Increasingly, software powers the modern world. From the networks that connect us and enable ecommerce; to embedded systems such as robotics and medical diagnostics equipment; to consumer electronics, appliances, cell phones, and automobiles, software automates our routine tasks and powers the technologies of our daily lives.

When the software in these systems fails — whether because of defects that result in reliability problems or security vulnerabilities that expose the software to malicious misuse — the costs can be very high. Software defects cost the U.S. economy some $60 billion a year, amounting to 0.6% of the U.S. gross domestic product (GDP), according to a National Institute of Standards and Technology (NIST) study released in 2002. By NIST’s own estimate, some $22 billion of these sunk costs could be recovered through better software testing. And this is only the tip of the iceberg: NIST’s costs don’t factor in the business costs of failure — such as damage to brand and customer loyalty, loss of revenue and market share, and even (in safety-critical systems) loss of life — that represent the “collateral damage” of software reliability and security problems.

The need to ensure the reliability and security of today’s software systems is reflected in IT spending growth. In 2004, the automated software quality tools market grew 11.2% to $1.1 billion in software revenue. Enterprises are spending tens of billions of dollars each year on security — including software, hardware, and services — to protect themselves from hackers and malicious insiders. Even so, as IDC’s 2004 Enterprise Security Survey showed, attacks are still on the rise and still successful: 75% of survey respondents who knew of attacks reported at least one successful security breach.

Hardware and electronics engineers have relied for a long time on metrics and well-defined testing protocols to ensure a high degree of reliability for fielded products. These metrics and protocols are critical to their ability to remain competitive; after all, quality problems can — and sometimes do — put manufacturers out of business. Software development, on the other hand, has often been treated more like an art than an engineering discipline. This situation is changing. The need to ensure a high degree of reliability and security in today’s software-enabled systems is driving interest in, and adoption of, process improvement frameworks, such as the Software Engineering Institute’s CMMI, that make quality the outcome of best practices in software and systems engineering. There is growing awareness that
quality must be designed into and built into software from the start — that is, that quality is not something that can be added late in the software development life cycle. This awareness is driving the growth in uptake of developer-focused automated software quality tools that help software developers design and write reliable, secure code.

As the NIST study shows, there is plenty of room for improvement in the software quality department. According to the study, software development groups may spend as much as 80% of their software development budget identifying and correcting defects — budget that could be much better spent on new features and functionality that deliver competitive advantage.

This white paper examines how software development organizations can improve the quality of the software they deliver, and tangibly improve their overall software development process, by incorporating static analysis tools into their routine development processes. The results are higher quality, more robust and more maintainable code, lower risk of failure because of software reliability problems and security vulnerabilities, a more predictable software development process, and lower overall life-cycle costs.

SOFTWARE QUALITY AND DEVELOPER NEEDS

The cost to find and fix defects and security vulnerabilities increases dramatically the further the defect is from the developer's desktop. Research conducted at IBM in the late 1980s showed that the cost of finding and fixing a defect at the system testing phase was about 15 times higher than the cost of finding and fixing that same defect during the design phase; subsequent research suggests the cost could be as much as 90 times higher. Finding and fixing the same bug in the field, after the application has been delivered to customers, can be 50–200 times more expensive than finding it during the design and coding phases. Although actual cost ratios will vary from one software development organization to another, generally the cost curve will resemble that shown in Figure 1.
Software development organizations that rely solely on testing to ensure quality — what is sometimes termed a “fix on failure” approach — put themselves at considerably greater risk of cost and budget overruns because problems that are discovered late in the life cycle — after the software has been designed and coded — often entail significant rework (and sometimes redesign) to correct. These teams are essentially forced to go through several extra iterations of the software development life cycle in order to meet their quality objectives.

To be sure, software testing is an essential component of the software engineering discipline. A mature organization will incorporate unit, integration, system, and user acceptance testing into its life-cycle processes. But testing isn't enough to build in quality: It occurs at least one step too late in the life cycle to tangibly impact the quality of source code as it is created. Although testing is critical to validate that the software meets its functional and performance requirements, and to find problems that arise from interactions between the source code that the software development team creates and the rest of the software stack, testing provides no insight into the quality of the source code itself — how it is architected, whether it complies with company-standard coding rules, whether it makes calls to "unsafe" functions, and so on.

Although many defects — including some that can be exploited as security vulnerabilities — are in fact caught during testing, testing is a relatively low-yield technique for finding defects in source code. Code inspections are much better at turning up the majority of coding errors. The problem is that no team can effectively

---

**FIGURE 1**

*Escalating Cost to Find and Fix a Defect or Design Flaw as It Is Discovered Late in the Life Cycle*
implement manual code inspections on its software asset — and certainly not for
every release of its software asset — given the complexity of today's software
systems, the amount of legacy code that is continually integrated with new source
code to create new products, and the rate of change of source code as it is enhanced
with new features and functions. Code inspection is thus a process that desperately
needs automation, and that's where static analysis tools come into play.

Risks and Costs of Poor Software Security

Software security is an increasingly important dimension of software quality. The press
is full of stories about hackers breaking into what were thought to be secure systems
and running off with thousands of credit card numbers, social security numbers, or other
sensitive and confidential information. Similarly, we regularly read about successful
attacks that bring down real-time, customer-facing transaction systems.

Over the past decade, we've seen tremendous growth in "edge of the enterprise"
applications that directly connect customers, partners, and suppliers to back-office
transaction systems over the Web via a browser. These Web-based applications
have enabled new products and services that generate new revenue and drive down
costs, but they have also become targets for hackers. As companies have secured
their networks and servers with strong perimeter defenses, hackers have begun to
realize that their easiest means of attack is at the application layer.

Many security vulnerabilities are simply defects that can be exploited by the adroit
hacker — buffer overruns, for example. From a risk management perspective,
however, defects in software systems that are exposed over the Web are major
threats because the likelihood that a particular defect will be deliberately exploited
increases. Managing — and mitigating — software security risk is another important
driver for the growth in adoption of static analysis tools.

Maintainability and Total Cost of Ownership

Because software is highly malleable, it has sometimes been thought of as
"disposable." We've often heard the saying "There's never time to do it right, but
there's always time to do it over." Nothing could be further from the truth: Software
development teams rarely get the time to do a major rewrite. Increasingly, new
software systems must leverage and extend legacy assets with new capabilities to
meet ever-shortened cycle times. The value of the software asset depends to a large
extent on the length of its lifetime and whether it is maintainable and extensible.

Understanding the software asset is the first step. This can be a difficult process
when the original team members have left the project, taking their intimate knowledge
of the code base with them. Without a good understanding of the software's
architecture, the existing team will find it very difficult to gauge the architecture's
quality, accurately estimate the impact of potential changes to the architecture, or
prioritize potential refactoring projects for maximum benefit. Gaining a good
understanding of the software asset is another important goal of code inspections and
one that static analysis tools are best positioned to address.
Static analysis is not a new technology. In fact, static analysis tools have been available for many years in university and other high-tech research environments. Early static analysis tools, however, were plagued by a poor "signal-to-noise" ratio: For every legitimate defect they caught, they flagged dozens of "false positives." Also, they weren't scalable; they couldn't be used to scan large code bases. Moreover, performance was a problem. The tools were too slow to be incorporated into the routine work of developers without slowing down the overall development process. Early optimism about the ability of these tools to automate code inspections and significantly improve code quality gave way to disappointment, and market enthusiasm cooled.

Over the past few years, enterprising vendors have brought to market commercial-grade static analysis tools that address these problems. The best of these tools can comprehensively analyze several million lines of code to identify defects and/or security vulnerabilities, and even pinpoint design flaws, with a very good "signal-to-noise" ratio, thanks to sophisticated heuristics. And, thanks to dramatic improvements in performance, static analysis can now be incorporated into the developer's routine coding process as an integral and automated activity. Static analysis provides developers with the "instant feedback" they need to quickly address defects and security vulnerabilities "in phase" — while they are still working on the code and it is fresh in their minds — and it prevents defects and vulnerabilities from making their way further downstream in the software development process where they become much more expensive to identify and remediate.

What Differentiates Vendor Solutions?

Vendors are leveraging static analysis in different ways, according to the problem domain they are targeting. Customer needs will dictate the vendor shortlist. Some vendors focus on a specific programming language such as C or C++, whereas others analyze applications constructed in multiple programming languages, for example, both C/C++ and Java. Software development organizations using multiple programming languages will want to investigate multilanguage solutions so that they can perform interprocedural analysis on a mixed code base.

Similarly, some vendors focus exclusively on software quality issues (that is, on finding defects in code), while others are focused on security (that is, on finding exploitable defects and flaws). Some vendors, such as Klocwork, provide both defect and vulnerability analysis capabilities. Again, customer needs will dictate the vendor shortlist. For customers in the embedded software market, for example, software reliability (finding defects) is critical, but software security may not be an issue. (As one of Klocwork's military/aerospace customers told us, "We develop robotic space flight software and haven't met any alien hackers yet!") In the financial services industry, on the other hand, ensuring both the reliability and the security of Web-based applications is a critical part of managing business risk.
Vendors also differ in the breadth of their solutions. Some focus exclusively on pinpointing defects, whereas others leverage their static analysis results to provide additional quality metrics and guidance. For example, Klocwork leverages its static analysis capabilities into architectural analysis to identify design flaws and support refactoring efforts, and it provides advanced, derivative metrics that help management assess and prioritize areas of risk in the software. Architectural analysis allows teams to visualize the impact of their independent code changes on the overall software system. This capability helps teams catch defects that arise from complex interdependencies that would otherwise be hard to spot and helps lower the defect injection rate in larger, more complex systems.

Finally, vendor solutions are maturing in terms of their reporting capabilities. Solutions (such as Klocwork's) that capture the results of successive analyses in a central repository can provide valuable metrics on code quality over time and highlight areas of the code that pose reliability and/or security risk that should be rewritten or rearchitected. Trend reporting provides management with significantly better visibility into project health and status and helps to ensure that projects stay on schedule and on budget.

As the market for static analysis tools continues to grow, IDC expects to see vendors further expand their footprints.

**Klocwork's Solution**

Klocwork ([www.klocwork.com](http://www.klocwork.com)) was founded in 2001 by software development veterans who had created software quality tools for embedded software development at Nortel Networks. Recognizing the broad applicability of these tools, not just for embedded but also for enterprise software development, Nortel provided seed funding for the company and spun it out. Klocwork has received funding from venture capital firms Mobius Venture Capital, Pequot Ventures, and U.S. Venture Partners. Cisco is also an investor in the company.

Klocwork has more than 65 customers in North America and Europe, including many Fortune 100 companies, primarily from high-tech industries such as telecom and network equipment, military/aerospace, and computer software. Representative customers include Alcatel, Motorola, Lucent, Avaya, Panasonic, Ericsson Siemens, Raytheon, Boeing, Northrop Grumman, NASA, IBM, AOL, and, of course, Cisco and Nortel. Klocwork's strategic alliance partners include IBM Rational, Wind River, Telelogic, the Eclipse Foundation, and Security Innovation.

**Klocwork K7 Development Edition**

In contrast to vendors that have focused solely on static analysis as a means of quickly identifying defects, Klocwork aims to provide broader support for development teams and management around improving overall code quality and the software process itself. Klocwork K7 Development Edition consists of an integrated, comprehensive set of code analysis and reporting facilities that provides a multidimensional view of the quality of the source code asset (see Figure 2).
Key capabilities of Klocwork K7 Development Edition include:

- **Static code analysis.** Klocwork’s static analysis identifies defects and security vulnerabilities in C, C++, and Java. Interprocedural analysis is an important capability because so many of the hard-to-catch defects that arise in complex applications occur when different programmers collaborate on related pieces of code. K7’s static analysis rules catch C/C++ problems (such as null pointer dereferences, buffer overflows, memory leaks, uninitialized stack and heap variables, and programming errors related to freed memory) and Java problems (including some of the same issues that arise in C/C++, plus tainted data, race conditions, SQL injection, XSS/cross-site scripting, and interprocedural resource leaks). Klocwork K7’s static analysis capabilities are extensible; therefore, customers can add their own customized rules to enforce company-specific coding standards and security policies. This capability is a boon for software development organizations that need to bring new programmers up to speed quickly, whether they are internal or external team members, and consequently helps mitigate risk when software development activities are outsourced.

K7 is integrated with popular IDEs including Eclipse, IBM Rational Application Developer, and Microsoft Visual Studio, making it a "one-click" operation to perform static analysis on a piece of code. Klocwork has also announced that K7 development tools are integrated into Wind River's Workbench IDE. In all of these integrations, Klocwork automatically directs the developer to the line of code when an error is discovered.
**Architectural analysis.** Klocwork K7 analyzes the code complexity and architectural quality of a complete software system. This capability helps teams identify design flaws and prioritize areas of the code that should be rewritten for better reliability and maintainability. Architectural analysis is especially helpful to teams that need to quickly come up to speed and understand a codebase they didn't create. Code complexity is an important metric for the overall quality of the software asset and one that is often missing from the "quality dashboard."

**Reporting.** Klocwork provides comprehensive reporting, including historical trend analysis, for a detailed view of code quality over time. K7 also provides integrated build-over-build defect management so that teams have up-to-date information on the status of all defects and security vulnerabilities (i.e., whether they are new or recurring or whether they have been fixed) and can prioritize and schedule fixes. Klocwork K7 stores the results of all analyses in a central repository and can also feed quality metrics and trend data to other dashboards via XML. K7's reporting capabilities provide a wealth of information to guide process improvement initiatives; for example, managers can compare quality metrics across projects or across teams and identify opportunities for further developer training.

**Additional Klocwork Packagings Target Specific Customer Needs**

Klocwork offers three "subset" editions of its K7 Development Edition that target organizations with more limited, specific needs:

- **Defect Discovery Edition** provides static analysis capabilities for the identification of software quality defects that could impact operational reliability.
- **Security Vulnerabilities Edition** provides static analysis for the identification of security vulnerabilities (i.e., exploitable defects and flaws).
- **Defects & Security Edition** combines the two preceding offerings and adds additional analysis capabilities to identify "include" and interface issues.

Klocwork prices its solution by "line of code" — that is, according to the size of the customer's code base. It is a less familiar but innovative pricing scheme that accommodates customers of different sizes and allows customers to engage on a project-by-project basis. Klocwork also provides consulting and training services to jump-start customer implementation. In addition, the company provides code analysis services (including architectural reviews, defect citing, and build-over-build metric reports) to assist customers with due diligence on potential software acquisitions.

**Benefits of Klocwork's Approach**

Putting tools such as Klocwork K7 into the hands of developers can bring a severalfold return on investment. The benefits of Klocwork's integrated, multidimensional view of software quality include:

- **Lower risk of failure for deployed solutions.** Higher-quality source code means a much lower risk of defects or security vulnerabilities making it into production systems.
Higher development team productivity. Teams spend less time finding and fixing defects and on the rework occasioned by design flaws.

Tighter project control. Better visibility into project health and status helps eliminate the late life-cycle surprises that inevitably result in project delay.

Increased software asset value. Improving the quality of the code — and its architecture — makes the software asset more flexible, more maintainable, and easier to adapt as business needs dictate.

Incremental investment. Customers target their areas of greatest pain — whether defects or security vulnerabilities — then grow into the full K7 solution.

CASE STUDIES

Mentor Graphics

Mentor Graphics (www.mentor.com, NASDAQ: MENT), founded in 1981 in Wilsonville, Oregon, is a leading supplier of electronic design automation (EDA) solutions with approximately 3,850 employees and $700 million in trailing twelve-month revenue. The company has 48 sales offices around the world, and its large, geographically distributed development team spans 28 engineering sites. Mentor Graphics solutions for PCB systems and IC design helps customers improve time-to-market, reduce costs and use the latest technologies in their competitive products. According to Kevin Pendleton, Director of Quality and Support Systems for the Systems Design Division at Mentor Graphics, "Mentor's code base is a complex combination of new technology, older technology, and acquired technology. We have different legacies for these technologies in terms of the languages we support, build environments, compilers, and so forth. Our challenge is to pull these different technologies together in order to provide integrated solutions for customers at every level of the PCB market, from entry-level shrink-wrap solutions through solutions for global, enterprise customers." A solid understanding of its software architecture is key to Mentor's ability to flexibly leverage its software assets into this broad array of solutions.
Mentor Graphics began to investigate Klocwork at the end of 2003. A quick overnight analysis of a small subset of Mentor's source code turned up legitimate defects in relatively mature code, and they were easily addressed because Klocwork points the developer directly to the line of code with the problem. A subsequent six-month pilot project focused on both defect identification and architectural analysis. "We saw an immediate return on our investment. Klocwork identified about 1,000 issues, and we were able to address 500 of them in our next release. All of them were found directly by the tool, without any QA team effort required," says Pendleton.

Mentor Graphics is integrating Klocwork into its build processes on Linux, Unix, and Windows and into the engineering desktop so that every time an engineer checks in code, a Klocwork analysis will run. This way, programmers can address any issues before their code enters the build process. Most of Mentor Graphics' source code is C and C++, but the company is doing more and more with Java — another reason why Klocwork's solution is a good long-term fit.

Mentor Graphics is also leveraging Klocwork's reporting capabilities to design a comprehensive dashboard with reports and metrics to help assess the ROI of future projects and to enable management and development team members to monitor the quality, health, and status of projects under way. "We've had good results so far," says Pendleton. "When we kicked this project off, one of the things I was concerned about was how we could measure ROI and progress over time. How many problems did we prevent from hitting the field? ROI was very quickly realized — within one release cycle. Based on our results so far, the tool is paying for itself."

**Panasonic**

Panasonic Mobile Communications Co. Ltd. ([panasonic.co.jp/pmc/products/en](http://panasonic.co.jp/pmc/products/en)), a wholly owned subsidiary of $62 billion Matsushita Electric Industrial Co. Ltd., is a leader in the mobile communications industry with approximately 3,000 employees and annual revenue of $7 billion. Panasonic Mobile Communications makes Panasonic-brand mobile handsets and wireless base stations and terminals for cellular communications networks.

In an industry of notoriously thin margins, short product life cycles, and intense pressure to innovate with new features and functions, software quality and time to market are critical competitive issues. Mobile phone manufacturers such as Panasonic must cope with a continuously changing software environment. Because the software stack in a mobile phone is so large today, mobile phone manufacturers can no longer afford to build all of the software themselves. Instead, they purchase a reference platform from the chipset manufacturer. Then, their in-house software development teams customize and extend the reference platform source code with additional features and functions to deliver competitive advantage. Andy Chappell, who heads software development for Panasonic Mobile Communications in Europe, says his team has had to work with three different reference platforms in the past 18 months.
This situation raises the software quality challenge to a whole new level. First, the team members need to quickly come up to speed on — and be able to reliably enhance and maintain — code that they did not write. Second, the software must be extremely reliable: At the typically tight margins that characterize this marketplace, a recall can devastate profits. At the same time, the software in the handset needs to be remotely maintainable and upgradeable, but with an important caveat: It must be secure from unauthorized updates. Security is a concern because the mobile phone sales channel typically discounts the price of the handset with the expectation of recouping the money on subscriber fees; phones that are sold with compromised IDs represent significant revenue loss.

The need to ensure the high quality of Panasonic's handset software and the need to accelerate the overall software development life cycle were both drivers for Panasonic's acquisition of Klocwork's solution. Panasonic's team codes in C and C++ on Unix and Windows; then, the software is moved to an embedded environment and is tested around the world on different networks. (Panasonic ships its software to about 80 countries and tests on 120 different networks.) Without automated tools, the number of test engineers required quickly spirals out of control (a project in Japan required 200 test engineers, or nearly half of the 500-person software development team).

Chappell's team started using Klocwork in the spring of 2003 and uses the product at several phases within the software development life cycle. At the beginning of the project, Panasonic uses Klocwork's architectural analysis capabilities to look at the reference platform software and understand its architecture and interfaces. Klocwork provides his team with a "complete system-level view of the software." The team also uses some of these capabilities at the back end of the project to assess the impact that a particular code change will have on the system during maintenance and bug fixing.

The team also leverages Klocwork at the developer desktop to enforce coding standards. Panasonic's developers have to bring their Klocwork reports to code review meetings. Chappell says that "tools like Klocwork free the developer to focus less on coding standards and more on what the code is actually doing."

Incorporating automated quality solutions into the development process is paying off for Panasonic. As Chappell points out, "The earlier you find the bug, the cheaper it is to fix. We're investing in tools to find the problems earlier in the process."

Cisco

Cisco Systems (www.cisco.com; NASDAQ: CSCO) is the world's largest supplier of high-performance computer internetworking systems, with $24.2 billion in trailing 12-month revenue and more than 34,000 employees worldwide. With a broad line of networking products — from routers and switches to technologies in areas such as home networking, IP telephony, optical, wireless LANs, network security, and storage networking — Cisco supplies the majority of the networking equipment used for the Internet.
Innovation in IP-based networking continues to drive Cisco's growth, and software is a critical component of every Cisco product. Software performance and reliability, time to market, and the ability to evolve its product line while maintaining compatibility are all key to its competitive advantage.

Cisco's software development organization has been using static analysis for about five years to automate code inspections. A year and a half ago, the company evaluated several tools and chose Klocwork for its comprehensive solution and maturity. Cisco uses Klocwork's tools in two ways.

First, Cisco runs Klocwork static analysis checks prior to code commit. Developers must look at their static analysis warnings and decide what needs to be fixed before proceeding. According to Mike Turnlund, Director of Engineering at Cisco, "We use Klocwork as an inline process monitor." Most of Cisco's code is C/C++, but its network management applications are written in Java, and Cisco is also investigating Klocwork's new Java tools.

Second, Cisco relies on Klocwork's architectural analysis to help restructure older, heavily maintained code. As Turnlund says, "We use Klocwork for dethreading, removing unneeded compiles [which cuts compile times], and to better understand our code base for maintenance and renewal. Code that has been touched by many different developers, and that has been adapted over time to do many different things, starts to look like a Boston road map. We use Klocwork on the high-touch areas of the code and on areas where we've had issues. With Klocwork, you can accomplish with a few people in a few weeks what historically took many people many months — and you also do it more accurately."

At Cisco, proactive maintenance is key to ensuring software reliability. According to Turnlund, "We ask ourselves: How can we make our code bases live a lot longer? You never get the opportunity to do big rewrites. As you put more people on a project to meet deadlines, it's easy to get to the point where you can't see everything the code is doing. Unless you take a hard look at your code and perform maintenance on the fly to keep it working, this becomes a real issue."

**CHALLENGES AND OPPORTUNITIES**

Static analysis tools are not a panacea for all of the quality problems that confront today's software development teams. Development teams still need to perform unit, integration, and system testing to find functional and performance problems. But functional testing is unlikely to catch all of the buffer overflows, memory leaks, null pointer dereferences, uninitialized variables, race conditions, and tainted data problems that static analysis tools readily identify by directly analyzing source code. Development teams working without the benefit of static analysis tools are sending code to quality assurance that is full of defects and that may also have design flaws that make it unstable and potentially vulnerable from a security perspective. As we've noted above, finding and fixing defects late in the software development life cycle — during QA — is much more expensive. Worse, some of these defects and vulnerabilities will survive QA and make it into production systems.
Although the static analysis vendors are start-ups, some (such as Klocwork) have already demonstrated significant traction within Fortune 100 companies, and the technology itself is mature. IDC believes that the main barrier to adoption lies in the ability of software development teams to impose the discipline required to make static analysis a routine part of their work. Companies in software-intensive industries — including network equipment manufacturers, software vendors, and telecom and other embedded systems manufacturers — already appreciate the benefits of bringing discipline to the software development process. As process improvement initiatives gain in popularity, static analysis vendors will see growing adoption in the financial services and retail industries as well as other industries where software reliability and security remain key concerns.

**CONCLUSION**

Static analysis should be part of every developer’s toolkit. Checking code for design flaws, defects, and vulnerabilities is a task that benefits greatly from automation. For its part, automation saves time and money, improves the overall quality and comprehensiveness of the code inspection process, enables the continuous monitoring of code quality, and allows software developers to focus on more important tasks.

Incorporating static analysis into the development process reveals critical information about the quality of the software over time. It gives management significantly better visibility into the health and status of projects under way, helping reduce the risk of cost and schedule overruns as well as the risk of failure as systems enter production. Incorporating static analysis into the daily work of the software development team helps improve the software development process itself by providing instant feedback to team members and helping to guide them in the process of improving the quality of the software they create.

**Copyright Notice**

External Publication of IDC Information and Data — Any IDC information that is to be used in advertising, press releases, or promotional materials requires prior written approval from the appropriate IDC Vice President or Country Manager. A draft of the proposed document should accompany any such request. IDC reserves the right to deny approval of external usage for any reason.

Copyright 2005 IDC. Reproduction without written permission is completely forbidden.