

Roll-Out Plans

Planning a Corporate Installation: Who does What, When and How

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

Executive Summary

This document articulates the processes and procedures to fully roll out and integrate Static Analysis tools into your development environment.

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

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.

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

Phase 1a – Critical Defect ROI Proof (1-3 Months)

Critical Defect Control Integration work: 1 Day to 1 Week per Project

The setting up the initial meetings with the Vendor's Sales Engineers and Vendor's Account Representatives should have occurred prior to this point. The target organization should have made a small investment to fully integrate the initial tool into the development process for a worthy project. The following is the integration effort:

Who should be involved:

- Architects – The architects for the application being analyzed.
- Team Leaders – Team leaders for the application being analyzed.
- Build Engineers – Build Engineers for the application being analyzed.
- Vendor's Professional Services – Mentoring target organization's efforts

What needs to be done:

- Use KMC to create a build to obtain Knowledge Base and Critical Defect Analysis
- Analyze the application build scripts
- Decide on the build approach (See paper on Vendor's Integration Approaches)
- Verify build approach
- Solidify Build approach
- Build documentation to record build approach
- Verify integration into production
- (Extra) Perform defect citing on Vendor's results (perform 3-4 days of citing)

(Critical Components can be reviewed within this time frame. Additional time may be required to assess the remaining defects generated by Vendor.)

When should it be done:

- First day or week generates a working prototype integration and initial data
- Follow-on time is devoted to integration into production and defect citing

How should it be performed:

- Build Times:
 - Daily Builds – Building the application using the Static Analysis tool at night
 - Weekly Builds – Building the application using the Static Analysis tool once a week
 - Monthly Builds – Building the applications using the Static Analysis tool once a month
- Reporting:
 - Review reports that show potential defect data
 - Use spreadsheets and email to inform developers

- Focus on most critical issues (Memory Leaks and Null Pointers)
- Build over build will filter all issues historically between builds
- Futures:
 - Improved defect citing capability
 - Manage bugs/issues build over build
 - Assistance in filter management

Note: If the integration is performed as expected, additional projects may be integrated into Critical Defect Management at Phase 1 since the proposed timeframe is 1-3 months.

Two Part Focus

The defect analysis tools from the Vendor actually provide three types of defects:

- 1) Critical Defects – As defined by critical memory errors.
- 2) Security Defects – As defined by critical security vulnerability errors.
- 3) Maintenance Defects – As defined by interface violations, header file issues, compile expansion problems, dead code issues and clustering problems.

This paper focuses on the first of these three since critical defects can provide the most available ROI when integrating the Vendor tools. The security and maintenance defects are dealt with in the same manner as the critical defects so Phase 1 can just be repeated to gain the value of the maintenance defects with little or no additional integration effort. Just turn on those areas of interest in the configuration file and run the analysis. For a complete listing of the defects found please see the Static Analysis tool documentation supplied by the Vendor.

Driving the Value

Once the integration is successful and the tools have been generating results for several days (assuming a nightly build process) the assessment of those results need to occur into three basic categories (which can be further sub-divided if desired):

Identified Critical Defects – These are messages that have been identified by the Vendor and have been declared to be of GREAT interest, as they will effect testing and field operations. These are issues that can be considered to be high-impact defects that should be fixed prior to the next testing cycle and prior to the next release to the field.

Identified Non-Critical Defects – These are messages that have been identified by the Vendor and have been declared to be of interest, but they will not greatly effect testing and/or field operations. These are issues that can be considered to be lower priority defects that will be scheduled for later development cycles.

False Positives – These are messages that have been identified by the Vendor and have been declared to be of no interest by the development team or quality management personnel.

Note: In many cases some “false positives” might become issues later as the team understands the interactions of the code and as issues are found in testing or in the field. In other cases, some “false positives” are in fact maintainability issues today but the development team wants to defer attending to them to a later product release. This information would further “hone” the assessment of issues into the “false positive” category.

Track the Value

The results of phase 1 are tracked for some amount of time so that an initial assessment of the ROI can be derived. Focusing on those issues that are considered to be Identified Critical Defects and Identified Non-Critical Defects and assigning a value to each issue will assist in the ROI calculation. For example if a set of builds for a week result in 100 Identified Critical Defects we can assume that 90% of those defects would have been caught in test while 10% might have been caught in the field. The value of those issues caught before test can be \$500 per defect with \$15K for those issues that reach the field. So it would result in the following calculation:

Cost of Defects Found =

$$\begin{aligned} & \# \text{ of Defects Found before Test} \times \text{Cost per defect before Test} \\ & + \# \text{ of Defects Found before Field} \times \text{Cost per defects} \end{aligned}$$

before Field

So for our example we have value of $90 * \$500 = \$45K$ plus $10 * \$15K = \$150K$ or a total of about \$200K within a weeks development effort assuming the amount of defects found.

Using this information along with the Smoking Gun paper will provide the necessary information to generate a reasonable ROI for your project. Therefore the objectives within the Phase 1 project plan can be achieved within a 1-3 month timeframe. Phase 2 provides the additional framework to implement the Metric Management tool and can ensure the successful implementation of the Critical Defect Management Rollout.

Phase 2 – Metric ROI Proof and Critical Defect Rollout (1-3 Months)

Metric Control Integration Work: 1 Day to 1 Week per Project

The target organization should have made a growing investment to fully integrate the first Critical Defect Control Management tool and should have identified a project (more than likely the original pilot project) that contains Metric Management concerns. The following is the integration effort:

Who should be involved:

- Architects – The architects for the application being analyzed.
- Team Leaders – Team leaders for the application being analyzed.
- Build Engineers – Build Engineers for the application being analyzed.
- The Vendor Professional Services – Mentoring target organization's efforts

What needs to be done:

- Use a previous build
- Gather Metric Documentation as defined by the team
- Review the Vendor's provided metric management information
- Set metric goals and objectives
- Run analysis using the Metrics Tool
- Define the metric decisions per documentation and architect's understanding
- Define multiple metric analysis if additional metric views are relevant
- Perform second run with same code and applied metric decisions
- Identify metric anomalies that are critical
- Identify required changes to improve the metrics that are NOT critical
- Schedule the code changes with Project Management/Dev Schedule
- Generate metric rules for Static Analysis tool analysis
- Generate the Metrics Tool configuration
- Update the Build scripts to include the use of the following:
 - New metric configuration rules
 - New the Metrics Tool generation
- Verify Automation works properly
- Implement change into Production Environment
- Verify change works in production

When should it be done:

- First day to a week generates a prototype integration and initial metric data set
- Follow-on time is devoted to integration into production and metric citing

How should it be performed:

- Build Times:
 - Consistent with current build times
- Reporting:
 - Review reports that show potential metric defect data
 - Use spreadsheets and email to inform developers

- Focus on most critical issues (Metric Violations and the Metrics Tool Trends)
- Build over build will filter all issues historically between builds
- Futures:
 - Defect citing tool will include metric defects
 - Manage metric bugs/issues build over build
 - Assistance in filter management

Note: If the integration is performed as expected, additional projects may be integrated into Metrics Control Management at Phase 2.

Driving the Value

Once the Metric analysis is successful and the tools have been generating results for several days (assuming a nightly build process) the assessment of those results need to occur into four basic categories (which can be further subdivided if desired):

Metric Anomalies – These are issues that are discovered during the first week of the metric evaluation and definition. These are the metrics that are found to exist that should not exist and make the system “brittle” in being able to withstand changes.

Metric Changes – These are the recorded issues that are found when defining that expected metrics verses the actual implemented metric signature of the application. These are changes that are less drastic as the anomalies in a tactical sense and more strategic in there sense of being able to improve the metrics to withstand the test of time and the associated changes that may impact the maintenance and operation of the system.

Metric Violations – Once the metrics are defined or a “Metric Transformation” has been defined the implementation of the configuration rules will enforce the use of the metric rules build over build. Such that after the anomalies and changes have been identified initially, no more anomalies and additional expansion of the metrics can occur by the developer. Thus the Static Analysis tool will find a “defect” based on the metric rules defined in the metric configuration file against the ongoing expanding code base.

Metric Control Charts – As a technology, predictive metrics for software development have been elusive in gaining consistent and accurate information that is timely. With the implementation of the the Metrics Tool product the on going collection and measurement can occur routinely without interference of the ongoing development effort. This non-intrusive integration build over build allows for the tracking and identification of software metrics over time. Thus, allowing for the potential to begin the process of assessing productive control charts that can assist in risk management and can be used to trigger activities to reduce the rate in which software defects would occur within a development cycle.



Track the Value

When the metric correction effort has been implemented and/or scheduled and the results are tracked for some amount of time an initial assessment of the ROI can be derived. Focusing on those issues that are considered to be anomalies and changes and assigning a value to each issue will assist in the ROI calculation. Thus if we use the calculations as defined in the Smoking Gun we will find the advantages within the area of risk management for metric violations, churn rates and defect rates from build to build.

As stated in the “Smoking Gun” there is two components that drive the value, defects caught before testing and defects caught before escaping into the field. Of which their value potentially can be \$10M and \$40M respectively within a 2MLOC application. Then breaking that into how many errors need to be “caught” by this method we can further define by dividing by \$500/testing defect and \$10K/field defect which equates to 4000 errors caught before testing and 800 errors caught before field tests or at total of about 5000 defects total.


If we assume a 20% churn rate per development cycle on 2MLOC we have about 400KLOC of code changing per development cycle. If we have 3 development cycles per year we have about 1.2 MLOC changing per year. Of that code the Metrics defects might catch some portion of the software that is considered “high risk” for generating additional defects based on the code’s complexity etc.

So at this point we can derive the “potential” savings by using Metrics management. If we assume that 5000 defects as described would enter the code base without metrics management that would be an effective density of 5000 defects/1.2MLOC or about a 0.4%. Where general defect densities tend to be in the 1-5% density rates.

Ultimately tracking files or components that have historically been problem children and focusing on reducing their metric signature will be the final arbitrary in defining the hard proof of the value of control charts in software development. And finally applying statistical tools on the relationships between the metrics and defect generation will prove to be the mechanism that will allow teams to focus on critical components.

So the “value” of every metric violation is dependent on the risk the section of code will be exposed to the changes that might result in a defect. Thus, when a change occurs in a troubled component (the metrics are out of the prescribed control limits) the effort to make the change would be equal to or could be applied to the effort to redefine the component to be within the prescribed metric control limits. The meaningfulness of this previous statement is the following:

*Any change applied to code or components that are outside the metric control limits could by definition be considered to be a defect and the reported violation would be worth \$500 to \$10K per reported **change**.*



Using this information along with the Smoking Gun paper will provide that necessary information to generate a reasonable ROI for your project. Therefore the objectives within the Phase 2 project plan for Metric Control Management can be achieved within a 1-3 month timeframe. The next part of phase 2 provides the additional frameworks to implement the Critical Defect Control Management tools to the remaining four example projects.

Critical Defect Management Rollout

The rollout of the remaining projects for Critical Defect Control for our example would happen in Phase 2. The rationale for this staging is to allow for the necessary infusion that it will take to ensure the managers and executives of the target organization have the proper support and understanding in which to change the company culture and paradigm on how to develop software. These decisions must be made within the context of working prototypes in which to point to for example successes and example ROIs.

Additional staging of remaining projects will further strength the data points as the teams report the effective ROI as the projects define their baseline and implement the build over build implementation of the Vendor's Static Analysis technology.

Since each project will more than likely have their own architects, team leads and build engineers the same objectives and goals apply to the remaining projects.

Project Timeframe

- *Projects #2, #3, #4, #5:*
- Total time is:
 - 1 day to 1 week integration + 1 week observation
 - 4 Remaining Projects
 - 4 days to 4 weeks total effort (Max 4 Weeks)
- Futures
 - Detailed home page for Management for Multi-Project Overview Defects
 - Detailed home page for Developers for Multi-Project Action on Defects

Phase 3 – Architecture ROI Proof and Defect Rollout (1-3 Months)

Architectural Control Integration Work: 1 Day to 1 Week per Project

The target organization should have made a growing investment to fully integrate the initial tool into the development process for the remaining 4 projects and should have identified a project (more than likely the original pilot project) that contain architectural concerns. Overly long development cycles, inability to test surgically, feature development in one area causing problems with other areas/applications are all symptoms of uncontrolled, not understood and overly complex software coupling at the highest levels in the product. A software architecture that is in this state is an ideal candidate for this next phase. The following is the integration effort:

Who should be involved:

- Architects – The architects for the application being analyzed.
- Team Leaders – Team leaders for the application being analyzed.
- Build Engineers – Build Engineers for the application being analyzed.
- The Vendor's Professional Services – Mentoring target organization's efforts

What needs to be done:

- Use a previous build
- Gather Architecture Documentation as defined by the team
- Quickly sketch out what is believed to be the current software architecture.
- Open the Modeling Tool Architect
- Abstract the software files/directories into architectural components as per the sketch and architects understanding
- Define multiple models if additional views are relevant
- Perform assessment and identify anomalies such as API avoidance, API misuse, unwanted coupling, inverted dependencies, etc.
- Set improvement goals and objectives.
- Identify required changes to meet the goals
- Schedule the code changes with Project Management/Dev Schedule
- Generate Architecture Rules to prevent further erosion of the architecture
- Update the Build scripts to include the use of the following:
 - New Architecture Configuration Rules
 - New Architecture Model exported from the Modeling Tool
- Verify Automation works properly
- Implement change into Production Environment
- Verify change works as expected

When should it be done:

- First day to a week generates a working prototype integration and initial data set

- Follow-on time is devoted to integration into production and architecture citing

How should it be performed:

- Build Times:
 - Consistent with current build times
- Reporting:
 - Review reports that show potential architecture defect data
 - Inform developers (eg. Emails, problem reporting, spreadsheets, portal)
 - Focus on most critical issues (Architecture Violations and Use of Externs)
 - Build over build will filter all issues historically between builds
- Futures:
 - Improved defect citing capability
 - Manage bugs/issues build over build
 - Assistance in filter management

Note: If the integration is performed as expected, additional projects may be integrated into Architecture Control Management at Phase 3.

Driving the Value

Once the Architecture analysis is successful and the tools have been generating results for several days (assuming a nightly build process) the assessment of those results need to occur into three basic categories (which can be further sub-divided if desired):

Architecture Anomalies – These are issues that are discovered during the first week of the architecture evaluation and definition. These are the relationships that are found to exist that should not exist and make the system “brittle” in being able to withstand the current features and the maintenance of those features.

Architecture Changes – These are the recorded issues that are found when defining that expected architecture verses the actual implemented architecture. These are changes that are not as drastic as the Anomalies in a tactical sense, but are more strategic in the sense of being able to improve the architecture to withstand the test of time and the associated changes that may impact the maintenance and operation of the system.

Architecture Violations – Once the architecture is defined or an “Architecture Transformation” has been defined the implementation of the configuration rules will enforce the use of the architecture build over build. Such that after the anomalies and changes have been identified initially, no more anomalies and additional architecture changes can occur by the developer. Thus the Static Analysis tool will find the “defect” as based on the architecture rules as defined in the architecture configuration file.



Track the Value

When the architecture has been corrected and/or scheduled and the results are tracked for some amount of time an initial assessment of the ROI can be derived. Focusing on those issues that are considered to be anomalies and changes and assigning a value to each issue will assist in the ROI calculation. Thus if we use the calculations as defined in the Smoking Gun we will find the advantages within the areas of software development productivity, code management mobility and surgical testing.

This previous analysis within the Smoking Gun focused on two areas of interest, Code Management Mobility and Surgical Testing. Both of fixed value within the opportunity costs without assessing the ongoing savings over time. The ongoing savings over time can be only addressed within the context of a Boolean decision. Either one has the ability to perform Code Management Mobility/Surgical Testing or one does not have the ability.

The meaningfulness of this previous statement is the following:

*Unless all violations are dealt with and addressed at the point of entry or at the build integration point, the investment and efficiencies gained by maintaining a consistent architecture is eliminated when a **SINGLE** violation is allowed to enter the build.*

So the “value” for every architecture violation caught would range from how much was invested in obtaining the opportunity for Code Management Mobility and Surgical testing (which can range into the \$100Ks) to what the business value of this capability is over a period of time (which can range in to the many millions of dollars). Based on this initial assessment of the business impact, an additional “value” assessment can be defined with the impact of slipping a delivery date based on allowing an architecture violation to escape into the build, preventing the surgical testing of components which would result in extending testing times and thus extending the delivery date.

Using this information along with the Smoking Gun paper will provide that necessary information to generate a reasonable ROI for your project. Therefore the objectives within the Phase 3 project plan for Architecture Control

Management can be achieved within a 1-3 month timeframe. The next part of Phase 3 provides the additional frameworks to implement the Metrics Control Management tools to the remaining four example projects.



Metrics Management Rollout

The rollout of the remaining projects for Metric Control for our example would happen in Phase 3. The rationale for this staging is to allow for the necessary infusion that it will take to ensure the managers and executives of the target organization have the proper support and understanding in which to change the company culture and paradigm on how to develop software. These decisions must be made within the context of working prototypes in which to point to for example successes and example ROIs.

Since each project will more than likely have their own architects, team leads and build engineers the same objectives and goals apply to the remaining projects.

Project Timeframe

- *Projects #2, #3, #4, #5:*
- Total time is:
 - 1 day to 1 week integration + 1 week observation
 - 4 Remaining Projects
 - 4 days to 4 weeks total effort (Max 1 Month)
- Futures
 - Home page for Management for Multi-Project on Metric Violations
 - Home page for Developers for Multi-Project Action on Metric Violations

Phase 4 – Architecture Rollout (1-3 Months)

Architecture Management Rollout

The rollout of the remaining projects for use of the Architecture Control capability for the remaining applications for our example would happen in phase 4. The rationale for this staging is to allow for the necessary infusion that it will take to ensure the managers and executives of the target organization have the proper support and understanding in which to change the company culture and paradigm on how to develop software. These decisions must be made within the context of working prototypes in which to point to for example successes and example ROIs.

Since each project will more than likely have their own architects, team leads and build engineers the same objectives and goals apply to the remaining projects.

Project Timeframe

- *Projects #2, #3, #4, #5:*
- Total time is:
 - 1 day to 1 week integration + 1 week observation
 - 4 Remaining Projects
 - 4 days to 4 weeks total effort (Max 1 Month)
- Futures
 - Home page for Management for Multi-Project on Architecture Violations
 - Home page for Developers for Multi-Project Action on Architecture Violations



Phase 5 – Protection Rollout (1/2 day to 1-2 Weeks)

The phase 5 rollout of the Protection mechanism called Desktop Static Analysis for the developers and architects that are responsible for changing the code for our example could happen in parallel with phases 1-4 or as a final phase as shown here. The rationale for this staging is to allow for the necessary infusion that it will take to ensure the development team of the target organization has the proper support, exposure and understanding with this technology in which to accept a change in culture and paradigm on how to develop software by preventing the defects from occurring as early in the development process as possible. These decisions must be made within the context of a working prototype example in which to experience this different point of view in a non-intrusive manner and non-offending manner.

Since each project will more than likely have their own developers the project rollout would include the following requirements.

Project Timeframe

- *Projects #1, #2, #3, #4, and #5:*
- Total time is:
 - ½ to 2 Hours for integration of Desktop Static Analysis tool into IDE
 - Easy four step process
 - Demonstrate the use of the tool on existing code
 - ½ day for official training or mentoring
 - Ongoing feedback to architects and team leads on configuration controls
 - Man-Month effort for 100 person team:
 - Integration - ½ hour x 100 systems = 50 Hours
 - Training – ½ day x 100 developers = 50 days x 6 hours = 300 Hours
 - Total impact can range from 50 hours to 350 hours
- Futures
 - Home page for Developers for Multi-Project Actions on All System Violations



Appendix A: Rollout to Small Teams

For small systems that are under 500KLOC and have about 10-20 developers the time frames can be reduced into days of effort. All 1 day to 1 week implementations can be reduced to several hours to only 1 day implementations. Implementation of the Protection mechanism by using Desktop Static Analysis would remain the same. Since the number of people involved and the “volume” of code is greatly reduced the implementation times can occur in less than 1 day and the verification can happen in a short follow-on meeting.

So typically we will find the same value proposition but a much smaller scale:

Defect Control Implementation for under 500KLOC

- ½ day to 1 day Integration effort
- 1 Month Observation

Architecture Control Implementation for under 500KLOC

- ½ day to 1 day Integration effort
- 1 Month Observation

Metric Control Implementation for under 500KLOC

- ½ day to 1 day Integration effort
- 1 Month Observation

Protection Mechanism for 20 Developers

- 20 developers 1 hour IDE Integration = 20 hours or about 3 days
- ½ day training = two training classes of 10 developers each = 1 day of training

Complete Integration Effort for all control points should last 2-3 days for a small target organization.



Appendix B: Rollout to Very Large Teams

For large systems that are over 25MLOC and have about 1000+ developers the time frames can be expanded to break the problem down to meet the requirements specified in this paper. Which would require the projects broken into 5 – 5MLOC projects with about 100-200 developers per project.

If that is not possible there can be an extrapolation that could occur based on the medium project expectations. All 1-day to 1-week implementations can be expanded to the appropriate level. Where the ultimate integration predictor is the number of build scripts used and the complexity of the build scripts. The more complex and the larger the build scripts the longer the integration timeframe if the direct integration method is chosen. Working with the new build-map technology those times can be reduced significantly, see the Vendor's Integration Guide for what will work in your environment. Implementation of the Protection mechanism by using Desktop Static Analysis would remain the same regarding the multipliers. Since the number of people involved and the “volume” of code is greatly increased, the implementation times need to be addressed on an individual basis for realistic time frames.

So typically we will find the same value proposition but within a much larger scale:

Defect Control Implementation for over 25MLOC

- 1-2 Week Integration and Mentoring (1/2 to 1 full day per major build script)
- 3 Month Observation

Architecture Control Implementation for over 25MLOC

- 1-2 Week Integration and Mentoring (1/2 to 1 full day per major build script)
- 3 Month Observation

Metric Control Implementation for over 25MLOC

- 1-2 Week Integration and Mentoring (1/2 to 1 full day per major build script)
- 3 Month Observation

Protection Mechanism for over 1000+ Developers

- 1000 developers 1 hour IDE Integration = 1000+ Man hours
- ½ day training = 100 training classes of 10 developers each = 50 days of training

Total Integration Effort should be between 6-18 months for large application integrations.