

The Silver Bullet

Understanding how this change will impact the organization

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

Executive Summary

Finding and removing software defects has a direct impact on the cost of software development projects. The National Institute of Standards and Technology (NIST) reports that "...80% of development costs [are spent] on identifying and correcting defects." The NIST continues to conclude "...all errors cannot be removed, more than a third of these costs, or an estimated \$22.2 billion, could be eliminated by an improved testing infrastructure that allows earlier and more effective identification and removal of software defects." (NIST June 28,2002)

The document called "The Smoking Gun: Finding the ROI in Large Software Development Projects with Static Analysis Solutions" focused on those ROI frameworks and processes that help in finding bugs before the test process through the use of static analysis. Most developers have been using standard static analysis programs for defect detection for years. With All Tools Vendor's combined improvement in the static analysis engines and introduction of metric and architecture enforcement the potential return on investment (ROI) becomes ever more promising. In addition, with the use of Internet web technologies these results can be defined and distributed immediately, reducing the time required to manage the results generated by the Vendor tools.

This document describes the process of adopting this new technology and how a company can achieve significant returns without disturbing the ongoing development effort. The paper will be defined into 5 adoption phases that will outline the timelines expected and answer the general question of who will do what when and how. The following is the outline of those phases:

- Phase 1 – Critical Defect ROI Proof (1-3 Months)
- Phase 2 – Metric ROI Proof and Critical Defect Rollout (1-3 Months)
- Phase 3 – Architecture ROI Proof and Metric Rollout (1-3 Months)
- Phase 4 – Architecture Rollout (1-3 Months)
- Phase 5 – Protection Rollout (1/2 day to 1-2 Weeks)

Furthermore this change entails fundamental changes to the business paradigms that include moving from defect detection to prevention and manual prevention to automated prevention with impacts to the business changing the way software is built and maintained. The following sub-sections will define these issues in more detail.

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



Process Change - Detection vs. Prevention

In the quest to develop and maintain ever growing code bases, the need to have tools that can automatically identify issues or concerns will grow. If current bug rates of several bugs per thousand lines of code continues to be an industry standard, the amount of bugs per system will grow to the point that very large systems will require such extensive testing and feedback mechanisms that these programs will not be maintainable or will only be maintainable by very large companies with accessible resources.

This industry norm is currently focused on the “Defect Detection” strategy. With companies having extensive investments in the testing process or in the bug correction process. These companies make investments to the level where customers and testing resources have to expend tremendous effort by participating in the quality correcting mechanism of bug identification and resolution.

The new strategy that is becoming important is “Defect Prevention”. This strategy has been embraced by many companies in the form of more manual processes within the development process and more complete and specific business specifications with the associated technical specifications. Even then organizations can expend many more resources and effort to ensure the technical design is correct and error free.

Automated Prevention

The ability to run incremental defect detection prior to submission of code to the code tree or the submission of an entire code base to testing is the next technological advantage. This type of testing is defined within the industry as “Static Analysis”. With the current static analysis tools and technologies, the written code or code base can be compiled and rules engines can be defined to simulate an execution to find critical bugs that would otherwise be found in the downstream test process or in the field.

Early static analysis tools, although useful, all suffered from noise that has detracted from their value. With the new Static Analysis Tools, the noise has been reduced such that these tools are currently very cost effective. In addition, combining static analysis with metric and architecture data from the tools suite further increases the accuracy of critical errors.

These mechanisms can be automated and managed to ensure that the users of the tools will prevent critical logic errors (bugs) from entering the code tree or the testing process. This process of prevention will change the industry in three ways:

- Testing will be able to focus on performance and conformance to business specs.

- Development productivity will increase through a significant reduction in analyzing and fixing defects reported by testing and defects reported from the field.
- Customers will not have to endure the problem of failed applications in the field.

Business Impact

The impact of this process change will positively change organizational business objectives such as the need to control software defects, to identify security vulnerabilities, to identify and manage the software metrics and KPI (Key Performance Indicators), to ensure offshoring success and quality, to prevent “Fire Drills” at the time of deployment, to allow surgical testing and to allow code management mobility.

A detailed discussion on how each of these business areas positively impacts the organization can be found in the business whitepapers. Where the remainder of this document will focus on how to manage the introduction and planning of this technology into the corporate setting.

Planning - Corporate Portfolio Rollout

When implementing a defect prevention paradigm into an organization there will be five primary phases that will be required. The first phase will focus on the proof of the Critical Defect ROI Value. The second phase, based on the success of the first, is the proof of the Metric ROI and Rollout of the Critical Defect Management. The third phase is the proof of the Architecture ROI and Rollout of Metric Management. The fourth is the Rollout of the Architecture Management. The fifth and final stage is to rollout of the protection mechanism to the development teams. It should be noted that all five (5) phases could be rolled out in any order, this order was chosen as the most typical arrangement. This paper provides an example of a typical rollout to a software organization with 5 product lines. The initial integration starts the adoption process at the point of least impact and provides the most immediate value driven feedback.

The basic project size assumption for this discussion will be that of a medium project size that can range from 1MLOC-5MLOC per product for a total of 5MLOC–25MLOC per system. Where the total number of developers and support staff can range from 100-400 individuals. The same framework can also be applied to small systems and very large systems as follows:

- Smaller projects can also find the needed benefit. Those projects tend to be from 50KLOC to under 500KLOC and would follow the same steps but would require less integration and follow-on observation. Please see the Appendix A for a description of how to plan for installations that do not exceed 1-2 MLOC with this project framework.

- Very large projects can also be dealt with in the same manner but on a much larger scale. Please see the Appendix B for a description on how to plan for very large installations that exceed 25 MLOC using this project framework.

Project Objectives

The importance of these phases cannot be ignored. This effort is primarily intended to provide the initial “proof” for continuing with an installation of the Vendor's Toolset. The objectives for this phase are clear:

- a) Find enough value to establish a baseline and prove ROI for rollout
- b) Leave working integrated tools for gathering ongoing project data build over build

Tool Implementation

As with any effort a decision must be made to focus on the areas that are most important to a customer to show the value of the tool. The lowest hanging fruit will always be the demonstration of the defect detection technology within a Static Analysis tool. If this is not the starting point for the target organization's project, the organization is free to mix and match the other sections of analysis to achieve their stated goals as this paper is only intended to provide a working model in which the target organization should be able to determine a rollout project management plan.

Initial project identification and initial Rollout

As in this example we have 5 total projects in which to rollout the product. There may be many reasons to start all 5 or start each project itself, but it would be prudent to start with the projects that lend themselves to have the most to benefit from these tools. That decision should fit the following criteria:

Criteria #1: Start with a small but troublesome project. This provides the organization with the motivation to think about what is happening in the development groups and provides them with a starting point that should provide a high potential of success. Choosing a group that has no defect problems and has minimal code changes would not yield as much value.

Criteria #2: System must be written in C/C++ or Java.

Criteria #3: The tool should be analyzed on Linux, Solaris or Windows Operating Systems. Please see the technical specifications for the current releases supported.



Total Timeframe

As there are five (5) main phases that range from 1 month to 3 months in length and it is clear that complete integration of all tools can take from 6 months to 2 years depending on the rate of adoption and the decision to perform some phases in parallel. The important note should be that the total applied integration work time (tool integration and tool process training/adoption) can add up to be 2-5 weeks plus or minus a week or two (1 day to 1 week times 5 phases) per project with this time spread over the 6 month to a 2 year timeframe. Comparing some of these implementation costs to the potential return, the ROI can achieve the high returns expected with software tools.