



# Introduction

To keep pace with ever-increasing customer demands on software functionality and time-to-market expectations, software developers have had to evolve the way they develop code to be both faster and higher quality. As part of this trend, the Waterfall method of software development began to give way in the late 1990s to a more lightweight method of software development: **Agile**.

The use of Agile has grown in the last decade and is still maturing. Software organizations are constantly looking for ways to improve their Agile environments, and minimizing software bugs is one area of focus. ***This paper will demonstrate that several of the core principles of Agile cannot be fully realized without implementing a repeatable process for ensuring code that is as bug-free as possible.*** The approach recommended in this paper is the use of automated source code analysis (SCA) technology to locate and describe areas of weakness in software source code, such as security vulnerabilities, logic errors, implementation defects, concurrency violations, rare boundary conditions, or any number of other types of problem-causing code.

After providing brief overviews of Agile and SCA, and discussing the importance of bug-free code in enabling Agile development, this paper demonstrates how key elements of SCA enhance the Agile development processes and empower Agile teams. You will learn the relationship between bug-free code and Agile development, as well as how to deploy SCA tools seamlessly into your Agile development process to ensure that it runs at peak optimization.

## // AGILE DEVELOPMENT – A BRIEF OVERVIEW

Simply put, Agile software development is an approach that provides flexibility to accommodate **continuous change** throughout the software development cycle. It stresses rapid delivery of working software, empowerment of developers, and emphasizes collaboration between developers and the rest of the team, including business people.

Agile contrasts with the still-popular Waterfall development approach, which is front-end loaded with comprehensive scope and requirements definitions, and which employs clear, consecutive hand-offs from requirements definition to design to coding and then to quality assurance. In contrast, Agile incorporates a continuous stream of requirements gathering that continues throughout development. Business people are involved early and often throughout the release cycle, ensuring that the software being developed meets the true needs of both the end-user and the business. Change to the requirements and to the overall feature set is expected to occur as outside opportunities or threats arise.

In short, Agile fully embraces change and Agile teams are structured in such a way that they can receive and act on constant feedback provided by the build process, by other developers, from QA, and from business stakeholders.

Agile is based upon a number of guiding principles that all Agile teams follow. For the purposes of this discussion, three principles – or values – are of particular interest:

- » Quality software development
- » Iterative flexibility
- » Continuous improvement

### **Quality Software Development**

The primary focus of Agile development is to enable the development of quality software that satisfies a customer need – i.e. provides a functioning feature or capability – within a specific period of time (typically no more than a few weeks) called an “iteration”. In theory, a product developed in an Agile environment could be market-ready after each iteration.

Delivering a series of market-ready products, each in just weeks, demands that a rigorous quality process be built into the Agile development cycle. Each iteration must be fully developed: tested, defect-free, and complete with documentation.

### **Iterative Flexibility**

With a focus on speed and nimbleness, Agile is open to changes that inevitably arise throughout the development cycle. The iterative process is flexible, based on an understanding that original requirements may (or will likely) need to change due to customer demand, market conditions, or other reasons. Because business users are involved throughout the process, and because each iteration is short, new requirements can be introduced and prioritized very quickly.

### **Continuous Improvement**

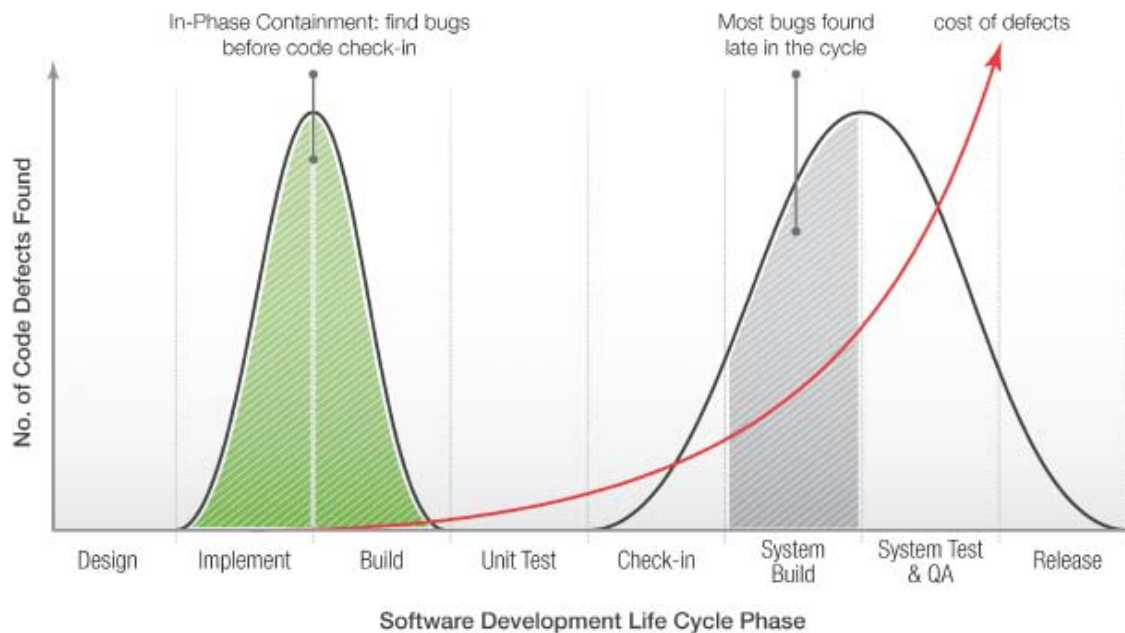
An Agile environment provides developers with an opportunity to learn new skills and to exercise greater autonomy to do their jobs. The iterative framework is empowering because it enables continuous improvement, with testing/quality assurance occurring as part of the iterative process, rather than only periodically or at the end of a long process when it is often difficult or not cost effective to fix coding defects or to incorporate lessons learned along the way. Agile also makes the testing and QA process transparent to the developers who originate the code, further contributing to their learning and facilitating future improvements and coding efficiencies.

## // BUG-FREE CODE GREASES AGILE'S WHEELS

One of the development principles put forth in the Agile Manifesto (widely recognized as the de facto definition of Agile) states that, "Working software is the primary measure of progress." Working software implies software that is free of issues that break builds, cause unexpected behaviour, or which do not meet the product's requirements, as well as mundane programming defects (a.k.a. "bugs").

This principle is not unique to Agile – many software development processes, including formal ones such as CMMI and Six Sigma, encourage the creation of bug-free code as a fundamental principle. These processes encourage in-phase bug containment – the practise of preventing bugs from being passed downstream from the phase in which they are created. Agile also implicitly emphasizes in-phase bug containment. Given its focus on short iterations, Agile processes must ensure that any potential software degradations are quickly identified and corrected so that the whole team can move on to the next iteration – all while creating functionally complete, working software.

### The Costs of Bug Containment



Even within an iteration, Agile teams apply this philosophy through continuous integrations and regressions. While this practise effectively addresses defects that can break builds or regression test suites, it is not as effective in cleaning up many of the most common types of programming bugs, which generally fall into these broad categories:

- » Memory and resource management
- » Program data management
- » Buffer overflows
- » Un-validated user input
- » Vulnerable coding practices
- » Concurrency violations
- » A variety of longer term maintenance issues

Bug-filled code creates downstream risk both within an iteration or in subsequent iterations, while clean code facilitates speed and agility in development. **Supporting this need for in-phase containment and bug-free code requires a solution that allows developers to take control of the bug identification and removal process while enhancing collaboration amongst all developers to resolve bugs as early in the process as possible.**

## // ADOPTING BUG-HUNTING TOOLS WHILE STAYING AGILE

While the Agile Manifesto's principle of "individuals and interactions over processes and tools" seems to de-emphasize the need for tools, Agile teams use many tools to support their development – including software configuration management tools, build management tools, requirements tracking tools, testing tools, project management tools, and more.

Most of the testing tools that are well-known to Agile teams focus on functional testing, unit testing, and build quality, with less of an emphasis on source code verification to identify non-functional errors such as programming bugs. This is likely due to the overly cumbersome options traditionally available to find these types of bugs; Agile teams don't have time for lengthy code review meetings or to profile their code for days to find a memory leak. Even unit testing, a stalwart of Agile testing, requires the creation and management of test cases to support a developer's bug hunting.

Agile teams must strike the right balance between using tools to ensure working, quality software while also exercising caution so that the adoption of tools does not hinder the individual interaction required by Agile. Smaller Agile teams may have an easier time striking this balance and may decide to manage the project at hand with nothing more than a large bulletin board and color-coded cue cards. Teams working on larger projects, however, generally employ tools to ensure that they have the best opportunity for success.

### **Automating Bug Detection: Source Code Analysis in an Agile World**

SCA is a bug-detection solution that requires no test cases, is fully automated, and fits well with milestones typically found in an Agile process. SCA technology has grown in popularity and is becoming a mainstream option for professional software developers to reduce the number of bugs in their code while also reducing costs and keeping software development on track.

The underlying technology associated with SCA is called Static Analysis and the current generation of technology solutions is capable of providing sophisticated, high-value analysis that will locate and describe areas of weakness in software source code – such as memory and resource management, program data management, buffer overflows, un-validated user input, vulnerable coding practices, concurrency violations, and a variety of longer term maintenance issues. SCA is distinct from traditional dynamic analysis techniques, such as unit or penetration tests, because the work is performed at build time using only the source code of the program or module in question. The results reported are therefore generated from a complete view of every possible execution path, rather than some aspect of a limited, observed runtime behaviour.

Since SCA is essentially a build-time analysis, it is most effectively used as a build milestone activity when individual developers or development teams run their builds – either at the integration-build level or the developer-build level.

### **Integration Build (a.k.a. system build, project build)**

Today's SCA tools go well beyond the syntactical and semantic analyses commonly associated with source code analysis. Good SCA technology today can be expected to include sophisticated inter-procedural control and data-flow analysis with advanced approaches for pruning false paths, estimating the values that variables will assume, and simulating potential runtime behaviour. The complexity of this type of analysis across a large system with millions of lines of code and an essentially unlimited number of potentially feasible code paths to consider is not trivial.

To make it all work, and to reduce the number of "false positives" (bugs that the tool incorrectly reports) and false negatives (bugs that the tool misses), vendors naturally provide integration with a project's system build – whether it is make, ant, Visual Studio, or other continuous integration tools such as Electric Cloud and BuildForge – to generate a complete view of the entire source code base. Of course, the downside to running SCA exclusively at the integration build level is that bugs created at the desktop are exposed to the main code stream and can impact other members of the team – both fellow developers and the QA team. When bugs are found downstream – even in a continuous integration context where integration builds are run much more frequently – an additional bug triaging process needs to be put in place to notify the developer of the error (by email, web reports, etc.). This adds more workflow and process, which is contrary to the spirit of Agile development.

Clearly, the solution is to push SCA to the developer desktop so that it can run in conjunction with a developer's build, even prior to him/her running unit tests.

## Developer Build (a.k.a. personal build, sandbox build)

SCA at the developer desktop offers a big payoff for any organization adopting this technology, in particular an Agile team. If most bugs can be found prior to code check-in, organizations will achieve in-phase bug containment, reducing the number of bugs in the main code stream or integration build. This allows QA to be more efficient by focusing on being a customer advocate and ultimately producing higher quality software – earlier.

For SCA to operate at the developer desktop, it must be delivered within the developer's natural work environment (i.e. a favourite IDE, text editor or command line) and the analysis must be every bit as accurate and intelligent as the centralized analysis that benefits from a view of the entire code stream. Code check-in (or commit) is an important milestone in Agile and many organizations operating in a continuous integration context have a series of gates (smoke tests, unit tests, etc) that the developer must pass in order for him/her to check-in code. SCA should be added to this series of pre check-in quality gates.

If organizations embrace SCA with the goal of in-phase bug containment, the three principles of Agile outlined at the beginning of this paper will be fully realized.

## Bug Free for Quality

Software bugs are more than a nuisance. Serious bugs can cause downstream inefficiencies, product recalls, or field disasters. Source code analysis can automatically find serious bugs in your code, such as NULL pointer dereferences and memory management issues that can lead to a system crash, or buffer overflows and un-validated user inputs that make a system susceptible to exploit by hackers. Removing these issues prior to shipping a product is critical and the earlier they are identified the better. This prevents critical issues from being passed to QA within an iteration, or passed from iteration to iteration, both of which greatly increase the risk of shipping buggy software. In addition, if bugs are dramatically reduced before code check-in, this will ensure they never impact the main code stream and will facilitate the testing and QA process. **With few bugs to find or report on, testers are able to focus instead on running functional and performance tests to ensure the product is customer- and market-ready.**

## Bug-Free Flexibility

To adequately test and assure the quality of software developed in Agile environments, the testing process must also be iterative and accommodate frequent change in requirements. Thus, there is little room to address programming bugs in the testing phase. If QA discovers these bugs, unacceptable drag in the development cycle is created –testers report the bug list back to the developers who must set aside their current work to shift back to the frame of mind they were in when working on the original code. By enabling developers to check-in bug-free code, this dramatically reduces or eliminates the time-consuming “rinse and repeat” cycle of check-in, find bugs, debug, and then rework. By using SCA tools in an Agile environment, developers can spend less time fixing reported bugs and more time writing new and innovative software. Within this context, **developers have the flexibility to control the quality and security of the code they create during new development, before the integration build is performed.**

## Bug Free for Continuous Improvement

It doesn't matter how an iteration appears to be progressing from a feature development standpoint – if the development team's commitment to quality code can not be measured, the project is by default accumulating significant downstream risk with each and every build and iteration. In a fast-paced and fluid development environment, Agile teams need to have strong, automated measurement capabilities in place, such as:

- » Measure and track bug fix rate at the developer build;
- » Identify what bugs are leaking to the integration build;
- » Establish nightly build quality milestones (or any other frequency over and above the continuous build schedule); and,
- » Track which teams or components are improving over time.

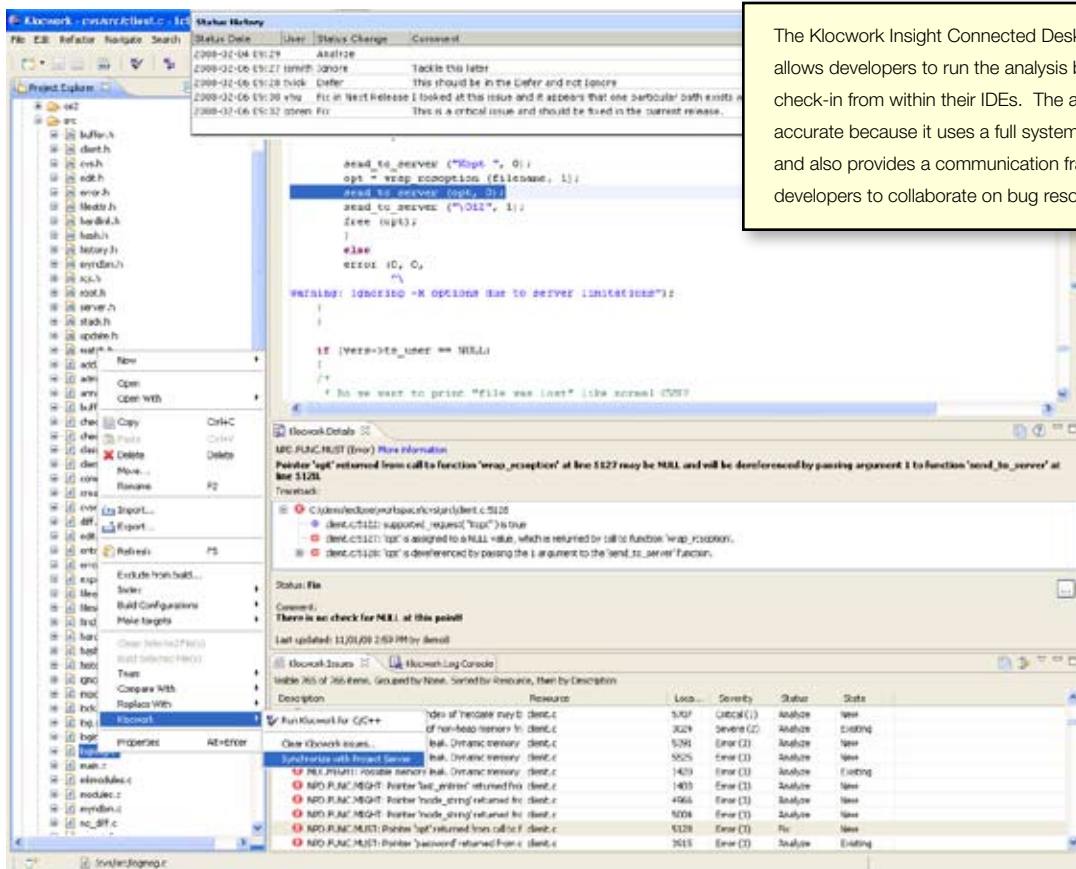
The answers to these questions allow Agile teams to implement targeted continuous improvement programs that quickly identify where help or training is needed so that all members of the development team can submit clean code into the integration build. **Measuring and tracking quality from the bottom up is necessary to know if iteration plans are any good and if a shippable product will be available at the end of an iteration.**

## // KLOCWORK® INSIGHT FOR AGILE DEVELOPMENT

Klocwork Insight is a leading source code analysis product and the first of its kind: a next generation SCA tool that allows developers to take control of the analysis process while also benefiting from the accuracy and value of centralized analysis – with none of the downstream auditing that previous techniques required.<sup>1</sup> This allows developers to uncover bugs as code is produced and empowers them to recognize, address, and communicate bugs across teams so they can fix them before they impact the code stream or reach QA.

**Because Klocwork Insight provides connected desktop analysis within the context of the entire code base, its an ideal tool in Agile development environments.** Klocwork Insight supports the key principles of Agile development with these particular capabilities:

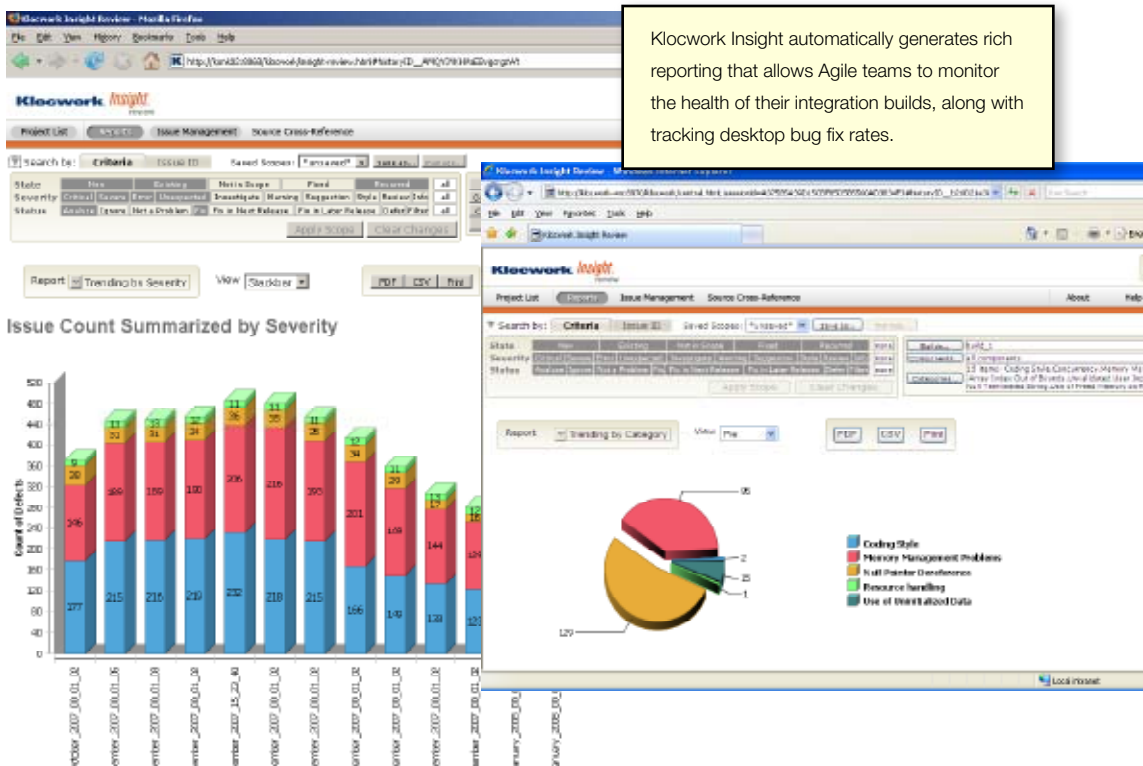
**Connected Desktop Analysis** – Klocwork can be integrated with your development both at the developer desktop and the system build. Uniquely, Klocwork Insight provides a patent-pending Connected Desktop Analysis that enables developers to run fast, local source code analysis at their desktops and which leverages the full system context for accurate analysis, provides collaborative bug mitigation between other developers to ensure nobody duplicates work on the same bug, and gives all members of the team an up-to-date view of the bugs in the code stream. Without a connected local analysis that understands the full system context, SCA at the developer desktop will experience a high-rate of false positives. That can cause developers to ignore the tool, and in an Agile context will create unnecessary inefficiencies and overhead.



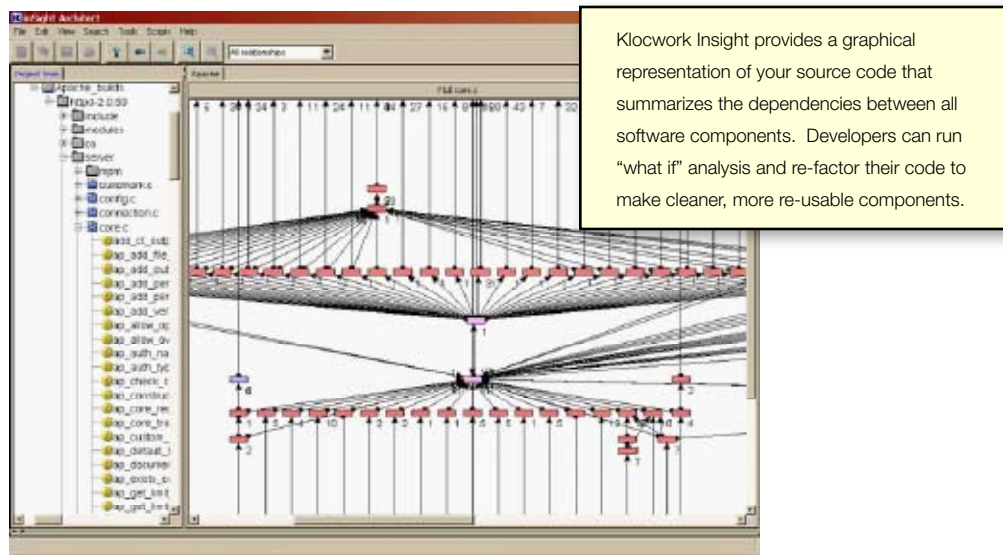
The Klocwork Insight Connected Desktop Analysis allows developers to run the analysis before code check-in from within their IDEs. The analysis is accurate because it uses a full system context and also provides a communication framework for developers to collaborate on bug resolution.

<sup>1</sup> For a comprehensive discussion of the evolution of source code analysis, download the Klocwork paper entitled Klocwork Insight – The Next Generation of Source Code Analysis, available at [www.klocwork.com/company/resources](http://www.klocwork.com/company/resources).

**Software Metrics & Reporting** – Klocwork Insight provides a robust suite of more than 100 objective and actionable product metrics, which are derived directly from your software code. Klocwork’s software metrics capabilities are designed to answer key questions about your software development process. For example, a key question for Agile is whether bugs are being found and fixed at the developer desktop, or whether they are leaking to the integration build. Klocwork Insight automatically aggregates information about what is being found and fixed at the desktop even though it is never propagated into the source stream. This unique capability allows teams to better understand the bug reduction activity that is happening before code check-in, generating a bottom-up view of how well your containment is working. This – combined with custom ownership models that allow metrics to be organized by people, groups, geography, components, and any other attribute that works for your organization – allows teams early in an iteration to identify the areas of greatest risk within their code base.



**Refactoring** – Klocwork Insight supports refactoring, an Agile technique for restructuring a unit of code to simplify its design and operation without changing its functionality. Klocwork provides a graphical representation of source code, displaying architectural components, blocks, and sub-blocks that comprise the system, as well as the relationships and dependencies between these components. Klocwork Insight can therefore assist with impact analysis or code discovery. The architecture of the code can also be manipulated to perform “what if” scenarios in order to create a more maintainable, reusable and less complex system.



## Overall Cost Reduction

Agile development organizations frequently work within a cost-sensitive business environment. Klocwork Insight's breadth of analysis allows organizations to reduce costs by deploying a single solution that provides these capabilities:

- » Multi-language support for C, C++ and Java;
- » Critical bug detection;
- » Security vulnerability detection;
- » Architectural analysis and re-engineering; and,
- » 100+ software metrics generation and management.

It also provides development teams with the opportunity to directly impact the true cost of software by virtually eliminating the risks associated with bugs that find their way into the field.

## // CONCLUSION

The ubiquitous nature of software today, coupled with the pressure to rapidly develop market-ready features and products in just weeks, has led to two related phenomena:

- » The widespread adoption of Agile software development principles; and,
- » The adoption of various tools by Agile teams designed to help streamline and de-risk development projects.

One of the most important types of tools that an Agile team can deploy is one that aids in writing better-quality code. Source code analysis tools provide an automated method to detect a significant number of software bugs or security vulnerabilities right at the developer's desktop – before any code is delivered to the integration build or testing team. This minimizes project drag caused by rework and enables Agile to run more efficiently: developers spend their time writing innovative code, while testing teams spend their time testing how the features of the project work rather than uncovering mundane code issues and retesting these again and again.

SCA may be right for your Agile team, particularly if you are finding large numbers of quality issues or security vulnerabilities and have to undertake a significant amount of rework as a result.

- » **Implementing SCA within your Agile environment does not have to be disruptive.** You can start small and analyze only a small project or a portion of a project. Compare the results against a similar project where SCA tools were not used. Our experience suggests that you will find opportunities to save significant time and money by using SCA in Agile development.
- » **Klocwork itself is an Agile development shop.** We run Klocwork Insight on Klocwork code and know first hand how our source code analysis tools operate in this environment. When you deploy Klocwork solutions in your project, you will benefit from our wealth of expertise, experience, and tools to help maximize your project's success.

## About the Author

Todd Landry, a Senior Product Manager at Klocwork, is responsible for guiding product direction and ensuring its fit with customer's preferred development processes. With more than 12 years of experience in software product management, he has worked with numerous Agile teams and projects. Todd is a Professional Engineer and a Certified Scrum Product Owner.

## About Klocwork

Klocwork is an enterprise software company providing automated source code analysis software products that automate security vulnerability and quality risk assessment, remediation and measurement for C, C++ and Java software. More than 300 organizations have integrated Klocwork's automated source code analysis tools into their software development process in order to ensure their code is free of mission-critical flaws while freeing their developers to focus on what they do best – innovate. Contact Klocwork for more information at [www.klocwork.com](http://www.klocwork.com) or [info@klocwork.com](mailto:info@klocwork.com).



© Copyright Klocwork Inc. 2008 · All Rights Reserved

IN THE UNITED STATES:  
8 New England Executive Park  
Suite 180  
Burlington MA 01803

IN CANADA:  
30 Edgewater Street  
Suite 114  
Ottawa ON K2L 1V8

1-866-556-2967  
1-866-KLOCWORK  
[WWW.KLOCWORK.COM](http://WWW.KLOCWORK.COM)

**Klocwork**