STATIC CODE ANALYSIS: A BUYER’S GUIDE

KEY DECISION CRITERIA FOR SELECTING TOOLS TO DEVELOP EMBEDDED SOFTWARE THAT IS MORE RELIABLE, SAFE, AND SECURE.
TABLE OF CONTENTS

OVERVIEW .......................................................................................................................... 3
WHAT IS STATIC CODE ANALYSIS? .................................................................................. 4
BENEFITS OF STATIC CODE ANALYSIS ......................................................................... 4
  Reduces Risks Associated with Large, Complex Code Bases .................................... 4
  Improves Security and Reduces Vulnerabilities ...................................................... 4
  Enhances Reliability and Safety .................................................................................. 5
  Streamlines Processes ............................................................................................... 5
  Saves Time and Reduce Costs through Early Detection and Remediation .......... 5
CHALLENGES OF SOURCE CODE ANALYSIS ................................................................. 6
  Static Analysis is Not Dynamic .................................................................................. 6
  Too Much Noise and Too Many False Results ...................................................... 6
  Too Many Errors To Fix ............................................................................................ 7
  Difficulties with Process Integration ..................................................................... 7
  Lack of Buy-In and Motivation ............................................................................... 7
BEST PRACTICES ............................................................................................................... 8
  Use Static Analysis for Progress Not Punishment ................................................ 8
  Use Coding Standards ............................................................................................. 8
  Prioritize Code Analysis .......................................................................................... 9
  Address Your Results .............................................................................................. 9
  Tune Your Analysis .................................................................................................. 9
HOW TO CHOOSE A STATIC CODE ANALYSIS SOLUTION ............................................ 10
  Requirement 1: Integration with Development Platforms and Processes ......... 10
  Requirement 2: Test Accuracy .................................................................................. 11
  Requirement 3: Actionable Diagnostic Output ..................................................... 12
  Requirement 4: Ease of Use .................................................................................... 12
  Requirement 5: Standards Compliance Checking ................................................. 13
  Requirement 6: Workflows and Reporting ............................................................ 13
CONCLUSION ................................................................................................................... 14
OVERVIEW

The Internet of Things (IoT) is no longer a far off concept; it’s here now. Modern society has become entirely dependent on software-based technology. Mobile phones have replaced alarm clocks and cameras, GPS has replaced paper maps, and the list goes on. New products and new markets are being rapidly created based on software innovations, offering unforeseen benefits to consumers and increased revenues for manufacturers. Delivering these new products, however, presents a unique set of challenges in software and product development. Software security, reliability, and safety are the criteria that will determine success or failure for many businesses.

Developers employ a wide range of tools to build embedded software, including tools for analyzing, designing, documenting, and editing code, as well as tools for compilation, debug, test, optimization, and verification. The tools used within an embedded software development environment are the single biggest factor in determining the productivity and effectiveness of developers. Among the most vital of these tools are those tools used to eliminate software bugs – the most expensive, time consuming, and high-risk aspect of product development.

Numerous studies have shown that significant cost savings can be achieved through static analysis of software code, particularly early in development. Further, static analysis contributes to improved performance, more efficient utilization of development staff, and a general reduction in downstream bottlenecks.

This paper describes the importance of static analysis to delivering quality embedded software as well as the advantages static analysis provides in ensuring embedded applications perform well in production. It also covers the limitations of static analysis tools, highlighting the need for establishing and maintaining the proper processes and organizational support to ensure success and maximize value.

Further the paper outlines several best practices for source code analysis and key capabilities to look for in a static analysis solution. Without the right tools and processes in place, simply automating code review activities will likely not deliver the desired results. Understanding how to apply the right tools and practices to make the most of source code analysis is fundamental to static analysis-based inspection and vital to ultimately delivering reliable, safe, and secure software.
WHAT IS STATIC CODE ANALYSIS?

Static code analysis is the process of evaluating a system or component based on its form, structure, content, or documentation. From a software assurance perspective, static analysis addresses weaknesses in program code that might lead to vulnerabilities; this type of analysis may be manual, as in code inspections, or automated through the use of one or more tools. Automated static code analyzers check source code for specific defects as well as for compliance with various coding standards.

BENEFITS OF STATIC CODE ANALYSIS

Static code analysis enables development and testing teams to make significant advances in conducting more efficient and more realistic large-scale code reviews. Automating code analysis and documentation also enables organizations to realize major reductions in development cost and time.

REDUCES RISKS ASSOCIATED WITH LARGE, COMPLEX CODE BASES

As software becomes more critical to delivering product value, code bases are rapidly growing and becoming more complex. Code is being developed across the globe by both internal and outsourced development teams. New code is combined with large legacy code bases that are being reused and modified for current applications. While reusing code can reduce costs, it can also increase complexity and the risk of defects. And while outsourcing development has advantages, it too adds more variability and complexity to a project. Static analysis tools address this complexity and its associated risk on several fronts. First, they enable organizations to analyze millions of lines of code in a small fraction of the time it would take a team of developers. Second, they analyze code before it is reused in a project and as it is added to a project. Third, they provide a way to enforce development standards across both internal and external development teams.

IMPROVES SECURITY AND REDUCES VULNERABILITIES

The growing reliance on embedded software raises the importance of security considerations during software design. Buffer overflows, resource leaks, insufficient encryption, insecure interfaces, and other defects and security issues remain too common in embedded systems. As more devices are deployed and connected to the Internet of Things, the number of entry points for hackers is increasing dramatically. Static analysis helps ensure coding consistency and is an effective means of finding the most common security defects including buffer overflows and resource leaks among others.
ENHANCES RELIABILITY AND SAFETY

Engineers developing highly complex, safety-critical software systems in automotive, aerospace, medical, and an increasing number of consumer products must ensure reliability and quality of those systems. Many systems that rely on software to operate safely and consistently undergo rigorous testing once they are built. These practices typically align with ISO 9001/CMMI and standards and practices specific to the certification of high integrity software systems such as SAE ARP4754. Static analysis tools can be used to minimize software defects that lead to disastrous system failures, injuries, or even fatalities. Static analysis is particularly useful to measure code complexity and verify compliance with coding standards (such as MISRA C) on DO-178C, ISO 26262, and other projects; demonstrate that the software will not fail due to certain types of runtime errors; and assess the effects of making changes to the application while helping to determine if changing code in one area may impact the reliability or functionality of code in another.

STREAMLINES PROCESSES

Static analysis provides feedback to help developers adopt better programming habits, enabling them to write better code faster. Tools that provide well-documented feedback to remediate a potential issue and can pinpoint the precise line of code on which the defect occurred enable developers to understand their errors in the context of what they were trying to achieve. With timely feedback, it is more likely that developers will learn from their errors and avoid similar issues going forward.

The faster developers learn about issues, the easier and quicker it is to fix the errors. Unlike testing that occurs weeks or months after the code is written, static analysis can be applied as the code is written. So developers don’t need to waste time trying to remember what the code was supposed to do, why they wrote it the way they did, or what downstream affects they need to consider when modifying the code to fix the errors.

Because code complexity affects maintainability, one of the other major benefits of static code analysis is the ability to measure code complexity and thus maintainability. In the long run, improving the maintainability of your application, even if it works as expected, will enable your organization to adapt code as your requirements change and evolve and extend it for reuse in other projects.

SAVES TIME AND REDUCE COSTS THROUGH EARLY DETECTION AND REMEDIATION

The most significant difference between static analysis and other testing methods is the ability to identify errors without executing software. Exposing defects without executing code is especially valuable for embedded applications, where comprehensive runtime analysis for such errors is often ineffective if not impossible.

Static analysis can be run immediately; there is no need to spend weeks setting up and configuring dozens of test cases. This type of analysis also helps software developers to automatically uncover errors that are typically missed by unit testing, system testing, quality assurance, and manual code reviews.

Any setback in embedded system development is often magnified by dependencies that span multiple engineering disciplines, including software, mechanical, and electrical. Delays in one stage can cascade into other areas, resulting in substantial development cost overruns and deferred revenue opportunities. Static analysis provides the early detection and remediation necessary to avoid this costly ripple effect.
CHALLENGES OF SOURCE CODE ANALYSIS

The advantages to automated static analysis testing are clear, but it’s not a panacea. Some commonly expressed challenges and objections to using static analysis include:

- Static analysis is not dynamic
- Too much noise and too many false results
- Too many errors to fix
- Difficulties with process integration
- Lack of buy-in and motivation

Despite these perceived challenges, static code analysis plays a prominent role in comprehensive test plans, particularly for projects with changing requirements, multiple hardware deployment scenarios, and geographically distributed teams.

STATIC ANALYSIS IS NOT DYNAMIC

Some problems are impossible to discover using static analysis alone. The execution and interaction of multiple functions can produce unexpected errors, which only come to light at later stages, such as component-level integration, system integration, or deployment. With this in mind, dynamic analysis should be used once the software is functionally complete. Dynamic analysis will test how the application performs as it executes, and how it interacts with other processes and its operating system. Static and dynamic analysis are entirely complementary. Static analysis finds errors early in the software development life cycle and dynamic analysis tests the code in real-life performance scenarios. The ability to test code early in development cycle makes it easier to find and address the root cause of performance and security problems.

Because static analysis tools and dynamic analysis tools are complementary, an observation that one category of tools has capabilities that the other doesn’t is accurate but not revelatory. It’s like noting that one cannot pound nails with a saw or cut wood with a hammer. An effective static analysis tools provides numerous benefits, but it cannot be expected to deliver the same results that a dynamic testing tool will.

TOO MUCH NOISE AND TOO MANY FALSE RESULTS

A wide variety of static analysis tools are available, ranging from simple open source bug finders to sophisticated platforms. Many simple bug finders can be difficult to use because they are inaccurate, flagging so many false alarms (false positives) that it is difficult to identify the real issues. By default, many of the more sophisticated and automated platforms provide hundreds or even thousand of rules to check. While these rules can be useful, it can be difficult to select the best options for your organization’s development environment. When organizations lack a proper understanding of how to use the rules selectively and apply them to analyze their code, the tools will generate lots of noise, distorting the organization’s perception of application code quality. This noise not only makes code analysis inefficient and a burden, it also undermines the whole effort by raising doubts about the tool’s effectiveness, increasing frustration, and demoralizing the development team.

While false positives create a burden, false negatives can be a bigger problem. When tools are ineffective or misused, defects go undetected (false negatives), leading to false confidence in the software being released.
TOO MANY ERRORS TO FIX

The use of static analysis doesn’t remove the need for human analysis, judgment, inspection, and ultimately defect remediation. Tools can produce many reports using a variety of formats and conventions. Developers and QA testers must perform triage on this output to distinguish real defects from false positives. Once false positives are weeded out, the developer needs to fix the real defects.

Without the right tools or processes in place to prioritize importance and urgency, fixing these errors can be overwhelming. Effective tools provide precise diagnostic, and guidance on how (and if) the error should be addressed. Tools that do not provide this information can lead to frustration and missed deadlines. In some cases, developers faced with too many errors will mark real defects as false positives or perform quick fixes to satisfy the tool. Processes that don’t allow developers time to fix errors increase the risk of developers circumventing procedures and manipulating results.

DIFFICULTIES WITH PROCESS INTEGRATION

Introducing any new tool or technology into the development process means change and people tend to resist change. Organizations must plan the integration of any new tool into existing processes to avoid creating a disruption that it is perceived more as adding “busy work” than improving the development process and addressing critical problems. Organizations must have a realistic expectation of the time and costs (both direct and indirect) of integrating static analysis into the development process.

LACK OF BUY-IN AND MOTIVATION

Depending on the static analysis tool selected, an organization may have more or fewer issues applying it in their process. The main challenge, as is almost always the case when it comes to technology, will likely come from people. Whether the tool deployment succeeds or fails depends on the people behind it. All stakeholders - developers, middle management, and upper management - must be motivated with a clear understanding of the expected benefits and a realistic view of the expected costs. Stakeholders may have different reasons for resisting change, from concerns about disruption to fear that it will expose the lack of quality and highlight the number of defects in one’s code. Wide acceptance of the tool by all stakeholders, however, is a prerequisite for getting the best possible benefits.
The following best practices can help you maximize the advantages - and minimize the challenges - of adopting static code analysis.

USE STATIC ANALYSIS FOR PROGRESS NOT PUNISHMENT

Whenever management mandates using a tool such as one for static analysis, there is a tendency among some developer groups to focus on its potential negative implications. Developers can view these types of tools as intrusive micromanagement used to monitor their progress and penalize them when things go wrong.

Managers who are introducing static analysis tools into either the build process or the developer workflow should present the tools in a way that encourages their use. It’s important to emphasize the goal of improving overall software quality rather than monitoring individual developer capabilities. Developers can improve their skills as they use static analysis tools, which codify a great deal of knowledge about common software issues. Using this knowledge to build more effective coding practices will benefit the entire team because not only will developers start writing better code, but consistency among developers will increase.

USE CODING STANDARDS

Development managers everywhere know that if you can’t measure it you can’t manage it. Or, put another way, what gets measured gets done. Organizations strive to produce code that is maintainable and reusable. A high degree of variability and inconsistency in the code base makes it difficult for others to understand what the original programmer intended and thus makes the code difficult to modify. By introducing standards and objective measurements, organizations can minimize inconsistency.

Coding standards provide developers with guidelines on how to write code and minimizes the confusion and inconsistency that results from each developer coding in their own style. Documented coding standards help to ensure that all programmers conform to consistent guidelines when coding. When a large project is coded to established standards, different parts will be coded by different developers, but the code as a whole will be more uniform. Not only does this approach make the code easier to understand, but it also means that going forward any developer who works on the code will know what to expect throughout the entire application.

Coding standards also provide guidelines for producing more reliable, safe, and secure applications. When guidelines are established based on best practices tied to safety critical applications, such as the MISRA C standard, they prevent defects and poor coding that results in functional performance issues and failures. Similarly, following coding standards such CERT C can prevent many coding errors that create exploitable security weaknesses.

By adopting a standards-based coding practices and automating compliance checks for those standards with a static code analysis tool, teams can make significant improvements to code security and quality without slowing the project or lowering team productivity.
PRIORITIZE CODE ANALYSIS

For large-scale tests, it’s a good practice to take an iterative approach in which prioritization is applied to address the code with the greatest potential risk. Michael Howard proposes the following heuristics for determining code review priority:

- **Old code.** Older code may have more vulnerabilities than new code because newer code often reflects a better understanding of security issues. Code considered “legacy” code should be reviewed in depth.
- **Code that runs by default.** Attackers often go after installed code that runs by default. Such code should be reviewed earlier and deeper than code that doesn’t execute by default. Code running by default increases an application’s attack surface.
- **Code that runs in elevated context.** Code that runs in elevated identities, e.g., root in *nix, for example, also requires earlier and deeper review because code identity is another component of attack surface.
- **Anonymously accessible code.** Code that anonymous users can access should be reviewed in greater depth than code that only valid users and administrators can access.

ADDRESS YOUR RESULTS

Modern static code analyzers can find many problems. However, it is important to address the problems as they are discovered because one problem could be masking another.

One of the easiest ways to ensure that problems are addressed as you go is to raise their visibility. An analysis platform that provides enterprise management dashboards and “drill downs” to stakeholders across the organization can help you achieve this necessary visibility. It expands the reach of code analysis by offering centralized reporting and code scanning results to everyone affected by code development activities and quality.

Organizations should strive to make static analysis part of the daily build and to produce ongoing reports of analysis results. It’s been well proven that finding and fixing problems early is the key to successful development. Static analysis and reporting, however, is only half of the solution; keeping everyone on the development team focused on remediating the problems in the reported results is the other.

TUNE YOUR ANALYSIS

In deriving value from static analysis tools, the role of a QA manager is fundamental. The most important thing to keep in mind is that the value of static analysis lies in the rule sets you apply and their ability to detect possible defects. As a result, it is paramount to choose the right sets. Whether your goal is to ensure security, safety, or performance, the indicator of quality you need to measure will dictate the defects you are concerned about. Any tuning task should be consistently applied across the team and organization. Further, it must be easy to properly tune the tool for the desired function.
HOW TO CHOOSE A STATIC CODE ANALYSIS SOLUTION

While the range of available static analysis tools may seem quite broad and many solutions will identify potential defects in your code, there are few that are designed to meet the needs of embedded developers and even fewer that enable you to follow the best practices outlined here and take full advantage of the benefits that static code analysis offers. Static analysis tools come in many flavors, from simple lint-like tools (both open source and commercial), to “bug-catchers” often bundled with testing tools, to more advanced automated code inspection platforms.

Using a Swiss-army knife offers the ability to perform many necessary functions at a low cost, but using purpose-built tools such as power drills and cutlery knives enables better results for specific tasks. In much the same way, dedicated static analysis platforms offer clear advantages over multipurpose open source packages that include a rudimentary static code analysis tool as one of a host of other tools.

When considering a static analysis solution, ask the following questions:

- To what extent does the solution integrate with my development environment? Does the solution provide accurate results with minimal false positives?
- Does the solution deliver actionable diagnostics to help developers fix defects?
- Is the solution easy to use, or will we spend weeks learning and configuring it?
- Does the solution include full-featured reporting and decision-making modules to help our team make the most of the results?
- Is the solution scalable? Can it handle millions of lines of code?
- Does it make business sense?

When evaluating potential static analysis solutions keep the requirements in this section in mind.

REQUIREMENT 1: INTEGRATION WITH DEVELOPMENT PLATFORMS AND PROCESSES

If you choose a solution that is not integrated with your developers’ environments, you’ll need to complete several time-consuming tasks on your own. First, you’ll need to learn how each developer environment and IDE you’ll be using works, including its limitations and constraints. Second, you’ll likely need to build, test, and maintain your own import, export, and reporting mechanisms through the platform’s APIs.

Static analysis solutions that offer integration with popular development platforms and tools simplify and accelerate the steps needed to use them. Look for a solution that supports easy integration with the vast majority of build automation systems and IDEs. A well-integrated solution eliminates the need for switching between multiple tools and screens to write, test, and modify code and enables an efficient and seamless workflow that speeds development.

Support for a broad range of development models is also critical. Whether you use an agile, waterfall, or hybrid approach, the platform needs to work within your current process and have flexibility to adapt if your process changes.
Since the goal of using automated static code analysis solutions is to provide ongoing testing of applications as part of the development process, the platform must be able to support the frequent code changes made by teams using agile development and continuous integration. Under these conditions, code testing tools must be efficient to avoid becoming a bottleneck. The addition of code analysis checks should not and cannot cause delivery delays. Automation and integration are vital to incorporating code analysis while maintaining development schedules.

**REQUIREMENT 2: TEST ACCURACY**

False negatives – defects that weren’t discovered during testing – can be due to a variety of reasons. Among the leading causes are incomplete application coverage (the entire application was not tested) and inadequate testing tools (testing was performed but the tools didn’t find the defect). Regardless of the cause, false negatives pose a huge risk.

One way to minimize false negatives is to maximize code coverage. It is important that all code on all tiers is properly tested, including reused libraries and components as well as internally developed code and externally developed code. Attackers are capable of finding almost any vulnerability, and maximizing test coverage to reduce false negatives is a vital countermeasure.

Likewise it’s just as important to eliminate false positives. When results contain false positives, security-knowledgeable developers must distinguish the real weaknesses from the noise. Excess noise increases the chances of missing security issues and significantly slowing down the development process, which is particularly problematic in agile environments where speed is a priority.

One advantage of using automated tools is to remove the human factor, but false positives add it back in, as human analysis is needed to interpret and triage results. Compounding the problem, false positives erode trust between management and coders. When overworked and overstressed developers are required to analyze and fix nonexistent issues they will grow frustrated with the tools that deliver the results and resentful of management’s decision to use these tools.

Static analysis can require balancing competing objectives: minimizing false negatives, minimizing false positives, and minimizing the time needed to complete analysis. Effective static analysis tools can be configured and tuned to meet the accuracy requirements of specific use cases. For example, organizations developing safety critical software and those seeking to eliminate security vulnerabilities prioritize minimizing false negatives, because those issues can significantly affect the overall success of the product.
REQUIREMENT 3: ACTIONABLE DIAGNOSTIC OUTPUT

Effective static analysis tools provide clear results so that it's there is no question why a potential defect has been flagged and what needs to be fixed. Individuals who are not expert coders or security experts need to be able to review and understand the results, without relying on specialists, in order to prioritize and remediate the reported defects and vulnerabilities themselves.

The quality of the tool’s output affects the speed of development because the output informs plans on how defects are managed and remediated. After the tool analyzes the application, if it cannot pinpoint the place in the source code where each vulnerability or defect originated, and if it cannot suggest remediation steps to be taken for the particular defect then the tool is of limited value. Conversely, the better the tools is at pinpointing errors and suggesting remediation steps, the more value it will provide to both developers and management. The analysis results must be clear and easy to understand while providing all relevant technical and risk information.

Application testing is part of project and risk management, and identified vulnerabilities need to be prioritized in terms of remediation urgency. A well designed static code analysis tool will enable and assist this process by providing results that make it easy to see both the risk associated with the defect as well as the remediation effort required. Further, well-documented and clear diagnostics enable developers to learn how to write better quality and more secure code.

REQUIREMENT 4: EASE OF USE

While incorporating static code analysis requires planning and the ability to accept change, it should not require an overhaul of your software development lifecycle (SDLC). Tools that require complex new procedures to be put into place have negative consequences. Either the tool will be limited to ad hoc use and never become part of the SDLC, or it will require a lengthy integration process that drains resources from the organization and takes too long to complete. In both these cases, the applications being developed remain at risk for quality and security flaws. Consequently, give careful consideration to the amount of training required to incorporate a new tool, and look for one that doesn’t require extensive training.

Tools that are too technical and that require significant expertise to interpret and act upon their results are difficult to integrate. If they require expert staff with a high level of coding experience or security training to operate them, then they will strain resources or may even require outside consultants. The process will be too slow and inefficient, especially for teams using agile development or continuous integration. If the tool cannot be used by existing staff and instead requires outside expertise then it introduces unacceptable bottlenecks in the development process.

In general, static code analysis solutions should be easy to deploy, configure, and scale. Keep in mind that some tools may take a bit longer to implement due to fine tuning of rule sets and workflows, but this investment in time up front pays off in the long run if the tool provides more accurate results and provides a greater ability to scale. In short, select a tool that you can integrate into your organization without significantly upsetting the SDLC process and creating unnecessary overhead.
REQUIREMENT 5: STANDARDS COMPLIANCE CHECKING

Documenting standards compliance can take valuable time away from other development activities. Automated static code analysis can be employed to monitor standards compliance, while significantly reducing the time required. An effective solution will quickly and thoroughly assess code complexity and check compliance to coding standards such as MISRA C/C++ or JSF++ before producing the supporting documentation of compliance. When evaluating solutions, consider how well each tool supports existing coding standards and how will it support customization for your organization’s own internal standards. It should enable you to select a subset of pre-existing coding rules, and supplement these with additional company-specific rules and styles.

REQUIREMENT 6: WORKFLOWS AND REPORTING

Development efforts affect a variety of different stakeholders and managers across the organization. With this in mind, look for static analysis tools supported by a platform that provides an infrastructure for process-driven review management throughout the development process. Such a static analysis platform should be comprehensive and flexible to enable different stakeholders to understand results and their associated risk.

Because not everyone needs the same information, consider whether the solution provides role-based visibility and management capabilities for different stakeholder levels and interests.

- Business owners, product managers, and project managers are concerned with the overall risk level and action items, and the solution should provide this big picture view, along with justification for the suggested remediation plan.
- Security and quality managers need to see the project status to properly allocate resources to projects with more serious issues, prioritize a global remediation plan, and evaluate the current exposure of the organization.
- Development managers and team members should be able to assess their quality and security status independently and as a group, to understand the associated risks and prioritize a defect resolution plan. They should also be able to measure their progress against their plans.

The importance of reporting shouldn’t be overlooked since it is crucial for communicating test results and project status to all stakeholders, including management. Many tools suffer from poor reporting and workflow capabilities that can’t be customized and are either too granular (e.g. at the line level) or too high level to be useful. Tools should be customizable, and because reporting needs evolve, it is important to employ a tool that supports multiple types of outputs, including PDF, Word, HTML, and XML for integration with other systems.
CONCLUSION

The pressure to deliver more software faster has never been greater, and the risks of delivering unreliable and insecure embedded software have profound business implications. Automated static code analysis can improve the scale and accuracy of testing efforts while saving time and lowering costs. However, static analysis testing by itself cannot meet all the challenges facing development organizations. In practice, static analysis testing is most effective when combined with best practice policies and processes and when combined with latter-stage dynamic and functional tests.

When selecting a static code analysis solution, evaluate the requirements that are common to most organizations, including integration with existing platforms and processes, test accuracy, actionable diagnostic output, ease of use, standards compliance checking, and reporting. Identify the requirements that are most important to your organization, and then evaluate the available solutions for their ability to fulfill those requirements.

Once you’ve selected a solution, motivate the transition by fully explaining to all stakeholders the goals and expected benefits of adopting the new tool or platform. Follow best practices to maximize those benefits, minimize potential drawbacks, and improve your organization’s ability to deliver embedded software that is more reliable, safe, and secure.
DETECT, ENFORCE AND MEASURE

Since 1985, PRQA has pioneered software coding governance in the automotive, aerospace, transport, finance, medical device and energy industries. Supporting both small start-ups and globally recognized brands, we provide sophisticated code analysis, robust defect detection and enforcement of both bespoke and industry coding standards through functional integrity and application security/safety.

PRQA's industry-leading solutions, QA·C, QA·C++, QA·J and QA·C# offer the most meticulous static analysis of commonly used programming languages. Innovations such as multi-threading and resource analysis (MTR) complement this with refined multi-thread inspection of code streams. Used locally or centrally deployed via the Quality Management System QA·Verify, we enable early find/fix at the desktop and on the server side complete control, visibility and history to the decision maker.

ISO 9001 and TickiT certified.

www.programmingresearch.com