Improving Software By Reducing Coding Defects

Investing in software defect detection and prevention solutions to improve software reliability, reduce development costs, and optimize revenue opportunities

Software coding defects increase the cost of development and support, tarnish a company's reputation, and limit revenue opportunities. By investing in a defect detection solution, a company can catch defects early, limit their financial impact, and institute a proactive approach of defect prevention.

This report provided by:

Klocwork

Klocwork Inc.
www.klocwork.com

New Rowley group

Bridging the gap between technology and business

Version 1.0 April 2004

Software defects raise costs and lower revenues

Software defects, also called errors or bugs, cost companies money. Defects raise the cost of developing software, and they can curtail revenue by limiting corporate flexibility and sales.

With software in every major electronic device – from PCs to cell phones to medical equipment to antilock brake systems – software instability or failure can drain worker productivity or, in extreme cases, endanger lives. It is critical for companies that develop software – whether for their own internal use or to sell to other firms – to improve the quality of their code.

Aim of this report: Reducing defects with automated tools

This report postulates that companies developing software can help reduce defects in their code by investing in automated software defect detection and prevention solutions. These tools are designed to sift through millions of lines of code and uncover errors as early as possible to save money on development and increase revenues dependent on reliable software solutions.

To understand the value of automated defect detection and prevention tools, we will look at the cost of software errors, the reasons why defects and bugs are so prevalent, and what these tools offer.

Report scope: Addressing coding-defect detection and prevention

Before we begin, we should note that there are two main types of software errors: *requirements defects* and *coding defects* (see Figure 1). Efforts to reduce requirements defects, including improving the way the business communicates the requirements of a software solution to IT, are being addressed by implementing methodologies like agile programming and leveraging concepts like the Unified Modeling Language (UML).

The solutions in this report are aimed at addressing the latter type of defect: coding bugs and errors.

Figure 1 Two types of software defects

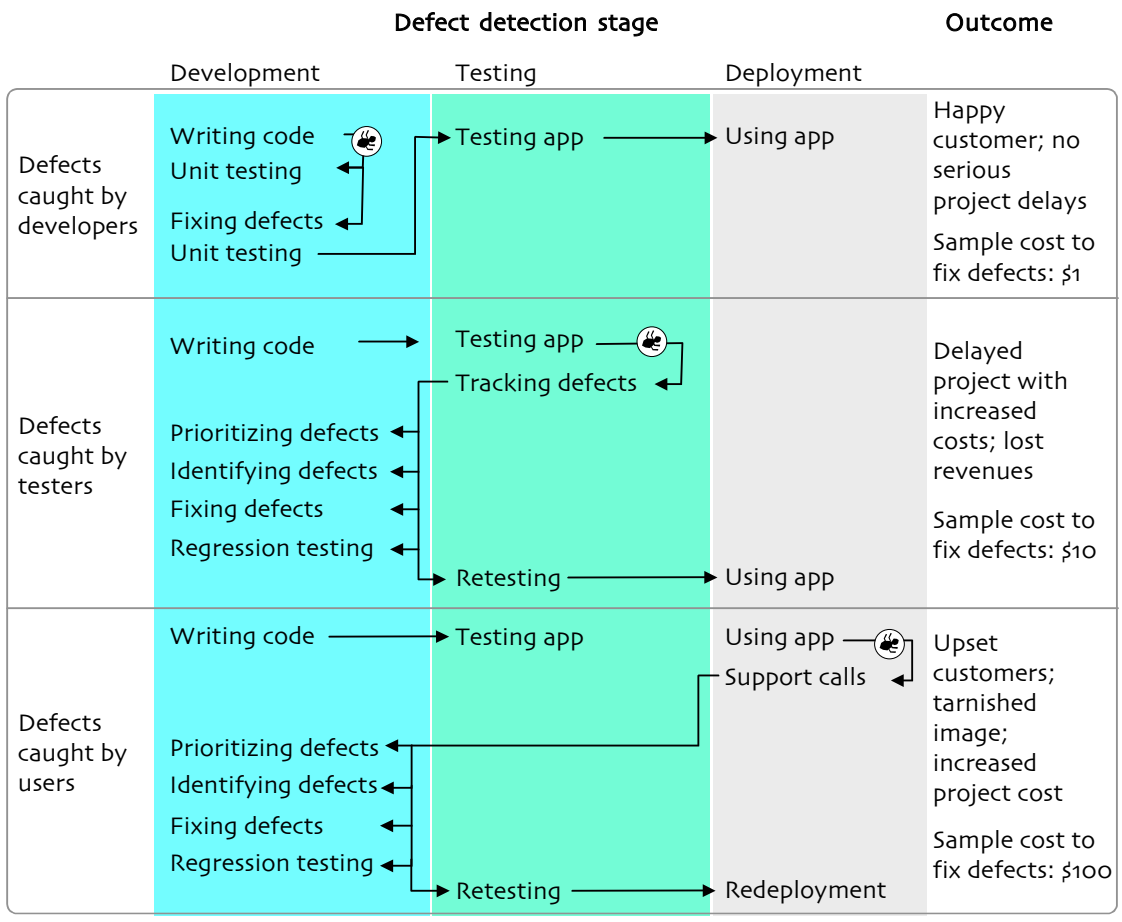
Type of defect	Causes	Effect
Coding	Developer mistake; reliance on other flawed code; development tool bug	Erratic behavior or failure of software; project delay; failure of deployed app
Requirements	Design failure i.e., wrong, changed, or misinterpreted requirements	Doesn't meet customers' or users' needs

The impact of defects

Software defects cost money for companies that develop code. A well-publicized study ("The Economic Impacts of Inadequate Infrastructure for Software Testing," May 2002: available at www.nist.gov/director/prog-ofc/report02-3.pdf) commissioned by the National Institute of Standards and Technology (NIST), estimates that software defects cost the US economy $60 billion annually. The study also claims that $22 billion of this yearly cost could be eliminated by improved testing procedures early in the software development process.

Why does catching a defect early matter? Because the later the bug is caught, the more it impacts the business from both a cost and revenue perspective (see Figure 2).

Figure 2 The effect of detecting defects late



Sample cost to fix defect amounts are based on ratios developed by Barry Boehm of the University of Southern California. Visit the NSF Center for Empirically Based Software Engineering at www.cebase.org for more details.

The fallout from defects on companies writing code

For companies that develop their own software for internal use or for use by partners and/or customers, the effects of defects include both hard (i.e., measurable) and soft impacts.

- **Increased IT development costs.** Software defects slow the development process, whether it's for a new application or an enhancement to existing software. An extended development period means that one project will most likely go overbudget while subsequent projects are delayed while waiting for developer resources. Defects that make it to end users cost the company even more; they involve not only reengaging developers but also other areas of the company, such as the help desk and customer support.

- **Sapped worker productivity.** Buggy or erratically behaving applications result in major organizational loss of productivity. While it's often difficult to measure lost worker productivity, increased help desk and customer support logs can usually paint a clear picture of just how many users are affected and for how long.

- **Delayed new offerings.** Companies, regardless of the industry, need IT systems to either directly or indirectly enable new offerings. Defects that slow new products and services are a double financial fiasco; they increase the cost of a new offering, and they reduce potential revenue due to delays or end user concern about system reliability.

- **Impaired corporate flexibility.** Buggy code often means that IT is slow – or incapable – of making a firm's technology capability support its business initiatives. This deficiency can affect activities like a company's ability to tie into a new partner's system, merge a newly acquired firm's apps, or enhance a customer service system.

The impact of defects on companies selling code

For companies that develop software to sell – whether as a component to be integrated into a larger software solution; a full solution, such as a Siebel CRM app or Microsoft Office; or as part of an embedded solution – software defects lead to additional problems. Examples include:

- **Increased development costs.** As with software developed for internal use, buggy code written for resale will raise the cost of the development project. For software sellers, this additional spending will directly affect the profitability of their product.

- **Reduced or delayed revenue.** When a development delay occurs, customers hold off purchasing the product. Carefully crafted sales projections can be rendered useless immediately as a project slips past its deadline. Missing a ship date due to debugging code directly affects a vendor's salespeople, who are counting on selling the new offering to meet quotas.

- **Damaged reputation.** Few companies can afford to repeatedly miss software deadlines like Microsoft and still make a profit. For most companies, delays and defects lead customers to question their commitment to a vendor's offering.

The causes of software defects

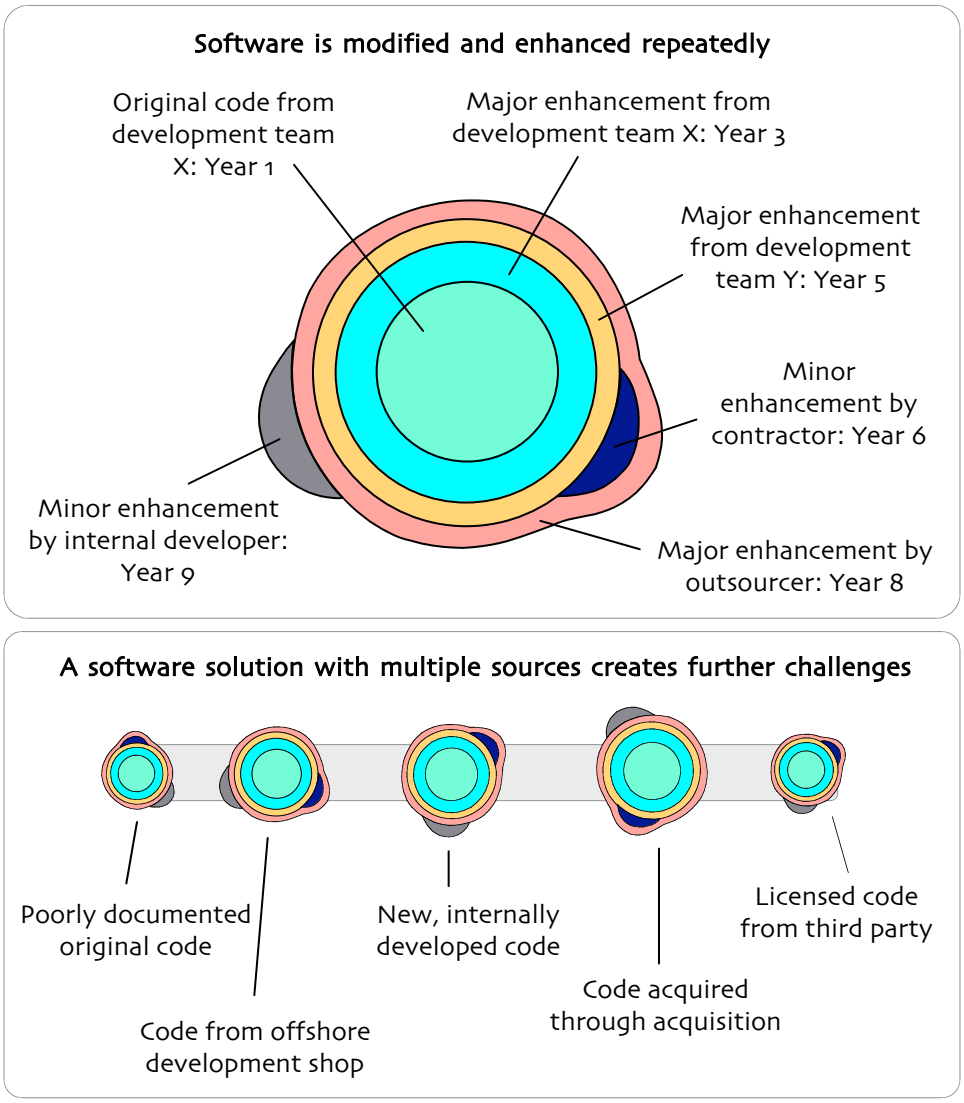
Nobody wants buggy software, especially the developers whose faulty code is the issue. Inevitably, the discussion of bugs leads people to compare software development – sometimes called software engineering – with other engineering efforts that appear to have far fewer defects. Why does code often contain defects and security vulnerabilities? Because of a project's:

- **Massive size.** In this case, size is number of lines of code. It may be possible for an experienced programmer to deliver a thousand lines of bug-free code, but it's almost impossible to do that when the project entails tens of thousands or millions of lines of code. The more lines of code, the more likely that the code will have defects.

  As an example, consider a 10-person software development team. In this scenario, each developer is responsible for 5,000 lines of code. If we assume that only 2% of each team member's lines of code have defects, then there would be more than 1,000 lines of buggy code for the project. Now, imagine this team being only responsible for one component of a multicomponent system. It's easy to see how software defects increase as the size of a development project increases.

- **Varied code pedigree.** It's not only the number of lines of code that causes defects, but often it's that a large software project involves a veritable code goulash (see Figure 3). If the project is using existing code, that code has probably been modified and enhanced many times over the years. In addition, portions of code may be from an acquired business or a third party, further confusing the matter.

Figure 3 Over time, software complexity increases



- **Diverse developer mix.** Most larger software projects involve numerous teams of coders tackling a particular part of the project. With increased reliance on contractors and offshore developers to meet developer staffing needs, the habits, tools, and processes of this eclectic group can affect overall code quality.

- **Shifting business requirements.** While this report is not about requirement defects, the changing requirements of a project does lead directly to code bugs as developers are continually given new criteria that they must implement. Every time the requirements are changed, the code must reflect these new demands. All of these changes to an

existing code base lead to potential errors within and linkages among the software components that make up the finished app.

Additional defect contributors

Additional factors lead to buggy code, including:

- **The independent, creative, and stylized aspect of coding.** While there are best practices in coding – for example, design patterns for particular challenges – coding is often a creative process akin to writing a novel, with many developers writing in their own style. This can lead to creative, individualized solutions, but it can also inject a variety of defects and potential security vulnerabilities.

- **Unmovable development deadlines.** As inevitable delays occur, stressed-out managers push stressed-out developers to finish projects. With budget cuts, most developer teams are being asked to do more with less, leading to buggy code.

- **Continuous code changes.** Business managers demand changes to software to do things like enable new functionality, lower maintenance costs, and improve performance. IT responds by enhancing a system's capabilities, porting code from one hardware platform or language to another, refactoring a code base to simplify the software, and service-enabling the system to make it available as a Web service (see Figure 4). These activities introduce bugs in the new code itself and surface defects caused by modified interactions with other software components.

- **Defect-prone programming languages.** Newer languages, such as Java and Microsoft's C#, were designed to minimize common coding errors. But for a vast majority of existing code, and particularly code for embedded devices, C and C++ remain the standard. Developers using these languages can easily introduce defects into their software.

- **A tolerance for defects.** While nobody wants bugs, everybody who uses PCs deals with their consequences nearly every day as applications crash and operating systems behave abnormally. A tolerance for acceptable bugs has been built up, and along with budget and time pressures, development teams learn to prioritize defects and determine which ones they can leave in the code.

Figure 4 Software development terms to know

Programming	
Languages	C and C++ ("c plus plus") are popular programming languages. Java and Microsoft's C# ("c sharp") are newer languages designed, in part, to reduce coding errors. COBOL is an older language used mostly on mainframes.
IDE	An integrated development environment, or IDE, is the software app used to write code. Microsoft, IBM, BEA Systems, Borland, and Eclipse.org make popular IDEs.
Bugs, defects, and errors	These are all names for the same thing: problems in code that cause errors, software failure, or unexpected behavior.
Activities	
Refactoring	Refactoring is the process of making existing software code better organized and simpler.
Porting	The process of moving code from one platform – hardware and operating system – to another.
Service-enabling	A process of making software capable of communicating with other software by using XML and other Web services technology.
Process improvements	
Agile programming	An approach to improve the software development process by being more flexible to changing requirements. An example is extreme programming (XP), a methodology that promotes doing activities like programming in pairs and "refactoring mercilessly."
UML	The Unified Modeling Language, or UML, is a vendor-neutral language for modeling new and existing software.
Testing/ analyzing	
Unit testing	During development, unit testing examines small portions of code to check for errors.
Regression testing	Testing that ensures that enhancements and bug fixes don't themselves break software.
Static design analysis	This process analyzes software code. Tools use a set of rules to identify problem code.
Runtime testing	Runtime analysis tools test the software for problems like memory and performance issues.
QA testing	Quality assurance (QA) teams test the application at runtime from an end user perspective. QA tests are often based on a series of functional requirements as well as use case scenarios. Resulting failures may be from requirement or coding defects.

Using automated tools to reduce defects

With a big enough budget – and time – even large-scale software projects with millions of lines of code can be mostly free of coding defects. Requirements defects are nearly impossible to avoid, given the length of software development projects and the fast pace of change in business processes and goals.

But the reality of the software development world is that few projects today even approach an error-free state because of many of the previously stated issues. Companies producing software, however, can significantly reduce software defects by investing in automated defect detection solutions, which analyze code, highlight defects, and offer prescriptions for improving the code.

Capabilities of automated defect detection tools

While uncovering software bugs is the primary goal of a tool for detecting software defects, these tools are much more powerful because they:

- **Test code, not the finished application.** Code is essentially a giant list of instructions until it is compiled or turned into a usable software application. Defect detection tools perform what is called static analysis. In other words, they are fed thousands or millions of lines of code, they analyze the code for its structure and defects, and they output a report that highlights and prioritizes problems.

- **Flag, categorize, and prioritize defects using included and custom criteria.** The tool determines what is a defect by applying rules (see Figure 5). Users can customize these rules as well as the categories and priority associated with each found defect.

- **Identify security vulnerabilities and other code characteristics.** While the tool can identify defects, it can also be used to flag other problem areas, such as security issues, redundant or extraneous code, and code segments that can be optimized to improve performance. In addition, filters can be set up to flag instances of individualistic coding styles that project mangers want to eliminate.

- **Analyze the structure of the software.** Defect detection tools aren't just useful for finding problems in code. They can also be used to understand how software was designed and implemented. By using visual displays and creating detailed reports, project mangers and developers can see how the various software components fit together and understand their dependencies.

Figure 5 Common types of coding defects

Type	Explanation
Translation	Defects introduced when turning software design into code
Logic	Coding mistakes
Security	Violations of security best practices and design parameters
Rules	Improper use of component APIs or system software interconnection violations
Life cycle	Defects that affect the ability to maintain, enhance, or test the software

- **Can be combined in suites for greater effect.** Many defect solutions are based on a single tool or around a collection of tools that are not integrated. However, suites of defect tools offer numerous advantages over single tools. For example, well-integrated suites can aggregate defect data to uncover defects that crop up in a variety of development teams, or they can ensure that the latest rules and filters are automatically applied to all of the defect detection tools in use.

Determining when to invest in defect detection and prevention tools

The cost of software coding errors needs to be high enough to justify investing in the tools. Some key criteria that will help determine whether to license defect detection and prevention solutions include (see Figure 6):

- **The critical nature of the software.** With a word processor, errors aren't that big of a deal. But developers in industries like aerospace and financial services creating mission-critical navigation or trading systems don't have the luxury of a single coding error.

- **The desire to protect the company's reputation.** While critical apps demand more attention by their vary nature, developers of noncritical software may choose to invest in a defect detection solution simply to avoid tarnishing their company's reputation and limiting its product's appeal.

- **The scale of deployment.** An application in the hands of 15 customers can easily be patched, but defects or security vulnerabilities uncovered in software running on thousands of network routers or millions of cell phones will be nearly impossible to fix in the field in any reasonable time frame.

Figure 6 Sample questions to determine whether to invest in tools

	Yes
Is the software **critical**? Will you lose customers, your reputation, or millions of dollars if it fails even occasionally?	☐
Is the software **deployed widely**? If a serious defect is found, can you even contact all of the users?	☐
Is the software **dependent on third-party code**, whether components licensed from a vendor or outsourced code?	☐
Will the software be **exposed to attacks** from the Net? Is it the type of solution that will attract hackers to uncover vulnerabilities?	☐
Is your company struggling to **reduce the high cost of support** from defective software?	☐

If you answered "yes" to any of these questions, you should probably investigate automated defect solutions. If you answered "yes" to two or more questions, it's in your best interest to investigate these solutions immediately.

A good example of this issue is Microsoft and its Windows patches. Often, patches for flaws are available, but the customer base simply doesn't know to – or choose to – update the code. The result? Whether it deserves it or not, Microsoft gets blamed for the effects of defects.

- **The use of third-party code of unknown quality.** When a solution depends on third-party code, such as code licensed from a software vendor, acquired from a merger partner, or developed by an offshore development firm, companies often don't have any idea of the quality of the software.

- **The programming language.** For companies whose internal apps or software to be sold or embedded in devices is reliant on the C and C++ programming languages, the inherent nature of those languages which makes it easy to introduce software bugs, may tip the scales in favor of an automated defect detection solution.

How to use defect detection solutions

Companies have a variety of reasons for analyzing code. They may suddenly come into possession of code through an acquisition, they may want to ensure code quality before licensing third-party components, they may want to inspect code from an offshore developer, or they may just want to improve their own internally developed code. Generally, defect detection activities fall into two buckets:

1. **Analyzing code at a single point in time** to make a decision, such as prioritizing defects or accepting third-party code. In this case, a code base is analyzed and the development team then scrutinizes the tool output to determine the quality of the code and prioritize fixes.

2. **Embedding the solution into the development process** to automatically uncover defects throughout the development life cycle (see Figure 7). By uncovering defects early, costs are drastically reduced (as noted previously – perhaps by a multiple of 10 or 100).

Getting proactive: Detection first, then prevention

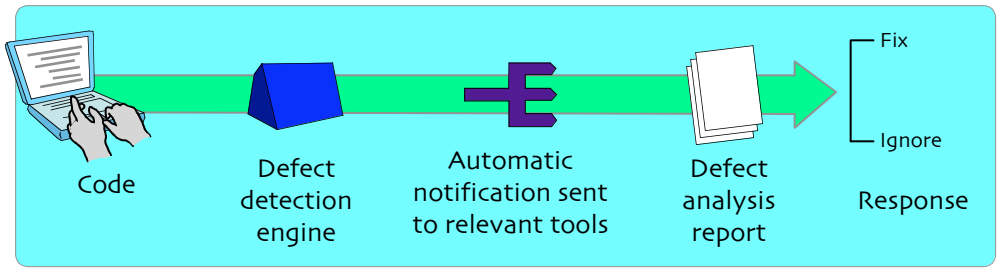
An aggressive approach to defect detection makes financial and common sense for many software developers. But a major driver in reducing overall software errors will involve moving from just defect detection – the process of finding and analyzing bugs – to defect prevention. Defect detection tools, particularly suites that can aggregate data and provide high-level views of companywide defect issues, facilitate defect prevention by identifying defect trends. This allows a company to:

- **Improve defect detection tool rules and filters.** The more that companies use defect detection tools, the more they can tweak the rules and filters to flag additional problem issues and flag practices that don't conform to corporate best practices. In the same vein, companies can also learn how better to prioritize coding-defect reports. With thousands of potential defects and little time to fix them all, these reports will save companies time and money.

- **Improve developer training.** By analyzing defect trend data, executives in charge of internal and external developers can pinpoint areas to address in training for individual developers, teams, and external services providers like offshore firms. The effectiveness of training can also be tracked by comparing defect trends before and after training sessions.

- **Alter the development process.** There are a variety of attempts to improve the software development process. For example, the extreme programming concept calls for a variety of process changes, such as pairing developers. How effective are these techniques? A defect detection suite could be used to compare the results of more traditional development methods with new techniques to help tweak a corporation's overall development approach.

Figure 7 Uses of software defect detection and prevention tools



	Scenario	Uses
Detection	**Code evaluation**	Determine quality of code: • Acquired through merger or acquisition • Delivered from offshore developers • Licensed from software developer
Detection	**Code improvement**	Uncover defects in code from: • Defect-ridden new development project • Old code base that is being enhanced • Projects to improve software performance and security
Prevention	**Continuous defect detection**	Reduce software defects by: • Using automated defect detection tools during new development and refactoring • Providing tools integrated into all developers IDEs • Analyzing common mistakes and training to avoid in the future

Final thoughts

There is no doubt that software defects raise development costs, tarnish a company's reputation, and reduce revenues. Better programming languages and development tools can only do so much. To minimize software defects, companies should follow a two-step process:

1. Investigate, and if it makes sense, implement a robust and cost-effective defect detection suite.

2. Move from defect detection to detection and prevention.

With defect detection and prevention solutions and processes in place, companies can both reduce the cost and time of code development and deliver more reliable and secure software.