

# Quality Management of Software Projects

## *Managing a software “Golden Egg” using static analysis tools*

By Greg Spehar ([spehar@full-knowledge.com](mailto:spehar@full-knowledge.com))

### Executive Summary

Managing quality within the context of software development has been an elusive goal with problems ranging from the quality of data that can be collected to the length of time it takes to gather the actual data. These factors and others will be further reviewed within this paper. The most important question that needs to be asked is, why is quality an important component of a software development process? The most interesting analogy is the idea that software could represent a golden egg where the software owns wealth or can generate the wealth, and the goose is the development team that delivers that golden egg. The fable of the “Goose with the Golden Egg” best describes this dynamic:

The Goose With the Golden Egg

“ONE day a countryman going to the nest of his Goose found there an egg all yellow and glittering. When he took it up it was as heavy as lead and he was going to throw it away, because he thought a trick had been played upon him. But he took it home on second thoughts, and soon found to his delight that it was an egg of pure gold. Every morning the same thing occurred, and he soon became rich by selling his eggs. As he grew rich he grew greedy; and thinking to get at once all the gold the Goose could give, he killed it and opened it only to find,—nothing.”

“GREED OFT O’ERREACHES ITSELF.”

<http://www.bartleby.com/17/1/57.html>

Æsop. (Sixth century B.C.) Fables.

The Harvard Classics. 1909–14.

Æsop was a slave who lived in Samos in the 6th century B.C. His moral animal fables have delighted young and old for centuries.

The Fable of the Golden Egg is a very appropriate story in that the real location of the wealth is not the egg itself, but the development team that is creating or maintaining the software or the “Golden Egg”. This concept is important since the “Quality” changes or quality tools support must be applied to the development team, not the code itself. Meaning that pushing the development team to the point of exhaustion or pushing the development team into areas that are not areas of expertise or replacing higher quality personnel with less expensive personnel may be a form of “killing the goose”. Where adding tools to assist the development team within the growth or management of the software “golden egg” will result in far more success and improvement of quality in the software itself.

#### 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)



If the fable for the Golden Egg best describes the concept of integrating quality into the development process, let's take a closer look at the definition of quality and see where the hot spots of change can occur:

“A good definition must let us measure quality in a meaningful way. Measurements let us know if our techniques really improve our software, as well as how process quality affects product quality. We also need to know how the quality we build in can affect the product's use after delivery and if the investment of time and resources to assure high quality reap higher profits or larger market share. In other words, we want to know if good software is good business.

...

In an influential paper examining views of quality, David Garvin studied how quality is perceived in various domains, including philosophy, economics, marketing, and operations management. He concluded that “quality is a complex and multifaceted concept” that can be described from five different perspectives.

- The transcendental view sees quality as something that can be recognized but not defined.
- The user view sees quality as fitness for purpose.
- The manufacturing view sees quality as conformance to specification.
- The product view sees quality as tied to inherent characteristics of the product.
- The value-based view sees quality as dependent on the amount a customer is willing to pay for it.”

Software Quality: The Elusive Target  
BARBARA KITCHENHAM , National Computing Centre  
SHAH LAWRENCE PFLEEGER, Systems/Software, Inc.  
January 1996, IEEE


For the purpose of this paper, to show the positive impact of static analysis on quality management, the fourth quality issue is where we will focus the discussion, “The product view sees quality as tied to inherent characteristics of the product.” This is due to the problem static analysis helps in correcting, the unintentional failure of the product that is based on mistakes performed by the developer or architect when adding or changing code. But ultimately, if the number of defects exceeds a certain threshold all categories of quality will be affected.

But why do developers make mistakes in such a way that they create defects where they affect “quality” and the defect then has to be discovered before or during testing? The answer does not lie in the competence of the developer, but more in the scope of knowledge about the change and the ability to fit that scope into the code and also consider all other scopes that may have been previously integrated.

“Inherent difficulties of software quality movement, which are interrelated to the inherent assumptions, in the field can be listed as follows:

- Solving the problem by its solution
- Increasing the uncertainties for decreasing uncertainties
- Delayed feedback
- Software metrics are process metrics too
- Information recursion
- Contextual relativity”

Assumptions and Difficulties of Software Quality Movement; Onur Demirors  
EUROMICRO 97. 'New Frontiers of Information Technology',  
Proceedings of the 23rd EUROMICRO Conference, Sept. 1997



Who gets the short end of the stick on this problem? Most people would say that the customer would get the short end of the stick. But that assumes that the customer does not have other options to purchase or use software that solves their specific problem. And in most cases there ARE other options. So the weight of this issue lies not on the customers or the executives of the software company creating the software, but on the very people (Usually project managers and product managers) charged with managing the team of developers to create a logical solution that will solve some identified problem.

“DO THE MATH. Software project managers today must be aware that each parameter - cost, schedule, staffing, functionality, and quality - is potentially critical. It is the customer - be they an end user for an in-house system or the marketing department for a software company - who decides what the proper balance is. It's also crucial to remember that the balance among parameters is dynamic and may need to be readjusted daily. After all, the business environment is likely to change in a dramatic, unpredictable way - and this can easily change the customer's perception of the importance of schedule, cost, and so on.”

WHEN GOOD ENOUGH SOFTWARE IS BEST

Edward Yourdon

IEEE Software

May 1995, page(s): 79-81, Volume: 12, Issue: 3

ISSN: 0740-7459

So now we know two truths, the goose is the development team and that quality efforts lie on the shoulders of those charged with guiding the development team. But the problem that appears next is that those charged with the management of the development team have a very limited view of what is really happening or what is occurring when the code changes.

“As discussed, a method has been devised to predict the types, frequency and number of errors remaining in the program. The approach not only permits us to measure the progress of the development effort, but also helps management focus on problem areas and potential problem areas. Since quality efforts must be preventive rather than corrective, identifying problems early helps develop more reliable software, keeps cost down and prevents unexpected delays in schedules.”


QUANTIFYING SOFTWARE QUALITY

Kenneth S. Hendis

ACM '81, November 9-11, 1981

Which brings us full circle to the need or desire of the people that manage the development team to pressure and implement processes or tools that increase visibility and encourage preventive efforts. As Steve McConnell states, the industry has been attempting to create a “silver bullet” for some time.

“Fifteen years ago in his classic article “No Silver Bullets,”<sup>1</sup> Fred Brooks predicted that no single tool or practice would produce an order-of-magnitude improvement in quality or productivity over a 10-year period. In spite of repeated “silver bullet” claims for innovations ranging from automatic programming to component-based development to object-oriented programming to CASE tools (the list is nearly endless), no single tool or practice has risen to the level Brooks described. Time has proved Brooks' prediction correct.



Software professionals have been burned time and time again by exaggerated claims for new tools and practices. My question is, Why are so many smart, experienced software professionals still so gullible about silver bullets?"

<http://www.computer.org/software/homepage/2002/03EIC/index.htm>

Steve McConnell  
IEEE Software

This leads us to Mr. Brooks himself stating the most obvious problem of the software industry is *plain visibility of a highly complex system*. And mapping that visibility effectively to the requirements in such a manner that correct decisions can be made when changing or adding software to a target software system.

"I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared with the conceptual errors in most systems. If this is true, building software will always be hard. There is inherently no silver bullet."

<http://www.computer.org/computer/homepage/misc/Brooks/index.htm>

Frederick P. Brooks, Jr.  
No Silver Bullet: Essence and Accidents of Software Engineering  
April 1987, Computer

With the introduction of static analysis this problem of visibility is being solved. And with a set of proper tools to automate that visibility and graphically view that complexity, this issue of visibility into the complexity of software can begin to be solved properly. This paper will outline how static analysis tools can assist in quality initiatives and answer the following questions:

1. How can we manage the "goose" without killing it?
2. What Static Analysis Tools are needed to manage the goose?
3. How does the process change to "Manage your golden egg"?
4. What is the financial impact of "Quality" and managing the goose?

### **Don't Kill the Goose**

In beginning to understand the problem of managing quality across an organization it is important to see that most managers and management executives always want more data. More information on results and on process such that when effectively collected adds additional overhead costs and can interrupt the people doing the "real" work. This reference shows the importance of better software quality through the use of better measurements:

"Software Quality via Better Measurement Metrics and Their Use

There are a variety of decision support systems and information systems that are available off-the-shelf for predicting the cost, schedules, and other strategic details whenever a software project is initiated. However, quality itself cannot be predicted just by employing such tools. Therefore, it is essential that organizations clearly define and quantify the quality that is desired from the software product.



To make realistic assessments about the quality of a software product, it is imperative that measurements be performed. This enables organizations to establish and regulate the levels of acceptable quality, predict quality, and also to continuously strive to improve quality. Collection of data about the process and the product is the primary way of monitoring quality in a number of organizations.”

Some Observations on Software Quality  
William A.Ward, Jr.

ACM Southeast Regional Conference archive  
Proceedings of the 37th annual Southeast regional conference

Article No. 2

Year of Publication: 1999 ; ISBN:1-58113-128-3

To better understand the quality of software as new tools or new processes are introduced efficient mechanisms must be deployed to measure and ensure previous decisions are kept honest. So it is no longer important to just observe the phenomena that occur when embarking on a quest to create the best and most popular golden egg. Additional tools must be present that can respond and signal the flock of geese (development team) as they fly to their intended goal. The following excerpt indicates that more automation is required to effectively manage the behaviour of a group over time.

"Negotiating stakeholder win-win relationships among software quality requirements is a technique that emerged during the 1990's in order to overcome the difficulties Based upon prior research, we start with an assumption that the Win-Win negotiation model is effective in surfacing and resolving system quality attribute conflicts between stakeholders. The following two research questions are addressed in this paper.

1. Can behavioural differences between stakeholders be observed within this application of the Win-Win negotiation model?
2. Do knowledge-based automated aids enhance surfacing and resolving system quality attribute conflicts?"

Where in the conclusion of the paper it states:

"1. Behavioural differences between stakeholders:

- Stakeholders usually accept reasonable solutions and escalate by exception only those conflict issues central to their individual interests;
- Users and customers are more active than developers when identifying win conditions. The active role shifts to the developer in the resolution portion of the cycle.

2. Enhancements provided by automated aids:

- QARCC and S-COST collectively surface a larger number of quality attribute issues and options than those raised manually by the stakeholders;
- Some of the QARCC/S-COST originated issues were false alarms. It was relatively trivial to filter out the specious items. This favors the potential of these automated aids to substantially augment the basic Win-Win model as applied to quality attributes."

23rd International Conference on Software Engineering (ICSE'01)

May 12 - 19, 2001

Toronto, Canada

Hoh In, Thomas Rodgers, Michael Deutsch, Texas A&M University

Barry Boehm, University of Southern California



An interesting point to note in this reference quote is that Barry Boehm, one of the largest contributors to software engineering research, has indicated that human operated tools and process alone are not enough, an additional mechanism is needed, one that is an “automated aid”. These types of mechanisms need to be cost effective and the set-up and learning time for these expert systems can be expensive in time and software. For any automation mechanism that will be integrated from the research lab into the business world, the mechanism must also scale and provide the data in a concise and timely manner. As Lori Clarke asks, “How do we show that it matters?”

“To convince industry that an idea is a reasonable technology transfer target, researchers must demonstrate its applicability. There are several ways to do this but each has its cost. For example, one can conduct an experiment demonstrating the superior qualities of a new technique. To be convincing, the experiment must be run on a large number of realistic programs by individuals who have the kinds of skills that industry would expect to employ for such tasks. Usually such experiments are run on an embarrassingly small number of extremely small programs by highly motivated, skilled graduate students who want to get their degrees. The experiments usually demonstrate the benefits of an approach but fail to convince any one of their applicability. To conduct a more realistic experiment would be extremely costly, and to be most effective would require industrial cooperation. Moreover, before such a demonstration could be undertaken, the research prototypes need to be improved to the point that they can stand up to “friendly” industrial use, and even this can be a major stumbling block. The benefits would be substantial, however. Researchers would learn where their ideas fail to scale and could direct their efforts to solving these problems. Eventually technologies would emerge that have been demonstrated to be effective at improving software quality.”

How Do We Improve Software Quality and How Do We Show that it Matters?

Lori A. Clarke

Department of Computer Science, University of Massachusetts

ACM Computing Surveys 28(4es), December 1996,

<http://www.acm.org/pubs/citations/journals/surveys/1996-28-4es/a203-clarke/>.

So this leads us to the ultimate question, “What is the cost of software quality?” In the quality circles of several years ago and the quality initiatives the answer has always been that “Quality is Free” or it should be free. Software quality, free as a bird, or should we say free as a goose, is the idea that at every step there is checking of quality by the people performing the work. As a manufacturing line monitors their quality as devices are put together or created, could there be a similar idea when creating software?

The idea of software quality within the category that we choose earlier, “The product view sees quality as tied to inherent characteristics of the product”, the answer would be yes, it can be “free”. There can be mechanisms to allow people working on the software change or problem to better manage their efforts and monitor the quality in the near term and from afar. As Sandra A. Slaughter et. al describe the cost of quality can be tied to the amount of labour required to monitor those quality indicators.

#### “Cost of Software Quality

The costs of quality as originally articulated by Juran and Gryna are those that would be eliminated if all workers were perfect in their jobs. Quality costs are important because every dollar and labor hour not spent on rework can be used for making better products more quickly or for improving existing products and



processes. The costs of quality are divided into two major types: conformance and nonconformance.

The cost of conformance is the amount spent to achieve quality products. It is further divided into costs of prevention and appraisal. Prevention costs are those associated with preventing defects before they happen. In software development, examples of prevention costs include the costs of training staff in design methodologies, quality improvement meetings, and software design reviews. Appraisal costs include measuring, evaluating, or auditing products to assure conformance to quality standards and performance. For software, examples of appraisal costs include code inspections, testing, and software measurement activities.

The cost of nonconformance includes all expenses that are incurred when things go wrong. Internal failure costs occur before the product is shipped to the customer. For software these include the costs of rework in programming, reinspection, and retesting. External failure costs arise from product failure at the customer site. For software, examples include field service and support, maintenance, liability damages, and litigation expenses.

So how can companies reduce the costs of software quality? A basic strategy is to drive failure costs to zero, invest in the “right” prevention activities to bring about improvement, reduce appraisal efforts as quality improves, and continue to evaluate and alter preventive efforts for further improvement [5]. The idea behind this approach is that real software failure costs can be measured and then reduced through the proper analysis of cause and effect. Elimination of root causes means identifying and permanently fixing defects as early in the software life cycle as possible, because the cost of correction increases the later in the software process the defect is discovered and corrected. As software failure costs are reduced, appraisal costs can also be reduced, and the total software quality costs decrease.

Important questions then arise concerning whether and how much to invest in specific software quality improvement initiatives. It is useful to approach these questions from a financial return on investment (ROI) perspective. We refer to this as the return on software quality.”

Evaluating the Cost of Software Quality  
Sandra A. Slaughter, Donald E. Harter, and Mayuram S. Krishnan  
COMMUNICATIONS OF THE ACM August 1998/Vol. 41, No. 8

With the concept that quality can be free, we may have “cooked the goose” as to how this can be applied in a repeatable and consistent manner. The concept of “free” must not be introduced as an overall cost since the mechanisms and training have to be paid for and that will be a significant fixed cost. The important point is that once the mechanisms and process are in place (The fixed costs are paid for) the “Variable costs” of labor become so small that for all practical purposes the implementation of quality is approaching the value of “free”.

So what are the requirements for achieving this goal of embedded quality? The following characteristics are derived from the previous discussions where we can safely say that the mechanisms must have:

- 1) Preventive quality mechanisms (defects, metrics and architecture)
- 2) Knowledge of syntax errors
- 3) Knowledge of conceptual errors



- 4) Visual representation of systems and relationships
- 5) Automated quality measurement/reporting/enforcement mechanisms
- 6) Automated Architectural enforcement mechanisms

The following section will map these requirements to the mechanisms that will satisfy these requirements.



## Quality Mechanisms

There is a set of mechanisms (products) provided by Static Analysis Tools that can provide the infrastructure that will make “real” Quality Management a possibility for software development teams. This set of products require the following framework to perform their jobs properly:

- Static Code Analyzer to Report Defect, Metric and Architecture Violations – InSpect
- Visual Representation of Interfaces for Entire System – Architecture Tools
- Desktop Static Code Analyzer to Prevent Violations – Desktop Analysis
- Tracking Defects, Metrics and Architecture through Time – Metrics Management

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

### **InSpect (Req. 2, 3, 5 & 6 reporting and enforcement)**

Build time tool that runs at every build or at an appointed build to provide a snapshot of the system metrics at a moment in time. The use of this build snapshot allows for the integration and execution of rules that will catch all code/security defects and any metric/architecture violations. These quality based rules or standards were agreed upon prior to the development effort. InSpect includes the most advanced static analyzer with the use of heuristics, interprocedural analysis and extensibility.

### **Architecture Tools (Req. 4, 6 Set-up)**


Architecture Tools is the architecture tool that allows architects and QA personnel to see the actual relationships between files. This allows architects and QA personnel to discover/investigate the quality of an interface between two components after a build is complete. It also allows the investigator to display the issues found by the InSpect tool in a graphical user interface. The architects can further use the tool to define an architecture that is enforced by each developer using Desktop Analysis and the QA team can monitor conformance to that requirement at every build using InSpect.

### **Desktop Analysis (Req. 1)**

The Desktop Analysis tool allows a developer level enforcement to ensure the developer does not violate the interface and defect controls for hard to find defect bugs and security defects. The tool can also track and warn the developer when they have passed metric thresholds that will put their code at risk down the road. The enforcement of architecture must happen at the developer’s desktop to gain the most immediate value. With the use of Architecture Tools the architectural intent can be translated to the developer such that they will not break the system level architecture interface laws that have been established by the team architects. This product is the preventative tool that will allow developers to inject quality at every step of a code change.

### **Metrics Management (Req. 5 reporting)**

Metrics Management is the tool that provides detailed metric data build over build. This tool will give the development team, QA team and management team the visibility and ability to gather information on many details including metric and architecture violations. Thus providing



that information to the development managers to back up their development decisions when the time crunches occur. The executive team, having access to the same data, can also provide incentives that have clear quality metric objectives and goals allowing effective code quality management and code risk management.

### Quality Process Changes

The use of these mechanisms need to be integrated into existing processes or their needs to be additional processes defined to ensure the tools are used properly. Those process changes enacted by development roles will provide the mechanism for success. There are many ways to use these products to keep laying golden eggs. The following are just suggested implementations by product.

#### InSpect

Managers – Build reports detail the progress of the development effort, which can be reviewed

Architects – Monitor the build reports to identify architecture/interface violations at a system level

**Quality Assurance – Monitor/review defects, metrics and arch. to make decisions on quality standards**

TeamLeads/Developers – Review defects and schedule for updating code based on the reports

#### Architecture Tools

Managers – Architect Reports are generated to monitor the progression of the architecture/interfaces

Architects – Discover, Understand, Manage and deploy architecture/interface rules and decisions

**Quality Assurance – Monitor and review architecture/interface to verify quality standards**

TeamLeads/Developers – Monitor using Architecture Tools and the architectural intent and interface decisions

#### Desktop Analysis

Managers – Monitors the development team to ensure the tools are performing

Architects – Performs periodic checks on the code using Desktop Analysis to ensure tools are performing

**Quality Assurance – Monitors the usage of the Desktop Analysis tool by monitoring the InSpect Reports**

TeamLeads/Developers – Enforcement tools used when making any changes to the code/interface

#### Metrics Management

Managers – Monitor churn rates and make development decisions based on trending and risk data

Architects – Monitor interface violation levels, complexity and risk points for indications of refactoring

**Quality Assurance – Monitor and review defects, churn rates, complexity and risk hot spots**

TeamLeads/Developers – Monitor the interface levels, metrics and trending to ensure compliance

Where these tools can be used by both the central system level management or distributed development management where QA management will have the ability to guide the quality efforts that have been established. The central system management and core product team and core architects can enforce and ensure the interfaces, processes and procedures coupled with the design decisions and standards that are deployed are used appropriately at the proper quality levels.

Now that the products and processes are better understood it is clear that an understanding of the value can be beneficial when making decisions for adopting a software quality management effort. The following section will apply hard numbers to just one of the areas of financial benefit, which is finding defects early in the process.

## The Value of Managing the Golden Egg

To ensure a complete evaluation of the benefit of using quality assumptions will be made that may differ from the target environment.

### Assumption 1: Development Costs

- Code Reviews - \$100 per hour per person – Average \$500 per 1000 Lines of Code (LOC)
- Test Defects - \$500 per defect if found prior to testing
- Field Defects - \$15,000 per defect if found prior to deployment into the field

### Assumption 2: Development size and change

- Project Size: 1,000,000 LOC
- Offsite Churn Rate: 20% of the code
- Effective Defect rate of code: 6 per 1000 LOC (KLOC)
- Testing Defects caught by Static Analysis Tools: 10% - 50% (Currently rules of thumb)
- Field Defects caught by Static Analysis Tools: 1% - 5% (Currently rules of thumb)

### Assumption 3: Use of metrics/architecture impacts included

The following would be the calculations to derive the benefit of these tools:

#### Code Review

- 1) First pass:  $1\text{MLOC} / 1000 = 1000$  hours of work \*  $\$500\text{KLOC}/\text{hour} = \$500\text{K}$
- 2) Churn Rates: 20% per year =  $200\text{KLOC}/1000 = 200$  hours \*  $\$500\text{KLOC}/\text{hour} = \$200\text{K}$ 
  - a. Effective IRR of 10% would provide net present value of \$2M
- 3) Total value is \$2M with churn plus one time run or  $\$500\text{K} = \$2.5$  M per MLOC

#### Finding Defects prior to Testing

- 1) Initial pass:  $1,000,000\text{ LOC} / 1000 = 1000 * 6$  defects = 6000
  - a. Per million lines of code there exists 6000 defects today
  - b. Static Analysis Tools may find 10% of 6000 =  $600 * \$500$  per defect = \$300K
  - c. Static Analysis Tools may find 50% of 6000 =  $3000 * \$500$  per defect = \$1.5M
- 2) Churn Rate of 20% assumed is  $200,000\text{ LOC} / 1000 = 200 * 6$  defects = 1200
  - a. Per million lines of code with churn of 20%, 1200 defects are added per year
  - b. Static Analysis Tools may find 10% of 1200 =  $120 * \$500$  per defect = \$60K
    - i. Effective IRR of 10% would provide net present value of \$600K
  - c. Static Analysis Tools may find 50% of 1200 =  $600 * \$500$  per defect = \$300K
    - i. Effective IRR of 10% would provide net present value of \$3M
- 3) At 10% the total value is \$600K with churn plus one time  $\$60\text{K} = \$660\text{K}$  per MLOC
- 4) At 50% the total value is \$3M with churn plus one time  $\$300\text{K} = \$3.3\text{M}$  per MLOC

#### Finding Defects prior to Field

- 1) Initial pass:  $1,000,000\text{ LOC} / 1000 = 1000 * 6$  defects = 6000
  - a. Per million lines of code there exists 6000 defects today
  - b. Static Analysis Tools may find 1% of 6000 =  $60 * \$1500$  per defect = \$900K
  - c. Static Analysis Tools may find 5% of 6000 =  $300 * \$1500$  per defect = \$4.5M
- 2) Churn Rate of 20% assumed is  $200,000\text{ LOC} / 1000 = 200 * 6$  defects = 1200
  - a. Per million lines of code with churn of 20%, 1200 defects are added per year
  - b. Static Analysis Tools may find 1% of 1200 =  $12 * \$1500$  per defect = \$180K
    - i. Effective IRR of 10% would provide net present value of \$1.8M
  - c. Static Analysis Tools may find 5% of 1200 =  $60 * \$1500$  per defect = \$900K
    - i. Effective IRR of 10% would provide net present value of \$9M
- 3) At 1% the total value is \$1.8M with churn plus one time  $\$900\text{K} = \$2.7\text{M}$  per MLOC
- 4) t 5% the total value is \$9M with churn plus one time  $\$4.5\text{M} = \$13.5\text{M}$  per MLOC