

# ROI Models for Static Analysis

*Making “a penny saved is a penny earned” come to life*

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

## Executive Summary

One of the more difficult aspects of managing software is attempting to estimate the completion of that software and the cost of the software under development. The second most difficult or frustrating part of managing software is attempting to build return on investment (ROI) models that help in decision making from adoption of new processes to acquiring and integrating new software tools. This paper will deal with defining several models for ROI evaluation based on Static Analysis tools.

Several methods of calculating ROI have been suggested historically and most of the data for these calculations come from Capers Jones work on gathering software project data for over 30+ years. With the evaluation of the cost of defects in test to be about \$500 per defect found to the cost of field defects to be about \$15,000 per defect found. Where the introduction of Static Analysis can promise to eliminate the need to find a whole class of defects before entering test or being released to the field.

The following are general models that can be used to create an ROI for Static Analysis:

Use of multiple project data points for equivalent comparisons

- 1) Use of single build of static analysis results
- 2) Use of historical data for smoking gun analysis
- 3) Use of historical builds over 1-2 year period
- 4) Use of benchmarking based strategies ( See Khaled El Emam’s work)

This paper will attempt to define each method that is used and advocate a method that will fit a customer’s needs depending on their organizational awareness. In most cases several methods will be used to analyze the true ROI as the number can be averaged over several methods.

Several assumptions regarding these calculations need to be stated:

- 1) Other ROI methods may exist and have legitimate assessments (They may get added later)
- 2) The dollar values may vary within an organization (\$500/defect test - \$15K/ defect Field)

The following sections will outline each ROI method and their advantages and disadvantages.

### 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](mailto:sales@full-knowledge.com)



**Equivalent Comparisons Analysis**

Many customers ask the question “What would the minimum buy-in for Static Analysis products if we are only concerned with finding and fixing software defects?” This question can only be answered in a precise manner by assessing the code base in its entirety. This is an impractical exercise for customers at this stage since they are in the process of evaluation. This section will focus on the industry standard defect density rates and utilize a simple test to define the proposed ROI.

For this analysis several industry data points are required:

1. The average cost to find and fix a defect in testing is roughly \$500/bug. (Software Assessments, Benchmarks, and Best Practices, 1996, Capers Jones)
2. U.S. Average Defect Rate FOR RELEASED SOFTWARE - 5.9 to 7 defects per thousand lines of code, which means a system with 1MLOC (Million Lines of Code) has about 5,900 to 7,000 defects. (Capers Jones)
3. Finding and fixing a defect at a developers desktop is roughly \$25 / bug. (Capers Jones)
4. U.S. Average Defect Rate BEFORE TESTING can exceed 120.8 defects/KLOC. Even at the injection rate for the top 1% of software developers, a 1MLOC system would enter compiling and testing with 11,000 defects (Effective defect rate of 11 defects/KLOC). That number goes to roughly 62,000 defects for the top quartile of developers (Effective defect rate of 62 defects/KLOC). Source: Developers Group: An analysis of data on more than 8,000 programs written by 810 industrial software developers (Data from SEI Fellow Watts Humphrey's article "The Quality Attitude" see [http://www.sei.cmu.edu/news-at-sei/columns/watts\\_new/2004/3/watts-new-2004-3.pdf](http://www.sei.cmu.edu/news-at-sei/columns/watts_new/2004/3/watts-new-2004-3.pdf))


“We now know how many defects experienced software developers inject. On average, they inject a defect about every ten lines of code. An analysis of data on more than 8,000 programs written by 810 industrial software developers is shown in Table 1. The average injection rate for these developers is 120 defects per KLOC, or one defect in every eight lines of code. Even the top 10% of the developers injected 29 defects/KLOC and the top 1% injected 11 defects/KLOC. Even at the injection rate for the top 1% of software developers, a 1,000,000 LOC system would enter compiling and testing with 11,000 defects. More typical developers would have 120,000 defects in their products.

Table 1. Defect Injection Rates for 810 Experienced Software Developers

Group	Average Defects per KLOC
All	120.8
Upper Quartile	61.9
Upper 10%	28.9
Upper 1%	11.2

That is why most large programs, even after compiling and testing, have from 10 to 20 defects per KLOC when they enter system testing. For a 1,000,000 LOC system, that would be between 10,000 and 20,000 defects. With current practices, large software systems are riddled with defects, and many of these defects cannot be found even by the most extensive testing. Clearly, while testing is essential, it alone cannot provide the quality we need to have secure systems.”

“The Quality Attitude”, [http://www.sei.cmu.edu/news-at-sei/columns/watts\\_new/2004/3/watts-new-2004-3.pdf](http://www.sei.cmu.edu/news-at-sei/columns/watts_new/2004/3/watts-new-2004-3.pdf)  
SEI Fellow Watts Humphrey



Based on the industry standards of 6-7 defects/KLOC for post release and 62 defects/KLOC for pre-testing there can be some initial ROI assessments to find the customer benefits of using static analysis. With a system that is a size of 1 MLOC the rates then are about 6000 defects at release time and 62,000 defects prior to the testing phase. This means that on average a 1 MLOC system will spend about \$500/defects \* (62,000 defects – 6000 defects) for testing, which is about \$28 Million per release. Which means that if two releases are performed a year, the testing can reach to be about \$56 Million in costs. This costs includes the time taken by the testers and the time taken by the development team fixing bugs and all the tools used to test the system itself.

If a portion of those defects were captured prior to testing, lets say 20%, the value of those caught defects would be:

$(20\% \text{ caught by static analysis} * 56,000 \text{ defects}) * \$500/\text{defect} = \$5.6 \text{ Million}$   
So per release there can be about \$5.6 Million in value found before testing or about \$11.2 Million for the year if two releases are scheduled. So an effective reduction of defects could occur from 62 defects/KLOC to 50 defects/KLOC or a reduction of 12 defects/KLOC.

For the field level defects we can assume that some portion of the 5.9-7 defects/KLOC will become critical issues that can cost an average of \$15,000+ to repair. In this case, the reduction of 1-2 defects/KLOC can have significant payback. A reduction of 1 defect/KLOC can result in removing 1000 field defects and making the effective defect rate 4.9-6 defects/KLOC. This value can be calculated at :


$1000 \text{ field defects removed} * \$15,000+/\text{defect} = \$15 \text{ Million}$   
So at \$15 Million a release and two releases per year that can be about \$30 Million in savings.

With both of these calculations, we find the potential benefit could be:  
 $\$30 \text{ Million} + \$11.2 \text{ Million} = \$41.2 \text{ Million}.$

But this is all paper savings and the real value will be the ability to perform more releases a year improving quality and increase product features. Let's say the bottom line savings will be as low as 10% of the value or \$4.12 Million.

Most software tools should have an ROI of about 1000% or more for the introduction of the technology to have effective payback and management of risks that might occur when introducing new tools. If a 1000% return is a factor of 10, then the overall cost for a project of this nature should be:  
 $\$4.12 \text{ Million} / 1000\% = \$412,000 (+/-\$200,000)$

Where this stated “budget costs” are the costs of the tools themselves plus the internal/external labor costs of introducing and integrating those tools into the normal process.



How do you find this type of analysis with an existing code base? Take the code base at hand, select a version that was prepared for testing (but was not tested) and a version that was delivered to the field (after it was tested). Run the static analysis tools against the two code bases and verify the effective “hit” rate, usually about 80%+ and then calculate the value based on the methods described above by the density rates that are obtained from the Static Analysis analysis.

Advantages of this approach:

- 1) Views the power of the tool as a pre-testing scrubber
- 2) Compares defect rates of Static Analysis before and after testing
- 3) Accurate comparison of customer performance against industry norms

Disadvantages of this approach:

- 1) Requires two builds, one before test and one after test.
- 2) Does not have a historical trending component, all code releases prior to testing vs after testing.
- 3) Suffer from first run, defects that can report excessive data vs the use of injection rates


This approach is one of the more favored approaches since it measures the code base against industry averages and can supply an accurate view of the cost saving at the point of testing as well as field defect savings.

### Single Project Static Analysis

The most used form of ROI analysis for Static Analysis is the single project build static analysis approach. This requires the need to perform the most accurate analysis on the code base as possible after the code has exited the testing phase. After the analysis is completed, the results would be searched for defects that would be classified as critical, important, non-critical and false positives. The critical defects then would be the ultimate find since they can be related to real world defects that can be traced to potential expenditures that would have to occur if the defect was actually found in the field.

The remaining issues found are interesting in the element of costs that might be incurred when the issues are fixed and the risk is low they would occur in the field. In this case we can use the previous numbers as before:

1. The average cost to find and fix a defect in the field is roughly \$15,000/bug. (Software Assessments, Benchmarks, and Best Practices, 1996, Capers Jones)
2. U.S. Average Defect Rate FOR RELEASED SOFTWARE - 5.9 to 7 defects per thousand lines of code (Capers Jones)
3. Assume 10% of those in the field are Critical Defects (Rule of Thumb), in the real evaluation an analysis of all issues would determine the actual number.



So in this case we can take the number of defects per KLOC and multiply this by the size of the code base, let's use 1MLOC again. That would be then  $7\text{Defects/KLOC} \times 1000\text{KLOC} = 7000$  Defects in total. Which may represent the actual size of the defects found by the static analyzer. More than likely the analyzer has found 2000 to 3000 issues. Of those issues there may be 200-300 critical issues.

So we can define the value of the analysis as the number of critical defects times the value of the defects and in this example that would be:

$300 \text{ critical defects} * \$15,000 / \text{defect to repair} = \$3,000,000 \text{ dollars}$

If the code base is released two times a year this would mean the value of the analysis, just in critical defects can exceed \$6,000,000 dollars. For the Non-critical issues a lower dollar amount could be used since it will not be of significant problems. Let's use the dollar amount of the testing \$500/defect and assume that 700 are deferred or not interesting and so in this example the value would be:

$2000 \text{ non-critical defects} * \$500 / \text{defects to repair} = \$1,000,000$

Which means that again the value over a year's time will be \$2,000,000 since there are two releases. Which means that the overall value of this analysis is  $\$6\text{M} + \$2\text{M} = \$8\text{M}$ . As before, the return should exceed 1000+% for the introduction of a process and tool to a software development group.

$\$8\text{M} / 1000+\% = \$800,000 (+/-\$200,000)$

So the cost of introducing this technology should not exceed \$800,000 (+/- \$200K). Where these costs are the costs of the tools plus the internal/external labor costs of introducing and integrating those tools.

Advantages of this approach:

- 1) If only a single run exists an ROI can be performed
- 2) Simple ROI Calculation
- 3) Common Technique across vendors

Disadvantages of this approach:

- 1) Single data point, meaning high possibility of single build to have false positive
- 2) VERY labor intensive, all issues must be reviewed to find the set of "Real" issues
- 3) Open for subjective analysis by developers as to what is critical or not

This approach is the LEAST favored approaches since it does not measure the code base against industry standards and relies on the very programmers that wrote the software to determine the VALUE of a defect. Additionally, it relies on ONE build and assumes that this one build is representative of the all the builds for the software.

## Smoking Gun Analysis

One of the most interesting ROI analyses is the “Smoking Gun” analysis. This analysis uses previous builds and current bug tracking systems. This tends to be the favoured approach with systems that are hardware oriented and software systems that are hard to replace. The payback can easily exceed the million-dollar mark with only one juicy find. Additionally, sampling from the bug tracking system compared to the static analysis results can give general indications of real defect rates.

This concept of a Smoking Gun can also apply to metrics and architecture, where as defects that are found in the filed often have metric relationships such as complexity and coupling as well as defects in the architecture such as API circumvention through the use of externs or other methods. Giving some analysis that can show real world relationships to vague concepts of metrics and architecture.

The specifics of this analysis can be reviewed in the paper called:

“The Smoking Gun: Finding the ROI in Large Software Development Projects with Static Analysis Solutions”

Full Knowledge Publication

Greg Spehar

Within this analysis the resulting ROI benefit and costing figures are as follows:

- 1) Defect Analysis
  - a. Benefit – \$5 Million
  - b. Costs – \$500K (+/- 200K)
  - c. ROI – 1000%
- 2) Metric Analysis
  - a. Benefit – \$11.4 Million
  - b. Costs – \$500K (+/- 200K)
  - c. ROI – 2280%
- 3) Architecture Analysis
  - a. Benefit – \$60 Million
  - b. Costs – \$1 Million (+/- 200K)
  - c. ROI – 6000%

So the overall costs of introducing this technique should not exceed about \$500 K (+/- \$200K) in software costs plus labor costs. The combination of Defect, Metric and Architecture provides visibility into the target code base that all the other methods cannot. So if complexity or architecture issues are suspect, this method should provide the framework to better understand the results.



### Advantages of this approach:

- 1) Single run with matching of historically reports field bugs
- 2) Includes metrics and architecture ROI calculations
- 3) Costly defects that have already been priced can provide organizational baseline

### Disadvantages of this approach:

- 1) Can become costly in labor if not automated
- 2) Single build dependent (Run several historical builds if possible 2-3 should do)
- 3) Requires input from testing, whom may not find this analysis funny or interesting

This approach is the on of the Luke Warm favored approaches since it requires detailed work to compare and find previous bugs. This approach may seem simple on the surface, but as it turns out it can be VERY tedious since the reported bugs in Static Analysis are not easy one to one matches to the reported bugs in other defect tracking tools.

In many cases, the resulting defect from Static Analysis may be part of several changes that are tied together or the defect is described in a non-intuitive manner. There are ways to work around this issue such as only record the file name and function name of the errors in question, and then do the comparison. Additional analysis will have to be done to verify the matching functions and errors fixed from the field (or testing) to match the errors found by Static Analysis.

## Injection Rate Analysis

One of the newer and more promising approach is the use of injection rates for developing the ROI. This approach can be very interesting since there are multiple versions of the software being analyzed over time which removes the “one off” analysis on a single code base. This approach is a result of the visibility of an architecture tool that can provide build over build visibility. With the proper analysis and review of the resulting issues, a consistent view of the defect injection rate can be established, providing a clear operating cost number that can be tied to an ROI.

1. The average cost to find and fix a defect in the field is roughly \$15,000/bug. (Software Assessments, Benchmarks, and Best Practices, 1996, Capers Jones)
2. Analysis is on one component that is 100KLOC and is one of ten (10) components
3. Assume, for this analysis a two year injection rate, starting from year 1
  - a. 300 defects found in time 0
  - b. 250 defects found in month 6
  - c. 200 defects found in month 12
  - d. 225 defects found in month 18
  - e. 210 defects found in month 24
4. A final defect rate without historical filtering is:
  - a. 200 defects remain in month 24

What is most interesting about this approach is that the numbers are consistent and real across multiple builds, and depending on the approach, ROI numbers may mirror actual operating costs of the software being analyzed.

This approach takes the following steps:

- 1) Add all the injection rates together
- 2) Find the defects that have been fixed
- 3) Find the costs of the fixed defects
- 4) Find the costs of remaining defects per year
- 5) Find the Net Present Value of the injection rate
- 6) Expand to evaluate the entire system

Use this set of steps for the component and then for the other 9 remaining components. The following is an example that will assist in the evaluation of this process. To perform this analysis the customer must have 4-5 builds across some amount of time. In this example we use 5 builds. The builds should be within one project such that build over build filtering can be utilized. This is important since we are looking for only the defects between builds that have been added. Then on the last build we will do one more build to find the number of defects that REMAIN. This will allow us to calculate the effective injection rate.

## Step 1: Add the Injection Rates

- 300 defects found in time 0
- 250 defects found in month 6
- 200 defects found in month 12
- 240 defects found in month 18
- 210 defects found in month 24
- 
- Total = 1200 defects generated over 2 years
- Average then of 240 defects every 6 months on 100KLOC

## Step 2: Defects that have been fixed

- Total defects minus remaining defects =  $1200 - 200 = 1000$  have been fixed

## Step 3: Cost of defects found

- At a cost of \$500 per defect in 90% of defects found
- At a cost of \$15,000 per defect in 10% of defects found
- $\$500 / \text{defect} * 900 \text{ defects} = \$450\text{K}$
- $\$15,000 / \text{defect} * 100 = \$1.5 \text{ Million}$
- Total cost of defects found = \$1.95 Million

## Step 4: Cost of remaining defects per year


- Two releases a year is 200 defects per release = 400 defects/year
- At a cost of \$500 per defect in 90% of defects found
- At a cost of \$15,000 per defect in 10% of defects found
- $\$500 / \text{defect} * 360 \text{ defects} = \$180 \text{ K / year}$
- $\$15,000 / \text{defect} * 40 = \$600 \text{ K / year}$
- Total cost of defects found = \$780 K / year

## Step 5: Net Present Value for Component

- At yearly costs of \$780 K / year
- Use an IRR of 10%
- Rule of thumb is 10x annuity
- $\$780 \text{ K / Year} * 10 \text{ (IRR of 10\%)} = \$7.8 \text{ Million}$

## Step 6: Expand to entire system or 10 components

- Component is one of 10 components
- Total defects fixed in past 2 years is  $(1000 * 10 \text{ components}) = 10,000$  defects
- Cost of fixing those defects should be about  $(\$1.95 \text{ Million} * 10 \text{ components}) = \$19.5 \text{ M}$
- Defect Injection rate is about  $(400 \text{ defects / year} * 10 \text{ components}) = 4000 \text{ defects / year}$
- At a cost of  $(\$780 \text{ K / year} * 10 \text{ components}) = \$7.8 \text{ Million / year}$
- Net present value at IRR of 10%  $(\$19.5 \text{ Million} * 10 \text{ (IRR 10\% factor)}) =$   
\$195 Million in value
- Estimated operational costs for the 10 components is between \$8 Million –  
\$10 Million / year
- At \$7.8 Million per year, the daily benefit to have the tool is:
- $(\$7.8 \text{ Million / year} / 2000 \text{ hours / year}) = \$31,200 \text{ a day (or } \$3,120 \text{ per component)}$
- With costs of \$500K (+/- \$200K) the breakeven point would be in about 10 to 25 days.



The most interesting aspect of this approach is the ability to predict some element of the customer's operating costs over time. The relationship between fixing defects and updating the code is so close that on older systems, the time spent adding new functionality could be about 30% of the time with the remaining 70% of the time dedicated to maintenance.

Additionally, with the relationships to the operating costs, a more refined ROI that relates the benefit to the daily operating costs allows the financial team to understand the operational breakeven point. In this example it is about 10 to 25 days. This approach plus another ROI calculation should provide enough data points to ensure the tools are purchased and integrated in a timely manner.

Advantages of this approach:

- 1) Detailed ROI in operating days
- 2) Highly accurate assuming the false positive rates are 80-90% or more
- 3) Shows customers in most cases that the number of issues fixed are injected

Disadvantages of this approach:

- 1) Multiple runs are required, usually a minimum of 4-6 depending on the timeframe
- 2) Clean and quick integration is required since 6 builds will take up to 6 hours or more
- 3) Customer is required to checkout 4-6 historical builds that are of the same type/quality


### **Benchmarking Based Analysis**

Utilizing the International Software Benchmarking Standards Group data set developed in 2003, Khaled El Emam presents a series of ROI models that are based on a set of companies and their reported defect costs and defect process. This approach is interesting since it allows for a comparison against other companies that may have similar processes and products. Khaled El Emam presents four (4) categories of analysis that can be defined from this data set:

- 1) Automated Defect Detection Model
- 2) Improved Maintenance Efficiency Model
- 3) Risk Assessment Model
- 4) Higher Reuse Model

A brief discussion of each model will be presented in the following section. A detailed analysis of each category can be found in Khaled El Emam's papers called:

“The ROI from Software Quality: An Executive Briefing”  
“Return on Investment Models for Static Analysis”



Where putting the value of the project into a spreadsheet that contains all the calculations defined in the paper can produce a reasonable ROI as shown in the previous ROI calculations in this paper. These calculations can be very complex and detailed, the remaining sections of this paper will outline some of the data used to perform the analysis and the final ROI calculations will be presented in the final paragraph.

### **Automated Defect Detection Model**

This model requires the following data:

- Effectiveness of automatic detection of defects, taking into account bad fixes – Example: 0.05
- Effort to find and correct a defect during automatic detection – Example: 1 hr
- Effort to find and correct a defect during post-release – Example: 50 hrs
- Effectiveness of testing – Example: 0.5
- Total project cost – Example: \$1 million

Where the benchmarking values are shown in each of the data examples. The calculations assume that these values are constant.

### **Improved Maintenance Efficiency Model**

This model requires the following data:

- Increase in good fixes during maintenance – Example: 1.1
- Effort to find and correct a defect during postrelease – Example: 50 hrs
- Effectiveness of testing – Example: 0.5
- Total project cost – Example: \$1 million

Where the benchmarking values are shown in each of the data examples. The calculations assume that these values are constant.

### **Risk Assessment Model**

This model requires the following data:

- Proportion of defects in the modules that were chosen for inspection – Example: 0.6
- Effort to find and correct a defect during post-release – Example: 50 hrs
- Effectiveness of testing – Example: 0.5
- Total project cost – Example: \$1 million

Where the benchmarking values are shown in each of the data examples. The calculations assume that these values are constant.



## Higher Reuse Model

This model requires the following data:

- Proportion of the system that is reused – Example: 0.1
- Proportion of development budget spent on discovering code reuse – Example: 0.05
- Total project cost – Example: \$1 million

Where the benchmarking values are shown in each of the data examples. The calculations assume that these values are constant. These calculations, as proven to be typical, can provide a very interesting result when compared to our project operating costs we calculated in the previous section.

If we assume a typical 1MLOC at 10 components has an annual operating budget of \$20 Million just for project maintenance and upgrades, using the spreadsheet from “Khaled El Emam” we find the “savings” to be about to \$6.87 Million. Comparing this to the injection rate results, which indicate a possible savings of \$7.8 Million per year that equates to \$31K a day. In contrast, this benchmarking method would indicate that the savings would be about:

$(\$6.87 \text{ Million per year} / 2000 \text{ hours per year}) * 8 \text{ hours per day} = \$27,480 \text{ per day}$

Which means that if project costs are about \$500K (+/- \$200K) the breakeven operating point is again 10-25 days depending on the integration and training costs incurred.

Advantages of this approach:

- 1) Simple project cost only approach
- 2) Comparative across many companies
- 3) Extensively researched using ISBSG as baseline data points

Disadvantages of this approach:

- 1) One Dimensional, spreadsheet is not open for change to other factors
- 2) May not represent unique industry or company process
- 3) Does not use data generated from static analysis

This provides a fifth and final approach to ROI calculations that all seem to be very consistent across the different methods. The final use of ROI calculations will always include the use of 2 or more of the methods defined within this paper. This analysis will give a general model that is specific to the problem at hand and provide confidence in the analysis to ensure the purchase and integration of static analysis tools will payoff in the long run.