

Dispelling the myths about Static Analysis

Keeping the “Liar, Liar” from making the business decision

By Greg Spehar (spehar@full-knowledge.com)

Executive Summary

Dispelling the myths about static analysis is important topic since the use of static analysis tools promise to provide the last major hurdles in coding style management and identification and prevention of logical bugs or code problems in software development. The most common problems that can occur with development of software include the following:

- 1) Business Opportunity is poorly defined or not understood
- 2) Requirements are non-existent
- 3) Tools and processes are not performing as expected
- 4) Code quality and unpredictable bugs make the product unusable and un-maintainable

“Most software projects can be considered at least partial failures because few projects meet all their cost, schedule, quality, or requirements objectives. Failures are rarely caused by mysterious causes, but these causes are usually discovered post-mortem, or only after it is too late to change direction.”

Lorin J. May

Crosstalk Associate Editor

The Software Technology Support Center

<http://www.stsc.hill.af.mil/crosstalk/1998/07/causes.asp>

“Measures of quality are powerful indicators since the bulk of schedule delays and cost overruns tends to occur during testing and is caused by excessive defect volumes, it can be hypothesized that lack of effective quality control on large systems is a major contributor to both cost and schedule overruns.”

Capers Jones

Software Project Management Practices: Failure Versus Success

The Software Technology Support Center

<http://www.stsc.hill.af.mil/crosstalk/2004/10/0410Jones.pdf>

For this paper we will focus on the 4th issue of code quality and unpredictable bugs, also referred to as software reliability. This issue of code quality has not been addressed in a rational manner in the software industry to date. Where as the other areas have been addressed to some level with many software products now able to assist in business opportunity management, requirements identification and code management tools. With these tools and well-defined processes the creation and management of code can be achieved within a predictable timetables. Bet even then there are still reliability issues that can impact the use of the tool in the field.

Target Companies:

- IT Organizations
- Software Companies
- Embedded Systems

Target Audience:

- Executives
- Directors
- Software Managers

Our Goal:

Creating wealth by assisting in delivering your software projects to market fast and reliably.

Our Guarantee:

We guarantee that we will create wealth with a ROI that will exceed 1000%, or the services will be one third (1/3) the price.**


** Only applies to the wealth creation package.

Contact:

Full Knowledge LLC
2477 Santa Rita Rd #32
Pleasanton, CA, 94566

Office: 925-484-8490
Cell: 503-332-3663

Email:
sales@full-knowledge.com



“Traditional engineers now use software tools to design and analyze systems, such as bridges and buildings. These new kinds of design and analysis resemble programming in many respects, because the work exists as electronic documents and goes through analysis, design, implementation, and testing phases, just like software.”

Software Engineering
Fact Index

http://www.fact-index.com/s/so/software_engineering.html

But all this management, process and tools has not reduced or improved the overall quality of the software built today. The problem of software quality continues to baffle the people working to fix the problem and the management whom have to manage the business opportunity that the software solves.

“Based on this research, The Standish Group estimates that in 1995 American companies and government agencies will spend \$81 billion for cancelled software projects. These same organizations will pay an additional \$59 billion for software projects that will be completed, but will exceed their original time estimates. Risk is always a factor when pushing the technology envelope, but many of these projects were as mundane as a drivers license database, a new accounting package, or an order entry system.”

The CHAOS Report (1994)

http://www.standishgroup.com/sample_research/chaos_1994_1.php

2004 Results are available here:

http://www.standishgroup.com/quarterly_reports/index.php

The following are some questions that can be answered to help fix this problem:

- How will static tools increase application quality?
- What tools are needed to reduce false positives?
- What processes need to be changed to ensure false positives are managed?
- How will the organization benefit financially?

Liar, Liar – The Break Even Analysis

The current process for quality management can range from several techniques such as manual processes to more dynamic testing. On the manual side there are code reviews that can drive out bugs on the development side and there are manual testing processes that require teams of people to use the software until it breaks, then you have the least quality focused approach by allowing the customers to drive out the bugs and report them back to the software organization team this is usually called Beta testing.


On the developer side there are automated dynamic testing harnesses that are run by the development teams such as Junit for Java and general unit testing for C/C++. This requires the developer to maintain an ongoing battery of tests that have to be continually updated.

Then there are the automated scripting tools used by the testing groups. Where the testing groups execute the application to perform conformance to requirements, performance testing and defect testing. These dynamic testing tools require constant updating and adjustment as the product changes. These tools therefore also generate false positive data if the scripts are not kept up to date or verified as to what is being tested.

Finally there are the automated static analysis tools, also called automated software inspection (ASI) tools that will find defects prior to the testing cycle. There are proponents that are for and against this technique and as one unpublished paper stated these tools do have a cost.

“An important issue with the use of automated software inspection (ASI) tools is the inevitability of significant amounts of false positives. False positives are when the tool identifies a fault but a deeper analysis of the context shows no problem. There can be as many as 50 false positives for each true fault found by automated inspection tools. Often, ASI tools can be customized and filters can be established so that certain classes of defects are not reported. Additionally, ASI pre-screening subcontract services examine the identified defects and the product will remove as many false positives as possible.”

Preliminary Results On Using Static Analysis Tools For Software Inspection
N. Nagappan, L. Williams, J. Hudepohl, W. Snipes, M. Vouk¹
¹ Department of Computer Science, NC State Univ, Raleigh, NC, USA
² Nortel Networks, Software Dependability Design (SWDD), Research
Triangle Park, NC, USA



If the overall average for all static analyzers is about 1 positive defect for 50 false positives (the reporting of an issue that after investigation is considered to not be a defect is referred to as a “false positives”), there can be a general “break even” analysis that can be defined for the use of static analysis tools. To fully understand the break-even analysis the following assumptions need to be made:

- 1) Cost of 1 hour of developer time = \$100
- 2) Cost of 1 hour of Code Review = \$500
- 3) Cost of finding 1 defect in test = \$500
- 4) Cost of finding 1 defect in field = \$15,000
- 5) Average amount of time to identify reported messages = 5-10 minutes (no fix applied)

So it is clear at this level for every defect found there is 50 false positives or 51 x 5-10 Minutes or 255-510 minutes (4.25 – 8.5 hours) of effort for every defect found. This translates to about \$425-\$850 per defect found for using the average static analysis tool. This shows that the faster the evaluation and the less false positives the more cost effective these tools will become. So if we hold evaluation to 5 minutes per message what are the breakeven false positive rates for testing and field defects?

- 1) $\$500 \text{ per defect (Test)} / \$100 \text{ per hour} = 5 \text{ hours} / 5 \text{ min. per message} = 60$
- 2) $\$15,000 \text{ per defect (Field)} / \$100 \text{ per hour} = 150 \text{ hours} / 5 \text{ min. per message} = 1800$
- 3) $\text{Testing Defects per Field Defects} = 1800 / 60 = 30 \text{ Testing/Field}$

Even at these high false positive rates the numbers are reasonable. So some groups and product vendors might say, “False positive rates are too costly!” - “Liar Liar” may be the right response.

Pants on Fire – Driving Triage Costs Under \$1

So now we know what the “Break even” requirements are for these static analysis tools. Lets take a look at the methods required to bring those numbers down from 1 in 50 (2% are defects out of 100 messages found) to 1 in 2 (50% of 100 messages are defects) or better yet to 9 out of 10 messages are defects (90% out of 100 messages are defects) and finally we will look at the advantages of reducing the citing time from 5 minutes to under 1 minute per message.


There are several methods that help to reduce the problem of false positives; they fall into the following categories:

- 1) Increase accuracy of the analyzer
 - i. Increase the accuracy of AST (Abstract Syntax Tree) analysis
 - ii. Increase the accuracy of other analysis mechanisms
 - iii. Utilize data flow analysis
 - iv. Utilize interprocedural analysis
- 2) Clarify defect categories
- 3) Utilize heuristics to select messages from defect patterns
- 4) Use of Knowledge Base to increase analyzer accuracy
- 5) Extensibility of checkers for application specific usage patterns

Through the use of these techniques and others not mentioned here, the reduction of false positives has hit levels above the 90% level allowing more than 90 out of 100 issues to be defects. Historically static analysis systems have had low hit rates, which discourages use. With these tools and techniques to accelerate accuracy and increase the speed of triaging issues, static analysis is becoming meaningful tool as we will see in the following analysis.

So if we perform the same analysis as before we will find the following to be true for those defects that can score better than 90% defect hit rates, so for every 10 messages there is 1 false positive. 10 x 5-10 Minutes or 50-100 minutes (.85 – 1.6 hours) of effort for every 9 defect found or (50-100 minutes) divided by 9 defects is equal to 5.5 – 11.1 minutes per defect. This translates to about \$83-\$166 per 9 defects found for using static analysis tools at a 90% hit rate. This is a reduction of 4500%+ in the time required to “mine” the reported messages. So if we hold evaluation/mining to 5 minutes per message what is the “value” derived for code reviews, testing and field defects?

- 1) \$500 per defect (Test) / \$100 per hour = 5 hours / 5 min. per message = 60
 - a. 90% of 60 is 54 defects found within this time frame
 - b. 52 Testing Defects* \$500 + 2 Field Defects * \$15000 = \$56K
- 2) \$15,000 per defect (Field) / \$100 per hour = 150 hours / 5 min. per message = 1800
 - c. 90% of 1800 is 1620 defects found within this time frame
 - d. 1560 Testing Defects* \$500 + 60 Field Defects * \$15000 = \$1.68M
- 3) Testing Defects per Field Defects = 1800 / 60 = 30 Testing/Field is still valid



Now how do things change when the amount of time to evaluate a false positive falls from 5 minutes to under 1 minute? To reduce the time for evaluation several things must happen, the ability to understand the priority of the issue and then to be able to view the issue and triage it quickly. This is done by the following way:

- 1) Each defect category has a priority
- 2) Defect citing tools that list and display issues in a manner similar to a bug tracking system


So with these capabilities, for every 10 messages reported of which 9 are actual defects the amount of time to triage or mine these defects would be about 10 minutes or about \$10. This translates to about a 5 fold increase in capability from 5 minutes to 1 minute. This verifies the faster the evaluation and the less false positives the more cost effective these tools are. So if we hold the evaluations to 1 minute per message, what are the breakeven false positive rates for testing and field defects?

- 1) $\$500 \text{ per defect (Test)} / \$100 \text{ per hour} = 5 \text{ hours} / 1 \text{ min. per message} = 300$
- 2) $\$15,000 \text{ per defect (Field)} / \$100 \text{ per hour} = 150 \text{ hours} / 1 \text{ min. per message} = 9000$
- 3) $\text{Testing Defects per Field Defects} = 9000 / 300 = 30$ Testing/Field still remains the same

We see that the number of static analysis defects before finding a field defect increases from 30 to 300. Finally, we need to verify that these numbers are achievable with current code base sizes. The accepted defect rate per thousand lines of code is about 6 per 1000 lines of code (LOC), which means that for every million lines of code there are $(1000 * 6 = 6000 \text{ defects})$ that can range from performance, to requirements to operational defects. (Applied Software Measurement: Assuring Productivity and Quality, Capers Jones, McGraw-Hill, June 1996 - Table 3.44 Defect Rates by System)

Since static code analyzers cannot perform performance related analysis today and the static analyzers cannot verify functionality today the 6 defects per KLOC needs to be reduced to about $\frac{1}{2}$ of the total. So we can make an assumption that about 3000 defects per million LOC have the possibility of being caught by static analysis. And if we find the 90% hit rate holds we can see that 3333 messages will be reported per million LOC on average with a cost of triage or mining of $3333 * 1 \text{ minute} = 55+ \text{ hours}$ or less than 1 week for two people performing the task. The identified defects then can be scheduled in the normal bug tracking system for the development team to correct.

Comparatively to existing testing systems that cost \$500+/- per defect found in testing, static analysis will evolve to \$1+/- per defect before entering test. More interesting is the potential of finding field defects that cost \$15,000+/- when found in the field and with static analysis tools they can be found for about \$1+/- . The cost staving are staggeringly enormous and beneficial for any



development team attempting to create or manage systems that can range from thousands of lines of code to multi-millions of lines of code.

Would it not be better to find these issues prior to entry into the code base? The static analysis tools are generally available for use on the desktops so these issues never reach the integrated code branches. With the cost savings that reach into the millions of dollars and the quality of the code base to increase even as more code is added to the system, integration of static analysis tools into the desktop is a must have option.

Finally, what would the labor costs be for a static analysis tool that was able to achieve 95%-99% accuracy? At this level of accuracy there would be no need for a triage mechanism and the defects should be able to be automatically imported directly into the bug tracking system, moving the variable labor costs to zero (0) for when using an extremely accurate static analysis tool.

For those that say “false positives are too costly” the truth may be these people have yet to understand the economics of static analysis tools or they are more interested in maintaining tools and systems that currently provide known process factors and the introduction of these tools may impact their carefully managed processes. Additionally, there are special interests with much time and effort invested in tools and processes internally or at a product level. It is clear that in 2004 and 2005 the techniques and technologies for static analysis will in fact come to pass, such that we indeed can say to the special interests and tool vendors that are resisting the introduction of this new technology:

“Liar, Liar, Pants on Fire.”

The Product Tools

There is a set of tools provided by the Static Analysis Vendors that can provide static analysis to the level of accuracy exceeding 90% and reporting tools that assist in reducing the mining and triage times from 1 to 10 minutes. This set of products require the following framework to perform their jobs properly:

- Static Code Analyzer and Reporting – Static Analysis Tools
- Visual Representation of Issues – Architecture Tools
- Desktop Static Code Analyzer – Desktop Analysis
- Tracking Defects through Time – Metrics Management

Each of these products are intended to supply the proper controls to allow the development team to identify issues at a high hit rate and triage/mine those messages quickly at the desktop or at a build. The following is a more detailed description of the tools.

Static Analysis Tools

Build time tool that runs at every build or at an appointed build to provide a snapshot of the system at moment in time. The use of this build snapshot allows for the integration and execution of rules that will catch the specified coding defects, code metrics and architectural violations that were agreed upon prior to the development effort. Static Analysis Tools includes the most advanced static analyzer with the use of heuristics, interprocedural analysis and extensibility. Future versions will take into account more accurate analyzers and dataflow analysis further reducing the false positive rates.

Architecture Tools

Architecture tool that allows architects to see the actual relationships between files and allows the architects to display the issues found by the Static Analysis Tools tool in a graphical user interface. The architects can further use the tool to define an architecture that is enforced by each developer and at every build.

Desktop Analysis

The Desktop Analysis tool allows a developer level enforcement to ensure the developer does not violate the defect controls for hard to find bugs and security defects. The tools can also track and warn the developer when they have passed metric thresholds that will put their code at risk down the road. And finally, the enforcement of Architecture must happen at the developer's desktop. With the use of Architecture Tool the architectural intent can be translated to the developer such that they will not break the architecture laws that have been established.

Metrics Management

Metrics Management is the tool that provides detailed metric data build over build. This tool will give the developers and managers the visibility and ability to gather information. Thus providing the information to the development managers to back up their development decisions when the time crunches occur. The executive team, having access to the same data, can also provide incentives that have clear metric objectives and goals allowing effective risk management.