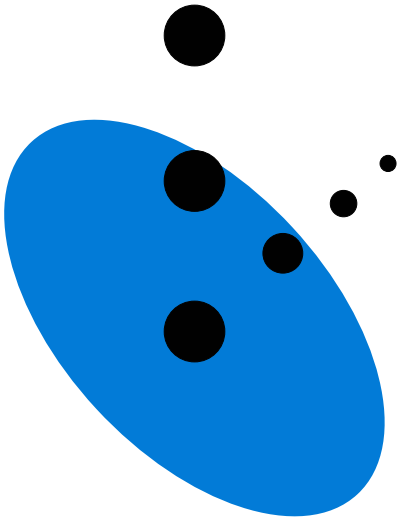
Finding Source Code Quality and Security Issues Faster

An Overview of Reflective's PRISM Technology



Reflective

June 1, 2004

Finding Source Code Quality and Security Issues Faster

I. Reflective's Vision – Safer Software

Reflective, LLC provides automated source code test and measurement solutions that enable enterprises worldwide to immediately reduce their risk from software issues and the costs associated with malicious breach of their applications. Reflective shifts the technology focus of software risk management from reactive to preventative measures and offers compelling economic justification for resolving issues earlier in the application development cycle. Reducing the high cost of software quality testing, Reflective helps customers identify and remediate problems in their code via an automated source code test and measurement platform – an industry first. The Company's vision is for the creation of safer software applications through the continuous automated assessment of source code. Reflective will achieve this vision by providing software quality and security testing, measurement, and metrics for enterprises seeking better code quality assurance.

Reflective's core product is the PRISM 1000 Source Code Analyzer™. Utilizing several innovative and proprietary technologies, PRISM is a code quality test and measurement platform that generates accurate and actionable issue data customers can use to reduce software risk and improve code quality. Through Prism's consistent, reproducible and fast source code analyses, customers gain better visibility into what's going on with their code at each stage of application development.

This paper describes the technology components of the Prism architecture.

II. The Challenge of Creating Safer Software

Conventional software development processes do not incorporate steps that specifically test code for source code quality or security weaknesses. The National Institute of Standards and Technology estimates that nearly 90% of all software developed each year for use in the United States goes unscreened for potential flaws.[1] The question is "why?"

[1] Source: National Institute for Standards and Testing - Report on Software testing. 2003

All customers and developers want safer software. However, three significant barriers stand in the way:

1. **Code Can Contain Many Types of Quality Problems**
   Testing software is difficult. Testing software for quality and security issues is an even tougher challenge. This is because the range of potential issues in code is broad, complex and constantly changing. Detection of issues (either static or dynamic) involves multiple processes. Defining what is a true problem, keeping current with the discovery of new issues, understanding what the trigger instance of the issue looks like and how it can be invoked, to simply capturing, codifying and controlling institutional memory concerning code quality standards, are daunting tasks for any development organization. The inability to easily check for a broad horizon of code quality issues leads to a lack of baselines and metrics; basic needs for any quality improvement process.

2. **Cost**
   Testing is expensive. Every software development organization today is tracked as a cost center with the constant goal of having more code created more cost effectively. Debugging and functional testing already chew up the majority of every software project budget. In addition, there is more code to examine. Over 450 billion lines of software are generated annually for U.S. organizations alone.[2] Given the additional expense of checking all that code for quality issues and the cost-conscious nature of today's development organizations, most companies conduct only basic quality checks on code if at all. "If it functions, release it," is still a prevailing mentality in many software development organizations.

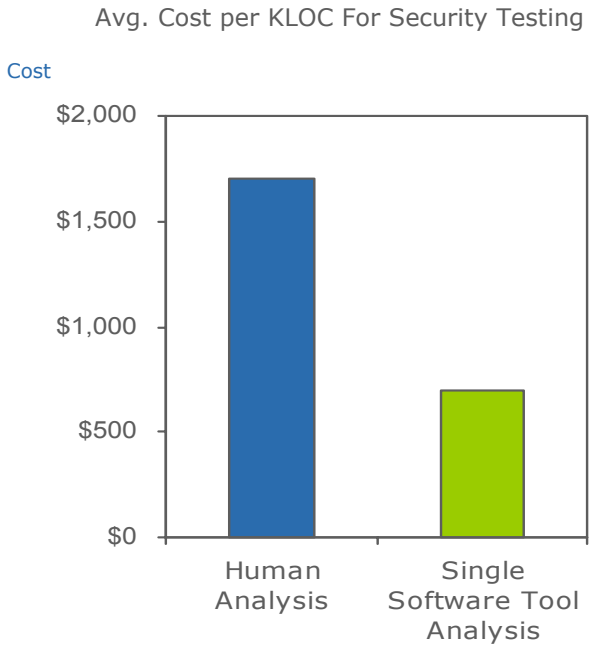


Figure 1 - Cost of Security Testing

[2] Source: National Institute for Standards and Testing - Report on Software testing. 2003

3. **Speed and Time**
   Checking code for quality and security issues today is largely perceived as a "nice to have" luxury for many organizations. It requires highly trained engineers as well as expensive software testing environments. A testing team experienced in the language the application is written in can generally only test sample code units versus the entire application itself. Few people have the skill set to analyze code for security flaws and those that do find the work tedious, boring and a poor use of their time. Examining source code by hand is also expensive and prone to error. Unintentional errors aside, testing engineers are simply overwhelmed by the volume of code that needs to be checked.

To generate higher quality code, the enterprise must strike a balance between risk, speed, quality and cost in order to determine whether and what code to test and how thoroughly to test it. Since the volume of code to check is large, the cost to check that code for quality and security high and the window of opportunity for software assessment short, companies simply cannot test all their software for as well as they should. The result is that businesses are forced to deploy applications that are often of poor quality and vulnerable to exploitation.

III. Applying Technology to Quality Testing

One of the maxims of manufacturing is that if flaws can be found closer to their source of origin, the easier and cheaper the fix and the better the overall quality of the end product. The same is true for software development. Since the emphasis in software QA is on function, most testing occurs during the dynamic stages of development (integration and beta). However testing for quality during functional test is too late in the cycle since most flaws are introduced in the coding/unit test stage of application development. (See Table 1 below.

Table 1 - Distribution of Software Flaws Based on Insertion Point – Source: NIST

	Stage Found					
Stage Introduced	Requirements	Coding/Unit Testing	Integration	Beta Testing	Post-product Release	Row Percentage
Requirements	5.0%	8.0%	2.3%	.2%	.2%	15.6%
Coding/Unit Test	NA	32.0%	40.5%	4.5%	4.5%	**81.5%**
Integration	NA	NA	2.3%	0.4%	0.4%	3.0%
Column Percentage	5.0%	**40.0%**	**45%**	5.0&	5.0%	100%

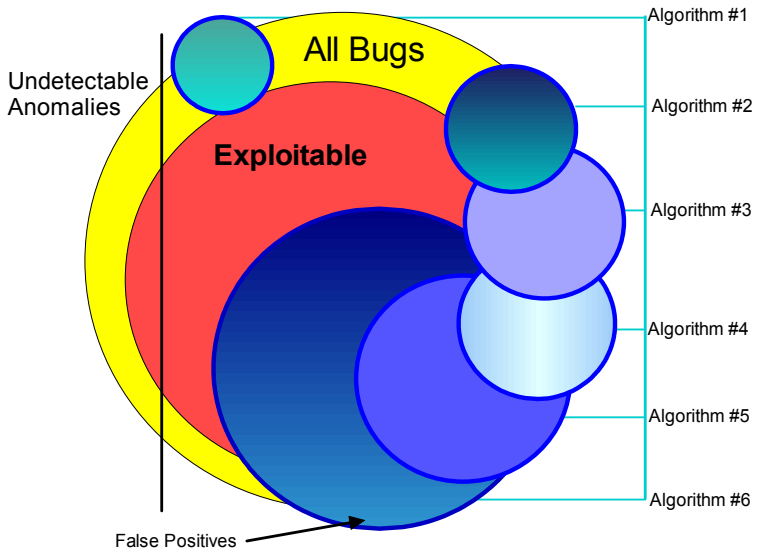
Consequently, if automated issue detection technology could be deployed at the static code unit construction stage of the software build, better quality code would be both cheaper and easier to construct. However, to achieve this objective requires the use of algorithms and algorithms, as they relate to software quality and security issues, are a more complex technical undertaking.

IV. Algorithm Based Testing – Strengths and Weaknesses

The use of algorithm based technology to test source code for quality and security issues is in its infancy today. Currently, there are over 166 individual tools available either commercially or as open source for testing code for quality or security. However, ***there is no single software tool available that allows a testing engineer to easily find a broad range of security and quality issues with few false positives at acceptable cycle times and cost***. To be useful in software quality analysis, an algorithm based tool must be faster than human analysis and lie within acceptable quality and cost parameters for the business.

How can algorithm based technology achieve these goals? The answer to this question requires a basic understanding of algorithmic analysis. The most critical technical challenge in algorithm based issue detection is the field and scope of the issues that must be checked. To test code for the broadest scope of issues requires that a series of algorithms be deployed. However, the mechanics of using multiple algorithms to test code are not easy. Figure 1 shows an example of how the coverage of potential issues varies by algorithm.

Figure 2 - Algorithmic Coverage of Potential Issues



Color	Classification

Yellow	Range of Known Issues
Red	Problem Issues
Blue	Algorithm's Scope of Detection Capability Note: Area of the algorithm outside the field of Known Issues = False Positive Detection

This graphic demonstrates the complexities of using software tools for source code issue detection. While some bugs are problems, others are not. Not all bugs can be found algorithmically, no matter how many tools are utilized. While each tool finds a specific set of issues, each tool also reports a range of false positives – issues that are not bugs. If the rules for each tool cannot be adjusted or the issue report cannot be filtered correctly, then these false positives will continue to be shown to the user.

Aside from the cost, time, training and complexity of deploying several tools for code quality testing across the enterprise, the user must develop a sophisticated rules and tracking database in order to stay on top of the ever-changing issue picture not only for the languages used in development but for the constantly changing software acceptance standards within the enterprise. Separate rules and instance data must be built and maintained for each language used in development. Assimilating, aggregating and reducing the output from each tool into actionable information that developers can use to fix problems is also slow and tedious. Engineers using more than one testing tool must generate and review separate reports from each tool. Once completed, duplicate "hits" must be eliminated, and found problems categorized and prioritized into before the code can be evaluated for critical flaws.

Therefore, to utilize software tools productively for source code analysis requires technology capable of:

1. Multiple algorithm assessment
2. Rationalization of the issues detected by each algorithm
3. A false positive reduction capability
4. A "tunable" rule set for each programming language used in development or each quality acceptance protocol set by the enterprise

V. Reflective – Delivering Next Generation Software Quality Analysis

Reflective has developed a set of technology components that together, address the critical issues associated with algorithm based code quality testing. Reflective's Prism Source Code Analyzer™ uses several different algorithms for issue detection, converts the output of those algorithms into a common uniform language, references that data against a proprietary rules and instances knowledge base and then delivers objective test and measurement data

through a platform that integrates seamlessly into any code development process. The output is a set of clear and consistent reports that development management, QA or the security team in the enterprise can use to improve code quality assurance. By technically automating and optimizing each of the steps associated with algorithm based testing into one integrated platform; Prism can rapidly and efficiently reduce the risk in code.

Prism™ is built on a unique architecture that uses multiple testing algorithms, intelligent analyzer "lenses," an issue classification index and a rules and instance knowledge base. Each of these technologies is a Reflective invention.

Multiple Testing Algorithms and the Prism Lenses

The Prism analysis process begins with the integration of multiple algorithms onto one test and measurement platform. Intelligent analyzer "Lens" technology, a Reflective invention, enables Prism™ to mix and match different algorithms in the same testing environment. A lens is a universal translator module that can be configured to work with output from any type of testing algorithm. The Lens design allows Prism to use any combination of GPL, partner, customer-specific and Reflective tools to test code for and then process the results through a rationalization, metrics and reporting engine. Multiple Lenses are linked together on the same system allowing Prism to scan a broad range of code quality issues or conduct a specific set of customer-driven tests or meet enterprise code acceptance requirements.

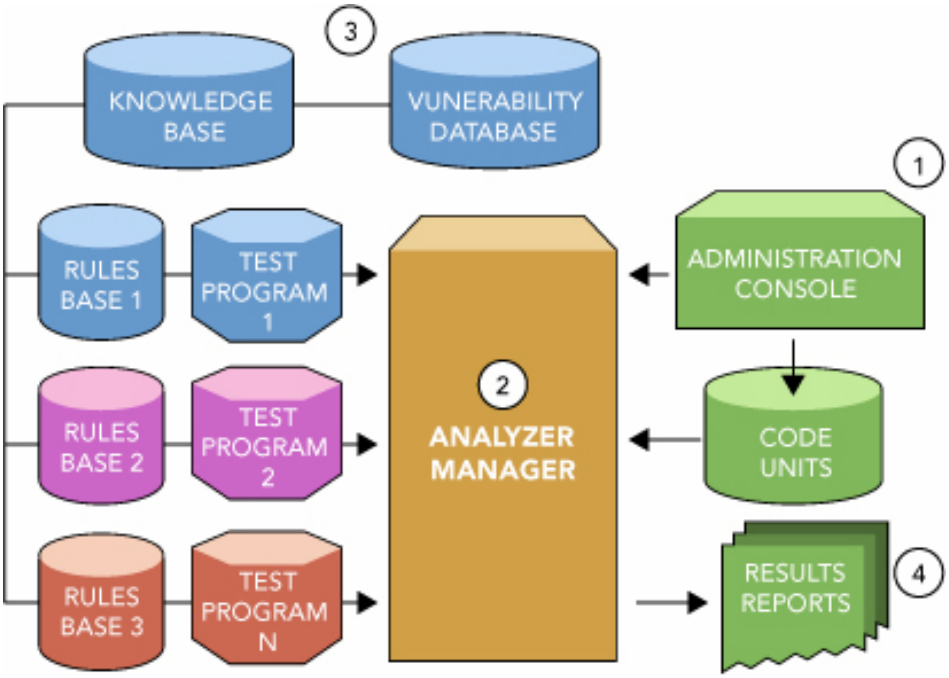


Figure 3 - How Prism Processes Code

Some algorithms are very broad but not very smart. They may find a set of issues which no other algorithm finds. The price of this capability is usually a large number of false positives – vulnerabilities identified that are not bugs. PRISM is able to reduce false positives and produce a single report with both breadth and accuracy. For example, a fixed buffer overflow problem found by Tool 1 is tagged and noted by PRISM. If Tool 2 finds the same buffer overflow problem, the instance is noted but the vulnerability is only reported once in the summary report. This helps keep the code quality team focused only on what are truly priority software quality issues versus wasting time chasing multiple instances of the same problem found by several different tools. The rationalization engine contains knowledge of each tool, the algorithms it uses, test cases to validate the algorithm and mappings from each tool to a single knowledgebase the results from which are then presented to the user.

Measurable Reporting Through Issue Tracking

The most valuable assets of the PRISM platform are the metrics that result from the analysis. Management wants answers to questions concerning software quality and safety. This has historically been a problem since there has never been a consistently comparable set of data regarding quality issues from build to build with which management could make informed business decisions. Reflective answers this metrics challenge with the ability to gather a wide array of objective, consistent and comparable data. Without clear, consistent and comparable metrics, management cannot judge the effectiveness of its code quality assurance initiatives.

Moreover, this information needs to be conveyed in a format that is designed for decision-making. Reflective's reports include: summaries; most problematic files; most seen issues; and exhaustive detail on all problems discovered by PRISM. The reports are also linked to both a tracking system and the developer's work environment so that clarity and consistency regarding code quality assurance can be pushed throughout the enterprise. Developers and software engineers get the assistance and information they need through the issue tracking system while reports provide managers access to the critical insight necessary to assess improvement in their software.

By automating and optimizing issue assessment, Reflective's PRISM technology can rapidly reduce the risk in code development enterprise-wide and provide the consistent metrics that enable the business to improve the quality of its application portfolio.

Technical Details

PRISM is delivered either as an appliance or as a web service. The platform is algorithm agnostic and runs in both a Win 32 and Linux environment. This flexibility in the architecture and the ability to run multiple Lenses on the same platform allows the system to be both

scalable and flexible. PRISM is designed to review code written in C, C++, Perl, PHP, Python and Java. The average processing speed is between 2,000 and 4,000 lines a second on a typical code scan. Output from PRISM is a set of comprehensive reports on the scan and the issues discovered. All reports can be filtered and sorted online and are presented in secure format by any web browser.

VI. Conclusions

To effectively use technology to help review source code for quality and security issues demands an algorithm-based capability that can detect issues quickly, examine large amounts of code cost-effectively, assess a wide problem horizon as well as be easy to deploy across heterogeneous development environments. Existing source code analysis software cannot deliver the combination of functions necessary for consistent and effective issue detection across the enterprise. Cost-effective source code analysis demands an automated testing function that is flexible enough to be deployed across different development environments and can deliver the metrics necessary for the business to both measure and improve code quality assurance without slowing down development cycle times.

Reflective's PRISM technology is designed to eliminate problems earlier in the software development cycle, generate clear and consistent metrics that drive code quality assurance programs and dramatically reduce the cost of checking code for security anomalies. Through its innovative use of new technology, Reflective reduces the workload on the testing team. By automating a majority of the source code checking function, Reflective's PRISM 1000 enables more code to be checked more thoroughly. The result is what customers are ultimately after – cleaner, safer code.