

The P&L of Static Analysis

Executive Level P&L decision making using Static Analysis Tools

Target Audience: Executives, Directors or Managers of Software Development
By Greg Spehar (spehar@full-knowledge.com)

Executive Summary

Managing software is an ever increasing complex task that is complicated with risks and costs that can be directly attributed by the lack of visibility into the code base and the need to perform labor intensive black box type testing after each cycle of development.

Static analysis promises to bring that effort of finding defects into the development process itself with tools and techniques that will capture costly defects and software complexity issues at the developers desktop. This paper will outline the key points of using static analysis to better control the profit and loss (P&L) of a development effort.

With static analysis tools the increasing complexity and coupling within a code base can be managed through the use of the control points such as defect control, security control, metric control and architecture control. This control will provide the visibility into the software complexity through the use of the metrics controls and architecture controls and provide an insurance net for any defects that might escape due to misunderstanding of the software context when changing the code.

This process change with automated tools can bring defect testing up to the point of defect entry allowing the developer to create unit testing to verify their intended functionality, perform automated white box testing (Static Analysis) to ensure there are no logic errors and perform automated component testing managed by the metrics controls and architecture controls to ensure complex components that are highly coupled are tested thoroughly.

This second step “white box testing” or static analysis will be addressed specifically in this paper and will provide the following benefits:

- 1) Decrease field defects reducing the number of software updates (Lowers P&L Cost)
- 2) Decrease the labor for testing teams/processes (Lowers P&L Cost)
- 3) Assist in providing predictable full function scheduled releases (Increase P&L Profit)

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

Return on Investment (ROI)

With static analysis technology the goal of increased software quality is within reach today since the overall product costs and project process integration risks have been mitigated such that the return on investment can exceed 1000%. This assumes that the opportunity costs of \$50K per year per developer and the overall software costs and process costs are about \$5K-10K per developer for the first year and \$2-4K per developer per year after the first year, such that the break even period can be reached in less than 30-60 days. Thus giving a clear set of technology to manage the bottom line:

- 1) Controlling the P&L (More Profit potential and less Loss potential)
- 2) Controlling the Risks that Impact the P&L (Visibility through Metrics and Architecture)

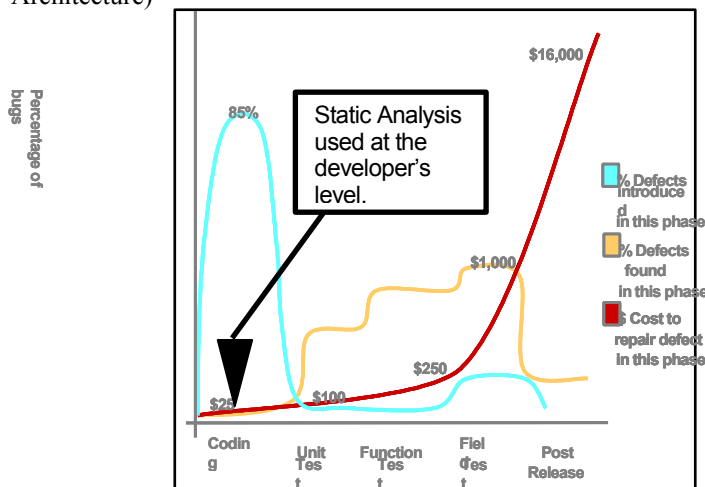


Figure #1: Typical Software Defect Cost Graph from Capers Jones

With the following figures showing the impact of using static analysis at the developers level to reduce complexity/coupling and defect volumes so that projects can be managed through time to completion and profitability.

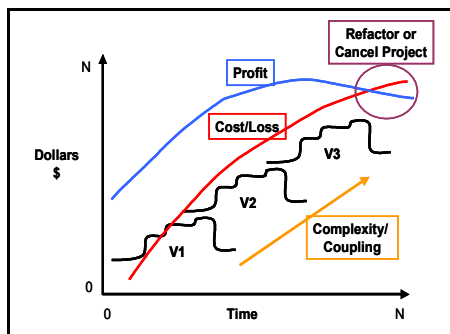


Figure #2: Multiple Versions **WITHOUT** Complexity/Coupling Control with Increasing Volume of Defects Due to Increasing Code Complexity and Coupling

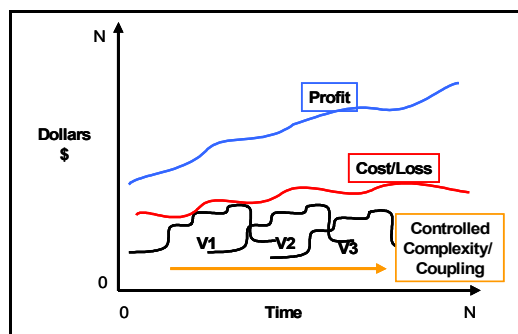


Figure #3: Multiple Versions **WITH** Complexity/Coupling Control with Maintaining Volume of Defects Due to Controlling Code Complexity and Coupling

Profit and Loss

The Need for Software Visibility

The need for increasing visibility into the software corporate environment is an ever growing concern. With the ability to predict the quality, cost and schedule of a development project becoming of critical importance. The quality of software is becoming an ever more important role in the perceived reliability of products on the market today. From computer software to embedded devices such as cell phones, customers and development organizations are either inconvenienced with software or devices that do not work or are burdened with the costs of fixing and re-issuing the repairs to the disabled devices.

The overall reliability and development costs of software is not improving over time as noted from the National Institute of Standards and Technology (NIST) "...80% of development costs [are spent] on identifying and correcting defects." The NIST continues to conclude that "...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) And finally, the actual delivery dates of software are always slipping and therefore making the introduction of newer software technology an ever increasing challenge.


Where can technology such as static analysis play a role in removing the risk of these issues in the long term? To answer this question one must first set some assumptions regarding the areas of concern at executive levels within the organizations charged with the job of creating and maintaining software. Two of the primary concerns that executives have for software management include the following:

- 1) Managing the P&L (Profit and Loss)
- 2) Managing the Risks that can Impact the P&L

The Profit and Loss for Software Development Teams

What specifically are the P&L points for software? There are many ways that this can be calculated, but for the most part it can be broken into three main areas of cost: people, hardware/assets and software that supports the people and hardware. With people being the most costly of the three, the real challenge then is to understand what the relationship is between the people and the software P&L.

In the P&L, the loss (cost) part of the equation is the cost structure of the people involved in building the software product at hand. The profit is the proportion of features supplied through the software that the market is willing to pay a going rate to utilize. Ever increasing functionality means ever increasing revenues and in some cases with heated competition, the increasing



of functionality means that the rates remain the same through time. Figure #1 from the executive section shows the costing incurred when defects are found in the testing phase. Figure #2 shows how this strategy of testing for defects can be problematic over time as the code increases in complexity. Finally, Figure #3 shows that if the complexity and the coupling of code is contained over time, this will lead to increasing profits and a more predictable testing strategy since the code is **NOT** becoming more difficult to update and therefore more difficult to test over time.

The Goal

It can therefore be said that the overall goal of the software executive is to provide the best people with the best hardware/assets and software to build the most functionality in the fastest time to meet the market need and create the most value/wealth over time with the most consistency without growing the complexity and coupling of the code base.

The Risks

This brings us to the second point, what can impact that process? What is at risk to get to a manageable and predictable P&L? This question again can have multiple answers but the primary risk impacts fall into the same three categories: people, hardware/assets and software. If the correct people with the correct skill sets are NOT hired or trained, the throughput of adding functionality will decrease. If the hardware or building assets prohibits creativity or the systems are too slow to perform the needed work, the throughput of the functionality is reduced. And finally, if the software being used by the developers is not working properly or is not the correct tools, then the throughput of the functionality is again reduced.

So with this perspective, an investment in tools and process will mitigate the impact on people and their ability or manage the throughput of functionality. And if we assume that all the right people have been hired with the correct skill sets, all the correct hardware/assets are in place and the correct software tools have been loaded and are working, the only final factor that remains is the actual software itself, the REAL product that is being produced by the people, assets and software. This is why we see the effect as diagrammed in Figure #2 where the profit based functionality and costs cross and a decision needs to be made to refactor the code or cancel the product line.

Creating the Perfect Environment for Software Development

If everything is in place, we should be able to create software without flaws on time and under budget as depicted in Figure #3. But that is not the case. Software continues to be problematic over time as the statistics mention the costs exceed \$22 Billion a year as of 2002. Ultimately the question is why don't we fully create error free software? Two parts can answer this question, the software itself either does not meet the specified requirements or the software itself lacks quality.



Increasing Software Costs with Increased Complexity

Impact to the Bottom Line

Generally, as new versions of software are created for a product most software will see a level of costs that increases beyond reasonable levels and the software is cancelled or re-written, again as shown in Figure #2. So let's assume that the requirements are accurate enough to provide value, the final issue then are the quality of the software itself. The ever increasing complexity and coupling of the software as diagrammed in Figure #2 creates unintended consequences that will impact the P&L negatively such as:

- 1) Increasing field defects requiring costly software updates (Increase P&L Cost)
- 2) An increase in the labor for testing teams/processes (Increase P&L Cost)
- 3) Unpredictable schedule slippage with limited features (Reduce P&L Profit)

The Unrealized Value

Financially, this generally translates into the fact that most development teams fight the increasing complexity of the software thorough increased testing defects and field defects verses trying to manage defect containment over time as depicted in Figure #4. This usually translates into a team that will expand their manpower to the size/complexity/coupling of the code base such that over half the year is spent in defect management while the other half of the year is spent in feature creation, Figure #5 shows this dynamic. If you assume a loaded cost of \$100K a year for the developer, that is about \$50K per year on the management of defects that were created by the developer when making the desired changes to the code base. That will translate to \$200 to \$400 per day per developer in unrealized activity on the cost side that could be used on the profit side.

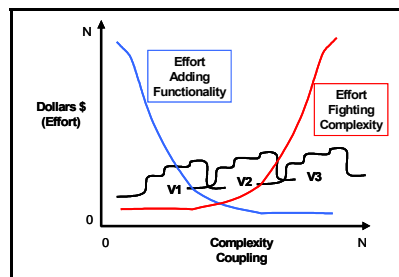


Figure #4: Increasing Code Complexity and Coupling Reduces Effort applied to adding Functionality (Features)

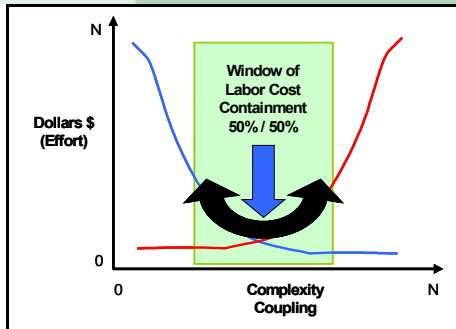


Figure #5: Labor Cost Containment Swings from Lower Complex Code Efforts (More Features) to High Complex Code Efforts (More Defects)

See Appendix A for further discussions on Current Development Process Techniques

Controlling the P&L

Static Analysis Effects the Bottom Line

With static analysis tools these unpredictable swings from focus on functionality to focus on complexity, as depicted in Figure #5, can be mitigated through the use of the control points such as defect control, security control, metric control and architecture control. This control will provide the visibility into the software complexity through the use of the metrics controls and architecture controls early in the development process to prevent the organization from entering into a focus on refactoring as depicted in Figure #6. With the use of static analysis of defects and security early in the process to provide an insurance net for any defects that might escape due to misunderstandings the software context when changing the code as depicted in Figure #7.

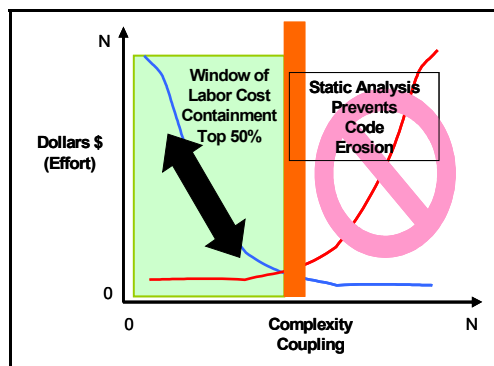


Figure #6: Labor Cost Containment is enforced to remain on the Left Hand Side through Static Analysis Tools

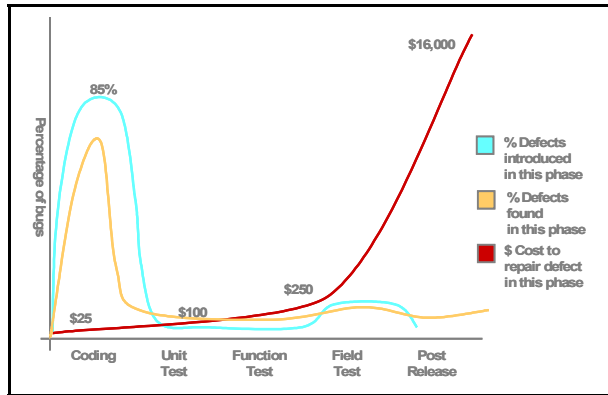


Figure #7: Defect Identification is moved from Black Box Testing to White Box Testing at the Developers Desktop

This process change with automated static analysis tools can bring defect testing up to the point of defect entry (See Figure #7) allowing the developer to create unit testing to verify their intended functionality, perform automated white box testing (Static Analysis as described in this paper) to ensure there are no logic errors and perform automated component testing managed by the metrics controls and architecture controls to ensure complex components that are highly coupled are tested thoroughly. This pro-active effort can make the cost side of the P&L become more manageable and predictable as shown in Figure #8. See Appendix B for detailed labor savings from using the White box testing prior to Black box testing.

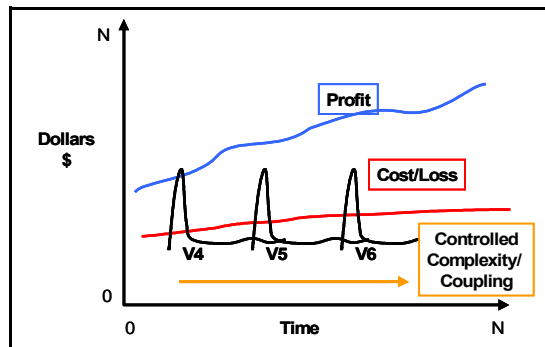


Figure #8: P&L impact of Defect Identification with Automatic White Box Testing Static Analysis

Therefore, this static analysis technology will:

- 1) Decrease field defects reducing the number of software updates (Lowers P&L Cost)
- 2) Decrease in the labor for testing teams/processes (Lowers P&L Cost)
- 3) Assist in providing predictable full featured scheduled releases (Increase P&L Profit)



Return on Investment (ROI)

With this static analysis technology the goal is within reach today since the overall product costs and project integration risks have been mitigated such that the return on investment can exceed 1000%. This assumes that the opportunity costs of \$50K per year per developer and the overall software costs and process costs are about \$5K-10K per developer for the first year and \$2-4K per developer per year after the first year, such that the break even period can be reached in less than 30-60 days. Thus giving a clear set of technology to manage the bottom line:

- 1) Controlling the P&L (More Profit potential and less Loss potential)
- 2) Controlling the Risks that Impact the P&L (Visibility through Metrics and Architecture)

See Appendix C for complete ROI calculations and Costing Graphs.

Appendix A: Current Development Environments

Current development efforts for containing complexity and coupling are limited. Some organizations are able to keep close tabs on these metrics but for the most part they are difficult to obtain, thus costly, and usually are not timely since it takes several days to capture. Thus most organizations abandon this effort and then focus on trying to be better at finding the defects through testing after a development build, also known as Black box testing. Thus they tend to put more effort on the right side of the complexity/coupling graph as shown in Figure #9.

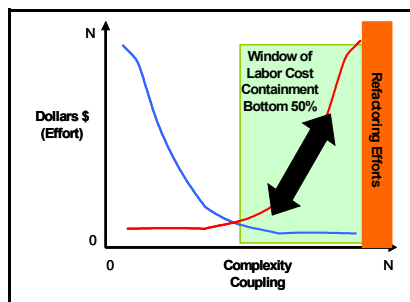


Figure #9: P&L impact of Defect Identification with only using Black Box Testing and Refactoring

The development management then places their decision making on the Black box efforts and attempts to track the defect rates that open in testing and the rate at which the defects are closed and then attempts to manage to a reasonable release date. This is best shown by a typical Testing Defect Management chart in Figure #10.

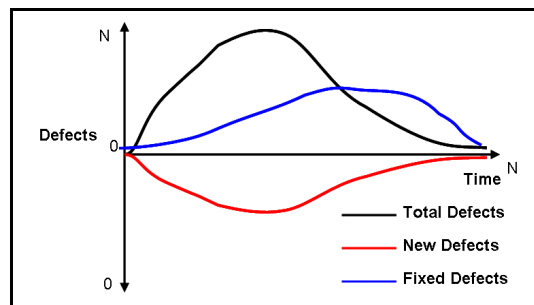


Figure #10: P&L impact of Defect Identification with only using Black Box Testing and Refactoring

This typical chart is usually all a manager has to gauge the success of their software development efforts. And as Figure #11 shows this is the EXPECTED result if all people, assets and supporting software is in place. But as we have mentioned in the body of this paper, the only variable that remains is the complexity and coupling of the software itself. As Figure #12 shows the realities are that with a growing complexity and growing coupling is a growing backlog of defects and an acceleration of new defects found and therefore the more reliance on black box testing to capture these defects.

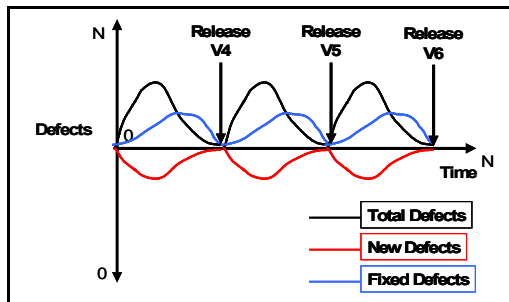


Figure #11: Expected Software Development defect removal graphs for a series of releases.

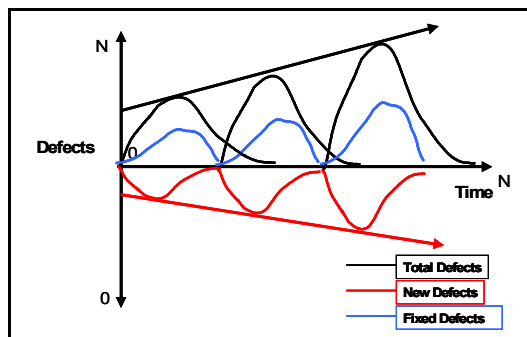


Figure #12: Actual result with ever growing backlog of defects as the code base increases in Complexity and Coupling resulting in a growing acceleration of new defects found.

This is what most development efforts see today, if there is no effort to consistently measure and manage the complexity and coupling in the system over time. When the schedule is pressing for a release and there is only measurement on defects over time the focus will always go to what is measured, removing defects discovered out of the testing cycle.

What can be done to better improve this process? What can be done to move out of this vicious and never ending cycle that can cost a company precious time and will eventually directly hit the P&L? See Appendix B for the discussion on how automated white box testing can solve this issue.

Appendix B: White Box vs Black Box Testing

The industry accepted options for defect removal are costly in terms of labor and time as shown in Figure #13. Black Box Testing, Code Reviews and Unit testing all require increasing amount of effort as the code increases in complexity and coupling. The obvious answer is to move the efforts into the development cycle and use automated solutions to ferret out the defect so they can be removed prior to black box testing efforts. But today these efforts are primarily performed through a manual process such as code reviews or many organizations are using such development practices as eXtream Programming (XP) (which is actually a constant code review with two people coding together in pairs).

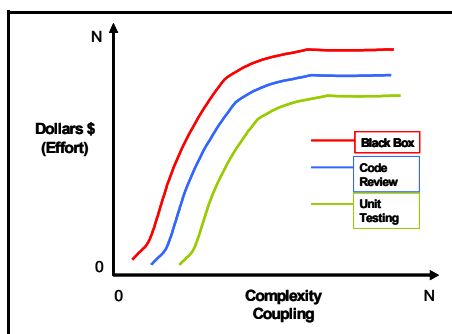


Figure #13: Comparison of Effort for Black Box Testing, Code Reviews and Unit Testing

With the use of automated tools within the development environment the effort vs complexity graph can change such that comparative effort will decrease over time. Why is this? Since the tools are automated the majority of effort is initiated at the point of introduction and as the automation is performed the increased complexity and coupling will not require any additional effort to perform the analysis as shown in Figure #14.

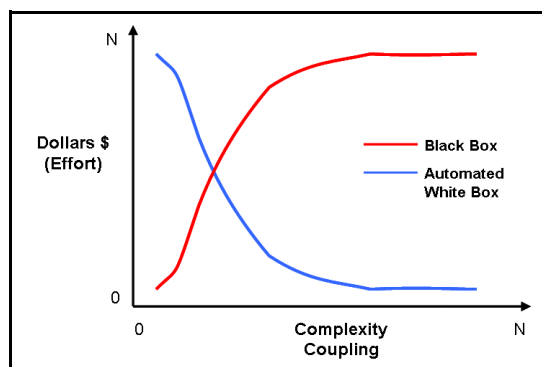



Figure #14: Comparison of Required Effort for Black Box Testing vs Automated White Box Testing



Why is this significant? Because the graph presented in Figure #7 is now possible without an introduction of additional manual processes and therefore additional costs and risks. This gives rise to the potential for all development efforts to engage in managing their software efforts to achieve a consistent behavior as depicted in Figure #8.

What has changed from before? Why has this not been “discovered” prior? The realities are that static analysis ALONE, without metrics and architecture, only points out the problems and does not provide the mechanism for real change. With metrics and architecture control ALONE they cannot fully measure the most problematic defects that no metrics or architecture can manage and that is the defect that has NO TESTABLE TESTCASE which can be defined to find the defect PIOR to the release.

Such that Static Analysis for Logical Errors and Security Errors within the context of White Box testing provides the CATALYST to effectively manage the metrics and architecture over time providing the P&L benefits described in this paper.

Appendix C: ROI Calculations for Static Analysis

Calculating ROI can be a difficult proposition since the value is obvious but the unknown impact to the development process can make a wise investment a bad choice if the tools and process changes do not take when introduced into the organization. With this in mind it is critical that the vendor supplying the static analysis technology provide the project planning mechanisms and services to mitigate this risk. With that being said we can assume that a wise choice has been made regarding a vendor that has a proven track record and documented processes and procedures to manage implementation and adoption risks. And we will assume for this exercise that those efforts will be about 2x the developer costs.

If the actual per developer costs for this technology is \$2000K to \$4000K the costs for introduction and process change impact should be about 2-3 times this amount. For this analysis we will use 2+ times or \$10,000 in total.

Additionally, the previous labor input articulated in the paper indicated 50% of the time a developer is devoted to the efforts of removing defects. With the use of static analysis we would expect this amount of time to be reduced to 10% to 30% of a developer's time on fixing defects. For this paper we will use 25% as the target range for dedication on fixing defects. This would mean that if the current labor costs are about \$100K a year and current effort on defects is 50% then the yearly costs for defects is about \$50K or \$400 a day. With the use of static analysis the goal would be to reach a point that defect costs per day would be \$200 or 25% of the overall effort. This means that a development team should be saving a loaded cost of \$200 per day per developer. Figure 15# shows this effect of savings in a graph of savings vs developers.

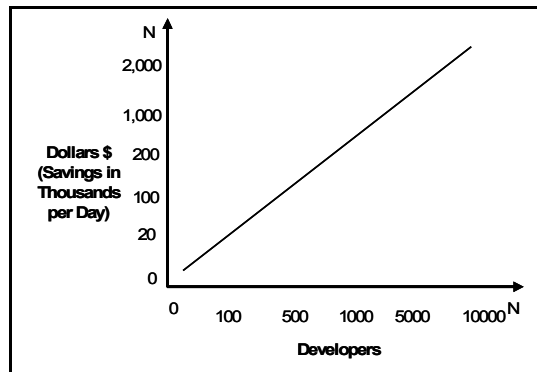


Figure #15: Savings vs Developers per day

It is clear that the potential savings is significant for organizations that have over 50 developers in house and the savings are astronomical for organizations that have over 1000 developers in house. So why has this technology **NOT** been widely adopted yet as of the beginning of 2005? First it is new technology with the advancement of computer processing power and cheap memory the required CPU cycles and the memory footprint can be handled by even high end laptops where

several years ago it was not even possible. And finally the processes and procedures on how to properly implement and leverage this technology is finally becoming codified and established. With that in mind what is the costs for the first year assuming an introduction of this technology costing \$10K per developer? Figure #16 shows this relationship between the savings and the costs in the first year the best.

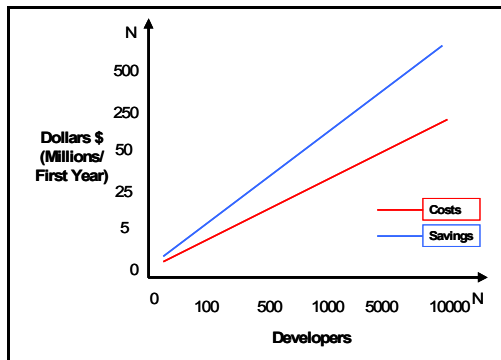


Figure #16: Savings and Costs in the first year using Static Analysis tools for project Control.

So ultimately this will translate to about 30-60 days for a payoff. Where \$10K cost per developer in the first year with \$200 per day savings is a 50 day payback. And then every year after that the costs will drop to under \$5K per developer and in some cases where the vendors sell their software as a capital expense that dollar figure can drop well below \$1K per year per developer.

And finally, what does this provide in terms of schedule impact? If almost 25% of a developer's time can be saved from doing defects they can be doing other things such as adding functionality and improving code quality. If we assume that a developer works 250 days a year on average then we can save 62.5 days of effort per year. And as our final Figure #17 shows this manpower savings is significant and will positively impact the P&L: as described in this paper.

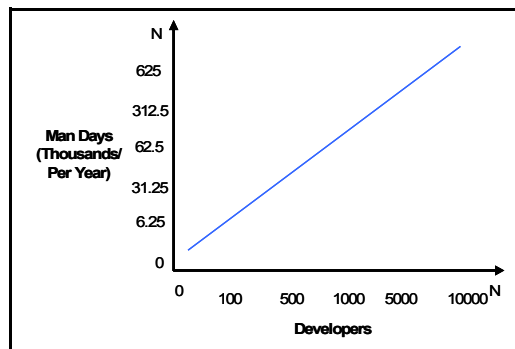


Figure #17: Manpower Savings per Developer per Year.