helped software safety to improve immensely. The recent Toyota unintended acceleration problem indicates that improvements are still needed.

CERTIFICATION AND STATIC ANALYSIS TOOLS

Functional safety standards don't specifically require automated tools, but to efficiently meet certification requirements, tools offer an excellent return on investment. For example, ISO 26262 (Road Vehicles – Functional Safety) specifies software unit design and implementation principles and coding guidelines. Static analysis tools are particularly useful in enforcing coding standards such as MISRA C.

Help with coding standards is useful but it's just a small fraction of the capabilities of a product such as GrammaTech CodeSonar. Certification standards need robustness, correctness, and consistency, which require design, coding, and testing rigor beyond the coding standards. Static analysis tools can find defects in the source code before and after it's part of the project. The tools can also detect bugs that are hard to find in testing and are expensive to debug and fix. In addition, avoiding complexity and increasing maintainability is difficult to manage manually, and tools such as GrammaTech's CodeSurfer help immensely in management the structure of the code.

TOOL QUALIFICATION

Software certifications require proof of implementation to the standard, which is often manually generated, but automation reduces the workload. Confidence is required in an automated tool's results in order for them to be acceptable certification evidence. To address this, tools vendors can seek certifications for the products they sell as well. Recognizing this need, GrammaTech CodeSonar is independently certified for ISO 26262, IEC 61508, and EN 50128. This means that developers can use the tools with confidence that the results produced are acceptable to approval bodies during certification. It's just too risky to use unqualified tools, which will only result in further testing, documentation, and certification costs.

REDUCING COST AND RISK IN SAFETY-CRITICAL SOFTWARE DEVELOPMENT

Static analysis tools provide tangible productivity improvements to software teams seeking stringent software safety certification. Using a qualified tool as part of the software development process from early stages of development can have significant benefits:

- **Code coverage isn't everything:** Many safety standards require high levels of code coverage (proof that tests executed most, if not all, statements and conditions). Although this is very exhaustive, it's very expensive to do and must be repeated in each major phase of development (unit, integration and system testing). The criticality of the software dictates the level of coverage with some less critical software requiring no formal test coverage (e.g. aircraft on-board entertainment). Testing code coverage is one metric by which to evaluate software quality, but there are cases where it doesn't catch everything.

- **Bugs that coverage-based testing miss:** Testing software based on coverage metrics is inherently unit-based (although coverage is evaluated at a system level as well).

Concurrency errors and security vulnerabilities are two key instances of defects that can be missed even during rigorous testing. Concurrency is often tough to program correctly and can yield errors (e.g. race conditions) that are undetected until some unforeseen condition during operation. Security vulnerabilities do manifest as bugs in the code -- the conditions creating the error are often due to types of input not considered during testing. Static analysis can discover these errors early and prevent them from being show-stoppers late in the development cycle.

» **Detect defects early:** Rigorous testing can discover most defects in software, but it's expensive and extremely time-consuming. Discovering and fixing these bugs when writing the code is also considerably cheaper than later in the development cycle (defect discovery is exponentially more expensive over time). Static analysis can detect bugs in the code as it is written -- as part of a developer's development environment -- greatly reducing the downstream cost of defects.

» **Analyzing SOUP:** Use of third party code such as commercial off-the-shelf software (COTS) and open source software is a fact of life in embedded software development. Some safety standards consider any software that isn't developed to the specific standard as software of unknown pedigree (SOUP) -- software that needs to be looked at carefully for inclusion in the system. Static analysis tools can analyze third party source and binaries to discover defects and security vulnerabilities in software that could be impossible to test otherwise (without including it and running it, an expensive option).

» **Accelerate certification evidence:** Static analysis (and many other testing and lifecycle management tools) provide automated documentation to support testing, coding standard and quality/robustness evidence. Much of the manpower used in safety certifications is documentation and evidence production, automation and specifically static analysis reduces this burden significantly.

THE RETURN ON INVESTMENT FOR STATIC ANALYSIS

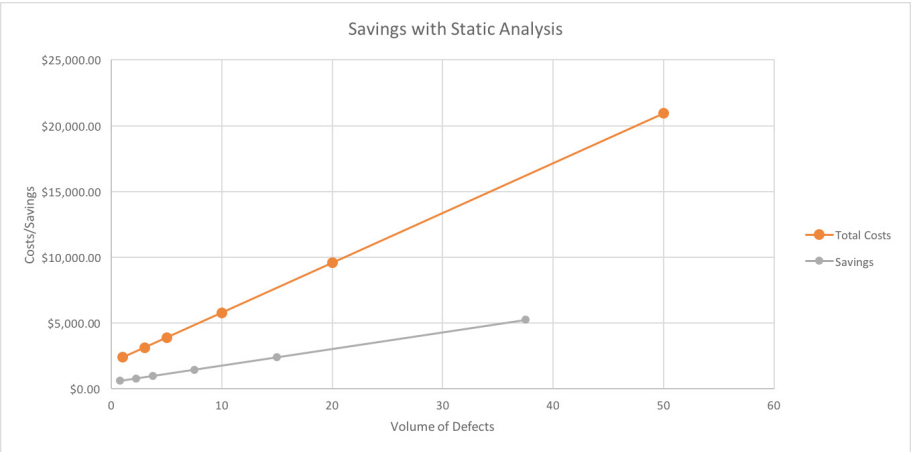


Figure 3: The savings versus total cost of reducing the number of defects entering testing at each phase by 25%.

So what is the return on investment given these factors? Static analysis decreases the volume of defects in software under development at all stages of development. A simple analysis is to reduce the number of defects from the data we have from Figure 3. Given this reduction in created defects during development, we can see a significant reduction in cost.

This simple analysis yields about $126 savings per defect, which, given an average of 15 defects per 1000 line of code (during development when defect volumes are high), gives a savings of $1,900 per 1000 LOC. Of course, results will vary, also based on other factors such as labor rates, defect detection and repair time, and defect density. However, given that many safety-critical systems employ 100 KLOC or more, the business case for static analysis is clear. (This analysis also doesn't include post-deployment costs which, as stated before, are much higher.)

STATIC ANALYSIS IS MORE THAN JUST DEFECT REDUCTION

In addition to defect-detection, CodeSonar is used to detect complex concurrency issues, analyze third-party source and binaries, and detect errors that traditional testing misses. These critical benefits are not factored into the rather simple analysis above, but clearly add to the tool's ROI. However, finding defects that "slip through the cracks" give the greatest economic benefits to the development team.

CONCLUSION

Safety-critical software is becoming exceedingly expensive to develop and manufacturers are looking for solutions that increase developer productivity. Static analysis tools are indispensable for safety-critical software development, so much so that experts in the field make them pillars of their software development processes.

Despite rigorous testing, failures still occur in safety-critical software, with catastrophic effects in human and economic terms. Static analysis tools are essential for safety-critical software, to ensure the development of software that is secure and high-quality. In some cases, safety certification standards recommend static analysis tools because of their ability to find defects that testing may miss and to enforce coding standards (among other benefits). The return on investment for static analysis tools is compelling, underscoring that static analysis plays an important part during development but also in system deployment into the marketplace.

By increasing development and testing productivity and finding bugs that are missed by regular testing, static analysis plays a key part in breaking through the safety-critical software affordability crisis.

REFERENCES:

Virtual Integration for Improved System Design, Redman et. al, 2010.

Safety critical software and development productivity, O. Benediktsson, 2000.

Four Pillars for Improving the Quality of Safety-Critical Software-Reliant Systems, SEI, 2013.

The Power of Ten – Rules for Developing Safety Critical Code, Gerald Holzmann, Jet Propulsion Laboratory.

Accounting for Toyota's Recalls, Foster School of Business, 2011.

The Economic Impacts of Inadequate Infrastructure for Software Testing, NIST, 2002.

A Short History of the Cost per Defect Metric, Capers Jones, 2009.

The Therac-25 Incident, Wikipedia.

GrammaTech, Inc. is a leading developer of software-assurance tools and advanced cybersecurity solutions. GrammaTech helps organizations develop and release high quality software, free of harmful defects that cause system failures, enable data breaches, and increase corporate liabilities in today's connected world. GrammaTech's CodeSonar is used by embedded developers worldwide.