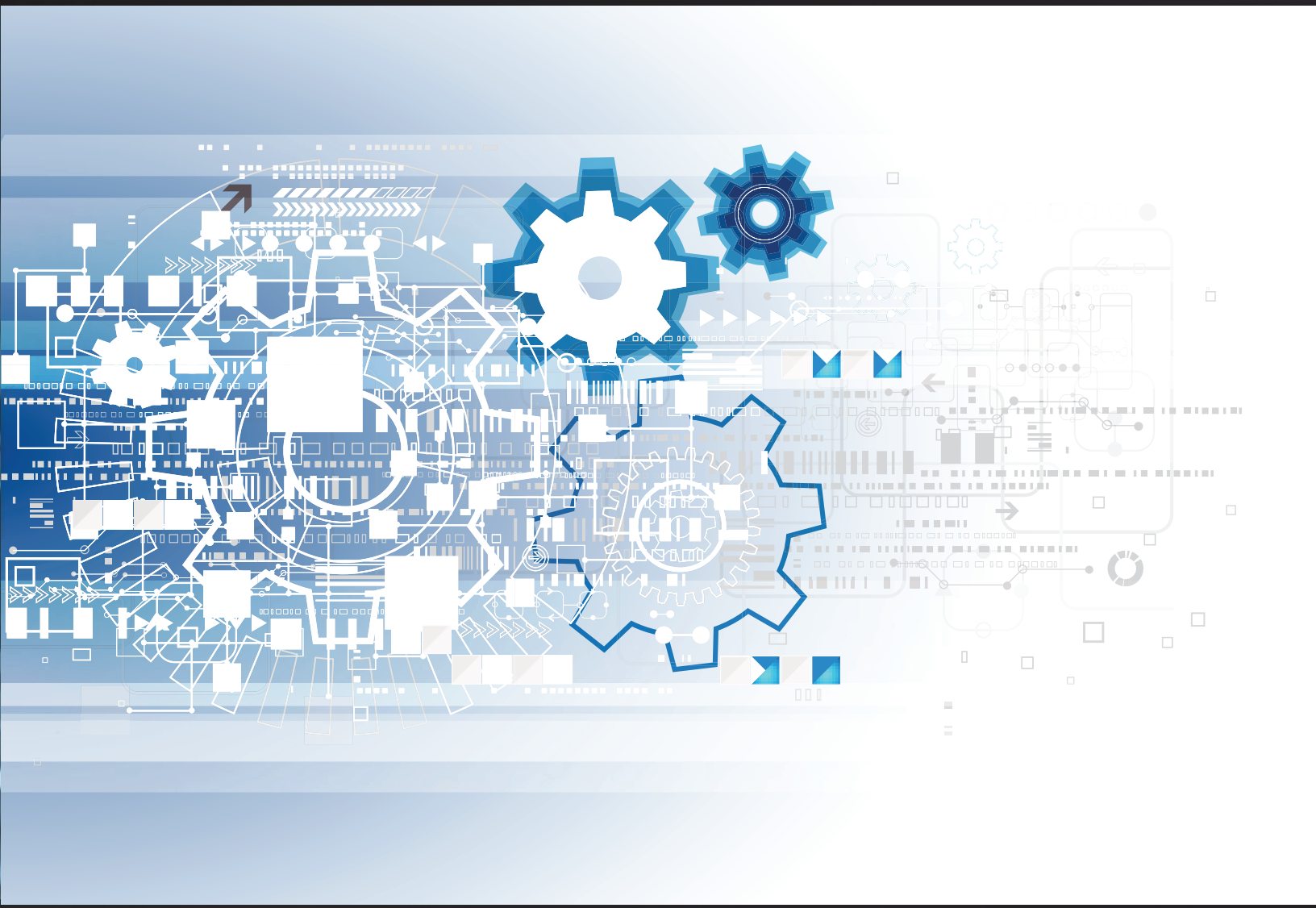
GRAMMATECH

ENHANCING CODE REVIEWS WITH **STATIC ANALYSIS**



TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

WWW.GRAMMATECH.COM

INTRODUCTION

Code reviews (or inspections) are a proven, effective way to reduce defects in software projects. In fact, defect removal rate due to code reviews can be as high as 75%, meaning two thirds of all bugs are removed during code review as part of the development phase of a project. Given the cost of dealing with bugs during tested, or even in fielded products, code inspections are part of good software engineering practice. This paper discusses how static analysis tools provide an ideal (and automated) companion to code reviews by supporting the process and increasing the defect removal rate.

INTEGRATING INTO AN EXISTING PROCESS

There are many opinions around about software development processes and the goal of this paper is not to argue about the process itself. Instead, the argument is that static code analysis should be part of your software development process. Inspections in general, not just of code, are a valuable technique to reduce defects. Inspections start during the requirements and architecture design and design phases of software development. The goal of inspections is to provide a qualitative measurement as to how well the work is progressing.

As with all practices, the earlier inspections are done, the bigger the pay-off in reducing downstream costs. Figure 1 shows the rough overlay of inspections (which is a superset of code reviews) over the software development lifecycle.

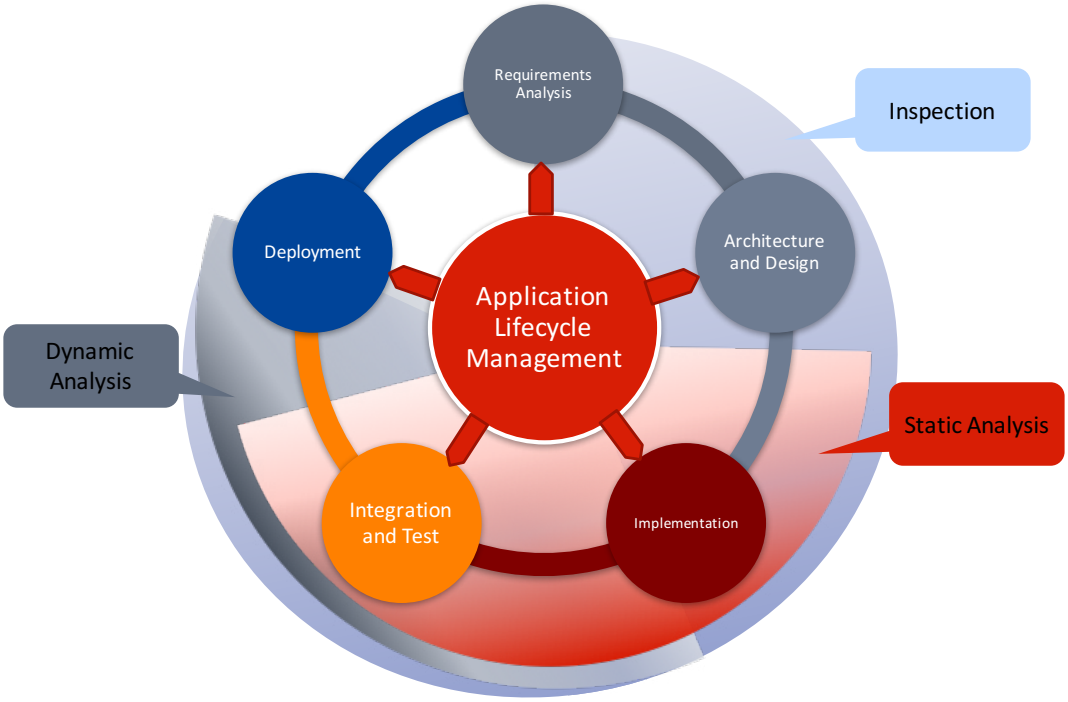


Figure 1: The overlay of inspection, static and dynamic analysis tools on the software development process.

Inspections, as with many activities during software development, can really benefit from the use of tooling and automation. Tooling encapsulates knowledge and makes it effortless to re-use that knowledge time and time again. Static analysis tools are a prime example of that. The tools encapsulate a certain body of knowledge about how to hunt for bugs and they can typically be extended to add more domain-specific validation if desired. A great primer in what static analysis tools can do is provided in GrammaTech's paper *Measuring the Value of Static Analysis Tool Deployments* (2016).

All software development projects have a workflow to check code into the code repository. This workflow usually begins with a developer writing code, typically including unit tests and subjecting that code to static analysis as part of his development workflow. Once he is done, he needs to go through the code review process. This typically requires either an in-person meeting with other developers, or a virtual meeting through tools like Crucible, Collaborator or others. In this meeting, the developer needs to present his code and the reviewers are responsible to review for correctness, understandability, style, security and many other properties. There typically is a checklist or some other document that is used in driving the review process.

BENEFITS OF STATIC ANALYSIS TO CODE INSPECTIONS

While everybody agrees that code reviews are extremely beneficial, there is a significant cost in man-hours to them. So any way that we can reduce the amount of time spent on them while enhancing the results is a sure win-win situation. This is where static analysis tools come in - they can provide a lot of input and automate some of the review process. Static analysis tools can be configured to take a fair amount of the manual labor out of reviewing the check-list. This is only one aspect of the benefits that automatic static analysis provides. Here are some of the other benefits that static analysis tools bring:

- **Work on large volumes of code:** It's often impractical to review all of the source code as usually new code is the top priority for inspection. Advanced static analysis tools can review millions of lines of code, including binary object code and libraries, for defects and security vulnerabilities.
- **Find difficult to see, obfuscated errors**: Static analysis tools can detect errors that can elude visual inspection and even functional testing.
- **Errors that span process/file boundaries:** Advanced static analysis performs complex code and data path analysis. This analysis can span functions and files that may be outside the scope of a unit under code review. Advanced tools also have unique concurrency checkers that detect multithreaded/multitasking issues which are very difficult to determine with inspection.
- **Detection of security vulnerabilities:** Testing and reviewing for security defects is difficult and involves a different mindset than testing for correct functionality. Static analysis tools can point out vulnerabilities and insecure coding practices. In addition, tainted data analysis discovers the path of input data to its eventual use within the system and cross references these with discovered defects.

- **Tool integrations:** Static analysis tools integrate with various Integrated Development Environments (IDE) such as Eclipse, build systems such as Jenkins, versioning systems such as GitHub and defect reporting tools such as JIRA. This brings the power of static analysis directly to the developer and incorporates it into the regular coding, build and test activities.
- **Reduce the code review effort:** Static analysis tools are automated, fast and effective. By running the tools on code before it's reviewed, it reduces the amount of defects in the code to be discovered manually. Static analysis tools can also enforce coding standards such as MISRA or NASA/JPL Power of Ten, removing that aspect of code reviews. Reports are available for each analysis providing supporting documentation for the review process.

Static analysis tools can be quickly integrated into an existing code review process. Figure 2 shows a modified code review process that includes using static analysis before manual review meetings. Presumably reports from the tools are included as part of the inspection and re-review.

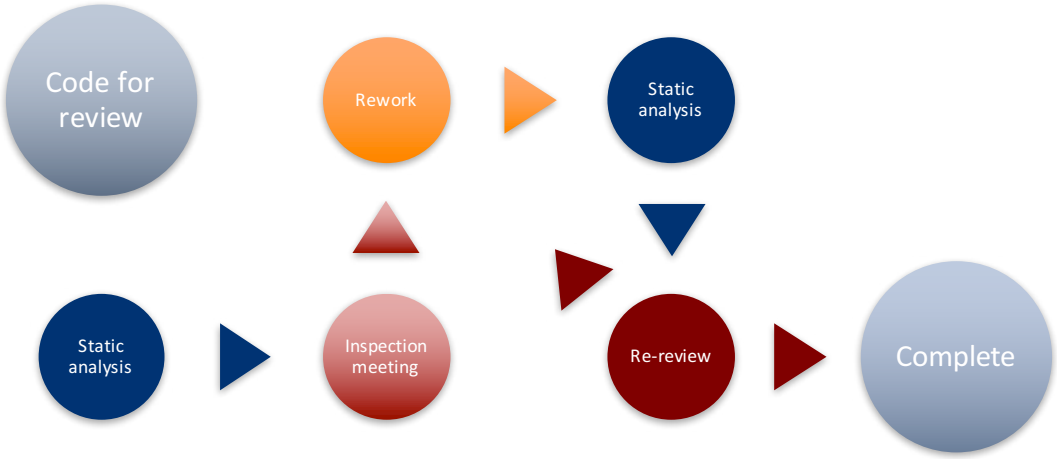


Figure 2: A modified code review process that includes static analysis.

REDUCE PROJECT COST WITH INSPECTION AND STATIC ANALYSIS

Inspections including code reviews have excellent return on investment. For example, one hour spent in inspection saves up to 33 hours in product maintenance. In addition, defect removal rate – the percentage of defects found early -- with inspections can reach 95%. Projects with little use of inspections have a considerably lower defect removal rate. For example, a project following a relatively light process such as CMMI (Software Engineering Institute's Capability Maturity Model) level 1, may have a defect removal rate of 78%. A project that makes use of modern processes that include inspections and automated tools such as advanced static analysis can achieve defect removal rates of 95% and above. Although the difference between 95% and 78% may seem small, it makes a big difference when defects cost five times more to fix in product maintenance than they do in development – a relatively conservative figure when outlier defects can be much more expensive.

The main discussion here is around software defects. However, we need to keep in mind that often software defects are the root cause of security vulnerabilities. A security vulnerability is typically built up around one or more software defects that allow an attacker to gain foothold into a software system. The cost of a exploited security vulnerability is often in the millions of dollars, which further builds a case to drive towards higher defect removal rates.

The cost benefit for removing defects early in the development lifecycle is well known. Let's look at how this relates to the defect removal rate. The graph in Figure 3 is based on the following assumptions:

- The engineering hourly labor rate is $75 per hour.
- Defects typically take four hours to fix in development.
- Defects caught after a product has been released are five times more expensive to fix, in other words, a single defect is $300 to fix in development and $1500 in product maintenance (again, a relatively conservative figure.)
- Approximately 1000 defects are introduced into the project during requirements, analysis and development phases. This is also a conservative estimate for a typical project with up to a 1000 unique function points.

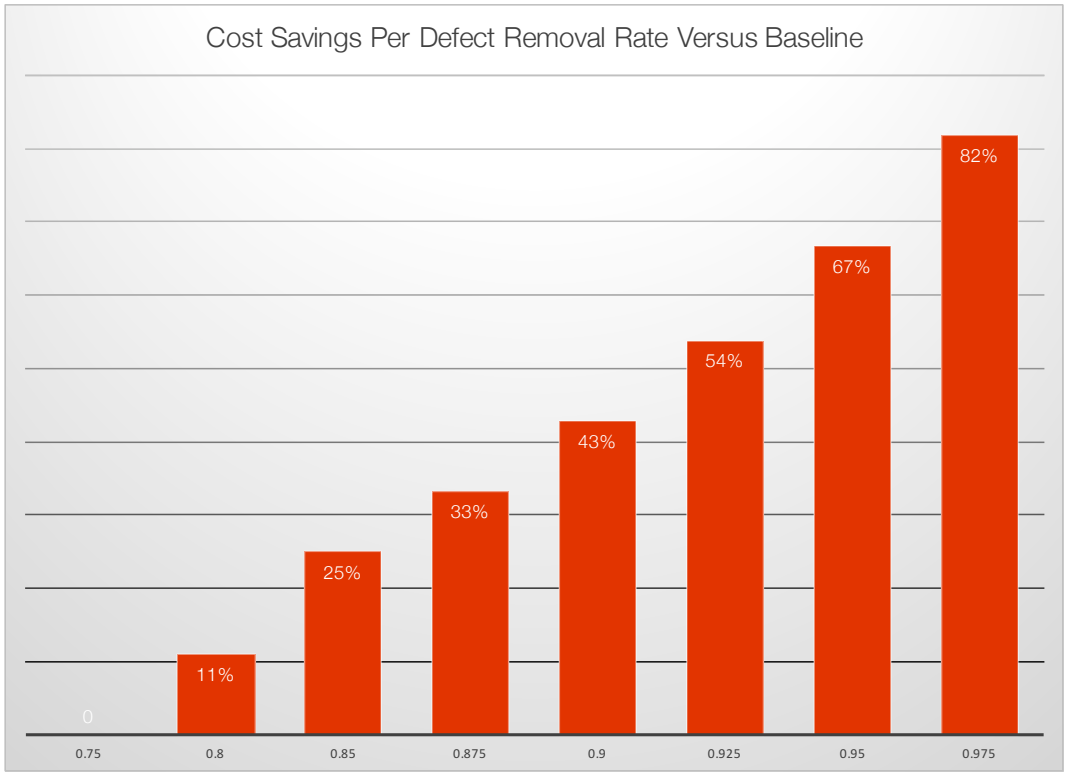


Figure 3: The relative cost savings per defect per defect removal rate versus the baseline of 0.75 or 75%.

Clearly, improving defect removal rate results in cost savings. As discussed in the previous section, the role of advanced static analysis in the code review process is to augment and improve the defect removal rate. Figure 4 translates the numbers from Figure 3 into dollars assuming that a project has 1000 defects introduced in the requirements and development phases.

For example, a software team already using code reviews can gain a reasonable increase of 10% in defect removal, say moving from 85% to 95%, results in double the cost savings ($240,000 versus $120,000.) Note that the biggest influencer in both graphs is the cost multiplier of fixing bugs and security defects after a product is released (conservatively set at 5:1 here.)

INTEGRATING INTO AN EXISTING PROCESS

There are many opinions around about software development processes and the goal of this paper is not to argue about the process itself. Instead, the argument is that static code analysis should be part of your software development process. Inspections in general, not just of code, are a valuable technique to reduce defects. Inspections start during the requirements and architecture design and design phases of software development. The goal of inspections is to provide a qualitative measurement as to how well the work is progressing.

As with all practices, the earlier inspections are done, the bigger the pay-off in reducing downstream costs. Figure 1 shows the rough overlay of inspections (which is a superset of code reviews) over the software development lifecycle.

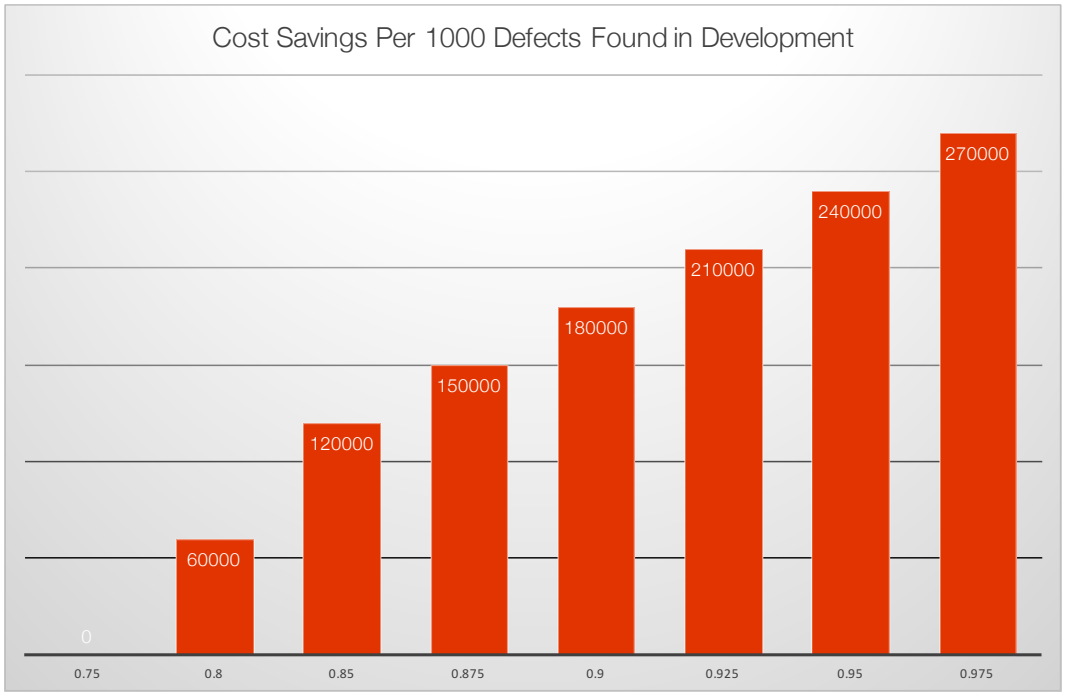


Figure 4: The relative cost savings in dollars versus the baseline defect removal rate of 75% (0.75) for every 1000 defects introduced during development.

GrammaTech CodeSonar

CODESONAR®

GrammaTech CodeSonar is an advanced static analysis tool that identifies bugs that can result in system crashes, unexpected behavior, and security breaches. Combining the bug finding prowess with team development support, CodeSonar is an ideal companion to code reviews.

WHY MANUAL REVIEWS ARE STILL REQUIRED

Code reviews and other inspections remain important. Static analysis tools are meant to augment an existing process and improve the outcomes. As such, it's important to highlight why code reviews are still required to achieve good quality and security.

- **Find defects that tools miss:** The success rate for good inspection processes is high, so code reviews are effective. By understanding the intention of what the code is meant to do provides human reviewers with a much better starting point to detect bugs. Luckily, automated static analysis and manual reviews complement each other as stated above.
- **Inspection is more than just code:** Reviews should be performed on all aspects of the software including requirements, design, test plans and then source. Finding and fixing requirements pays the biggest dividends by far in complex projects. It's important to realize code reviews are part of a broader inspection process.
- **Provide important context to tool reports:** Developers understand the requirements of the system (or should!) and the expected outcomes of its operation. Static analysis and other tools can only reason so much about the intended purpose of code and human-led reviews provide the required context to reviews and associated tools reports used.
- **Validate tool findings:** Static analysis tools require interpretation of the errors reported. The code review meetings provide a forum for clearing up any contentious reports.
- **Intangible benefits:** Code reviews are very much a human process and as such the process, meetings, reviews and the interactions therein provide important benefits such as mentoring new developers, consistency in design and implementation, team cohesion, and providing management with a rough barometer of progress and quality.

CONCLUSION

Code reviews are here to stay but are enhanced with static analysis tools. The human-led inspection complements tool automation nicely. Static analysis tools reduce time and costs for code reviews while enhancing the outcome of the process.

REFERENCES

1. “A Guide to Code Inspections”, Jack Ganssle, May 2012, The Ganssle Group (link: http://www.ganssle.com/inspections.pdf)
2. “Measuring Defect Potentials and Defect Removal Efficiency”, Capers Jones Software Productivity Research LLC, 2008, (link: http://static1.1.sqspcdn.com/static/f/702523/9242274/1288742153797/200806-Jones.pdf)
3. “Software Defect Origin and Removal Methods”, Capers Jones, Namcook Analytics LLC, 2012 (link: http://www.ifpug.org/Documents/Jones-SoftwareDefectOriginsAndRemovalMethodsDraft5.pdf)
4. “Get Software Quality Right”, Capers Jones, Dr. Dobb’s, 2010 (link: http://www.drdobbs.com/architecture-and-design/get-software-quality-right/225701139)
5. “An hour of inspection saves 30 hours of maintenance”, Glen W. Russell, 1991, IEEE Software, 8, no. 1, (link: http://ifsq.org/finding-iip-2.html)

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.