

Efficiency and Effectiveness Measures To Help Guide the Business of Software Testing

Abstract

This paper is based on actual Hewlett Packard work experience. The results of the “Process Metrics” team will be discussed. The “Process Metrics” team was part of a special 100 day effort to identify ways to improve the current Software Testing processes, infrastructure, and measurement. The specific focus of the “Process Metrics” team was to identify and define efficiency and effectiveness metrics for the Boise LaserJet Test Lab. The “Process Metrics” team consisted of five members who contributed approximately 10-20 hours per week for a three month period. This is a total of approximately 900 engineering hours (15 hours p/week * 5 engineers * 12 weeks). This paper will explain what metrics were chosen and what process was used to select the metrics. This paper will enable other Software Testing organizations to examine their current metrics solution and adjust it to better meet the needs of Software Testing.

Introduction

Until recent years, many Software Testing organizations have often been services funded by Software Development organizations. As a result, the measurements reported by Software Testing organizations have often been tailored to fit the needs of Software Development. Although it is important to measure the quality of the product under development, it is equally important to measure the effectiveness and efficiency of Software Testing itself as an activity – not a service.

Specifically, the measurements described in this paper first answers the question of whether Software Testing is "doing the right thing" (effectiveness). Once there is assurance and quantification of correct testing, metrics should be developed that determine whether or not Software Testing "does the thing right" (efficiency).

By measuring effectiveness and efficiency, a Software Testing organization can better communicate its own importance using factual information. This enables Software Testing organizations to break free from the misconception that Software Testing measurement should concentrate on issues important to the Software Development community.

1.0 Background - Why Metrics Specific To SW Testing Are Essential

Tom DeMarco, a consultant and metrics expert, has said “if you don’t measure, then you’re left with only one reason to believe you are still in control: hysterical optimism”.¹ The effort and dollars put into Software Testing today demand that professional testers rely on more than “hysterical optimism” to manage the business of Software Testing.

1.1 New Development Costs

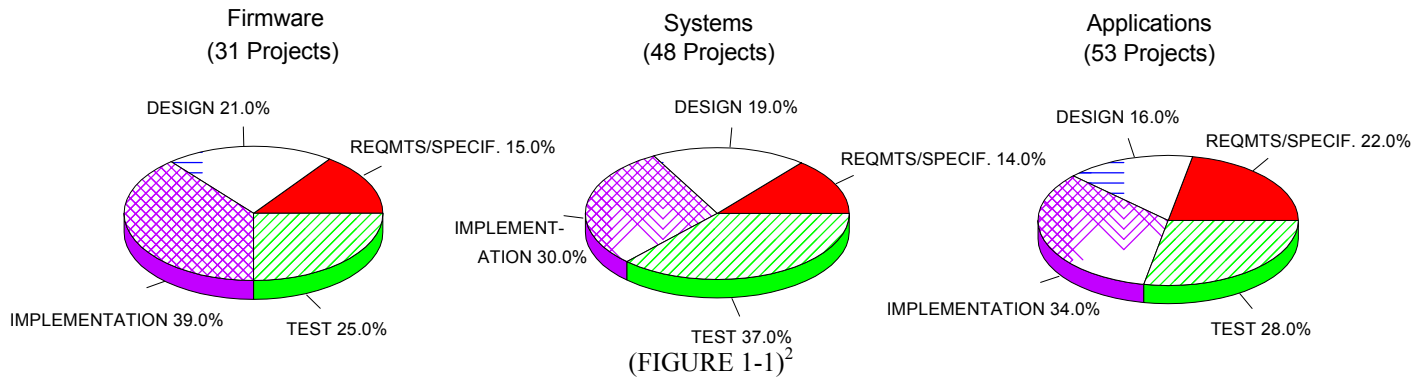
The data in Figure 1-1 represents the amount of time attributed in a total of 132 Hewlett Packard software projects involved in new development. “The data is further separated into three categories that are generally accepted in HP: firmware, systems, and applications software. Firmware is software that runs in ROM (Read Only Memory) or RAM (Random Access Memory) under control of a microprocessor. Systems software executes from the memory of one or more networked computers. Applications software operates on top of systems software in one or more components to solve specific user problems... the resulting development activity percentages are

- Requirements/Specifications – 18%

- Design – 19%
- Implementation – 34%, and
- Test – 29%”²

This data gives some indication of the amount of effort attributed to the Test phase when developing new software.

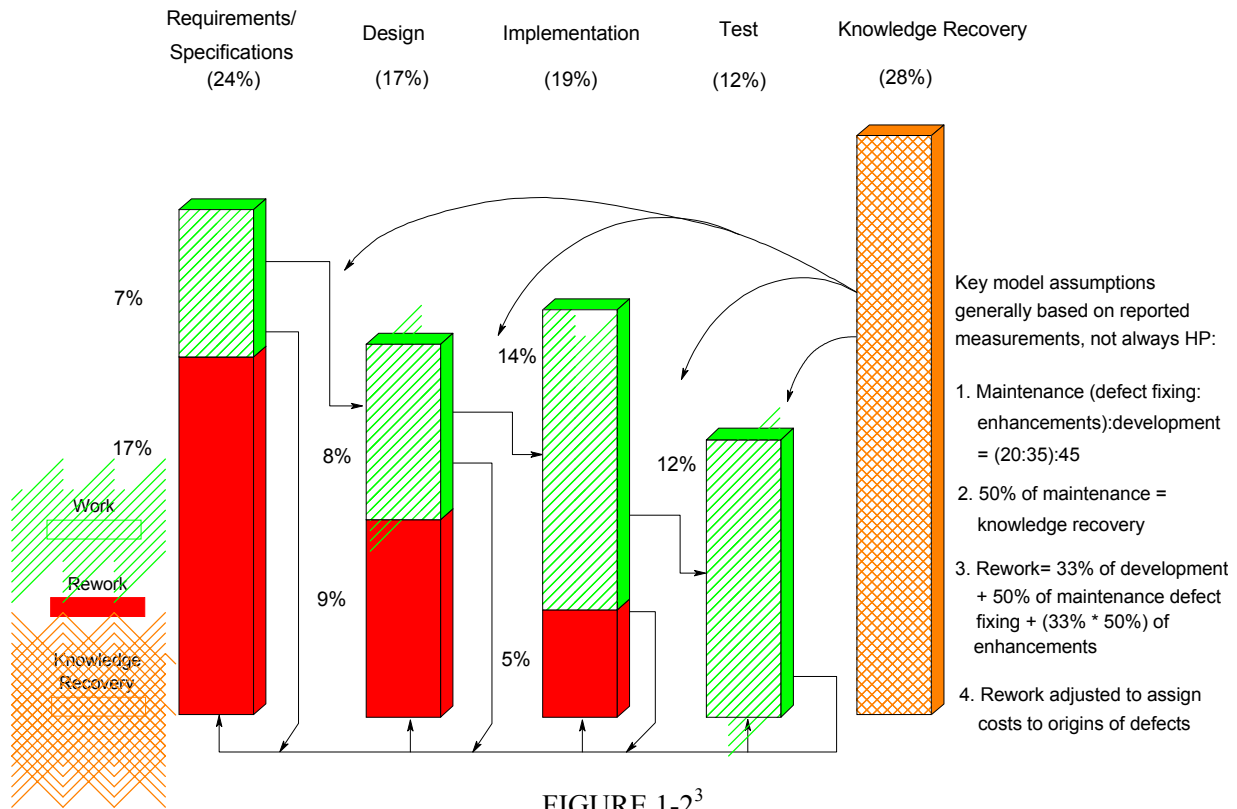
Percent Engineering Hours by Phase



1.2 Software Testing Costs Compared to Total Development Costs

Compared to the data in Figure 1-1, which represents new software development costs at Hewlett Packard, the Software Development/Maintenance Management Model described by Robert Grady in Figure 1-2 takes into account many different types of software projects. The model shows “Work”, “Rework”, and “Knowledge Recovery” as major components exhibited throughout the software lifecycle. The model is based on extensive data gathered for many years from Hewlett Packard and other industry software projects. In Figure 1-2, “Work” is the effort necessary to create the software. “Rework” is comprised of development, defect fixing, and enhancement implementation activities. “Knowledge Recovery” is described as the effort involved in learning and understanding the software system. One aspect of “Knowledge Recovery” is the amount of time needed to bring new people working on the software up to speed. Factoring in each aspect of this comprehensive model provides a complete picture of software that suggests the minimum amount of effort focused on testing to be 12%. This twelve percent, however, does not include the effort allocated to software testing “Rework” and “Knowledge Recovery”. If these factors are added to the twelve percent, software testing represents a significant portion of the overall cost of software development and maintenance.

A Management Model of Major Software Development/Maintenance Cost Components



As with any other business critical activity, incremental, sustained improvements in software testing products and process will lead significant overall benefits. Successfully measuring Software Testing can help with these improvement activities.

1.3 Maturity Model for Software Testing Measurement

One way to evaluate and incrementally improve a Software Testing metrics program is to apply a maturity model. Test Process improvement (TPI) is an example of a software testing maturity model consisting of 19 key areas, 4 levels, and checkpoints within each level for each area⁴. In the model, each key area is designated a particular level based on whether or not particular cumulative checkpoints are met. Only one of the key areas in the model will be discussed. This is the key area called “Metrics”. For the purposes of this paper, the levels within the “Metrics” key area will be described with terms comparable to other maturity models. The four levels that exist within the “Metrics” key area are:

I) Repeatable – At this first level, the emphasis is on product metrics. Test project input data such as “resources used (hours), performed activities (hours and lead time), and the scale/complexity of the system under test”, is collected. Project output data relating to test products and progress is also accumulated.

II) Defined – Emphasis at this level is on the testing process. In addition to the “Repeatable” checkpoints being met, the following metrics are described in the checkpoints; “defect location effectiveness (% defects found in test in relation to total, which tests should have found the

defect(s)), defect location efficiency (defects found per hour of test), level of test coverage(depth of testing), defects on testware (% of defects attributed to testing as compared to total), and perception of quality (through reviews and interviews with users, testers and other people involved).

III) Managed – The focus at this level is on system metrics. The model describes this level as “The statistics mentioned above (meaning at the Repeatable and Defined levels) are collected for new development projects and for maintenance on operational systems. The statistics are used for to evaluate the effectiveness and efficiency of the test process.” It appears that the only difference between this TPI level and the previous one is the application of the previous two levels at the system, as compared to the software, level.

IV) Optimized - The concentration at this level is on organization metrics. For this level, the TPI model states “Throughout the organisation comparable statistics are collected for the data mentioned before. The statistics are used to evaluate the effectiveness and efficiency of separate test processes, in order to optimise the generic test method and future test processes.”

The TPI model is useful in the right context. Namely, the types of metrics mentioned at the repeatable and defined level are areas of emphasis that must be considered. What the TPI model lacks is application to the needs of specific testing organizations. The “optimized” level mentions organizations but still lacks exactness. For example, at the defined level, “level of test coverage” is mentioned. What is not discussed is the types of test coverage that are appropriate. Is it branch coverage, call pair coverage, profiler coverage, requirements coverage, or use case coverage that should be measured? These types of specific questions, corresponding to the metrics mentioned at the repeatable and defined levels, are only answered by examining the particular measurement needs of an organization. The paper will now examine how metrics specific to an organization are determined and linked back to the TPI model.

2.0 Gathering Requirements for SW Testing Metrics

2.1 Start with GQM

The Goal Question Metric (GQM) process, created by Victor Basili and his colleagues at the University of Maryland, is a good place to begin targeting the specific measurement needs of an organization.⁵ The main emphasis of GQM is goal directed measurement. An organization usually starts with generic goals that must be refined. For example, “Reduce the number of failures found on a project”. This is certainly a goal, but is it well enough refined? One technique to further refine goals, making them specific enough that they are applicable to the direction of the organization, is the SMART technique⁶. This consists of five parts:

Specific – Is the goal specific? Even for developers and testers working on the project, a percentage and timeframe should quantify the words “reduce” and “failures”.

Measurable – Can the goal be gauged in comparison to other data? In this example, the answer appears to be “yes”. What is lacking is why this quantity is being measured? What decisions are being made? What conclusions can be drawn? It is important to consider questions like these when refining goals.

Attainable – Is there agreement that this goal is achievable? Has consensus from the rest of the team been obtained? Are resources allocated to work on the goal?

Relevant – Is the goal impractical or imprecise? Is the goal scaled in the proper perspective? Is the goal within the scope of what you are responsible for and expected to accomplish?

Time-limited – Does the goal have a specific start and end date? Is there time in the project schedule allocated toward collecting data and tracking progress toward the goal? In the example above, this is not the case.

In addition to “SMART”, every valid goal should represent a “stretch”. In other words, achievement of the goal is not something that will be accomplished without effort and focus. When the organization reaches the goal, there will be agreement that improvement has definitely occurred.

Continuing with the example, a refined goal statement is “Find an additional 20% of critical failures (severity level 3&4), as compared to the last project, from now up until one month prior to manufacturing release.” This refined goal meets the “SMART” criteria and represents a “stretch” for the organization.

The next step in the Goal-Question-Metric process is to begin asking questions. The idea is to generate questions about the goal that will lead to specific metrics. A few questions to consider are:

- Is this project similar enough to the previous project that this type of comparison makes sense?
- What are the causes of critical defects?
- What data about duration testing indicates that 20% more critical failures can be found using these techniques?
- In the last product, what was the percentage of “critical” failures found, for the corresponding time period, as compared to the total?
- How many critical defects are expected for the same period on the next project?
- What duration test suite is appropriate for this project?
- Does duration testing enable finding a higher percentage of critical defects than regular testing?

Once a list of valid questions are created, measurements are generated. When considering metrics, it is often helpful to list the raw data that must be collected. This raw data is sometimes referred to as “primitive metrics”. In this example, some important raw data is:

- Number of critical defects with a severity level of three and four.
- Time in duration testing.
- Total number of defects found in duration testing time period.
- Number of critical defects found on the last project for the corresponding time period.
- Number of total defects on last project for the corresponding time period.

Once the raw data is defined, more complex, or “computed” metrics are generated based on combinations of primitive metrics. Deriving measurements from raw data and translating that data into something useful to managers and/or developers is essential in tracking real progress towards a goal. Important computed metrics in this example are:

- $\frac{\text{Number of critical failures found in duration testing time period}}{\text{Total number of failures found in duration testing time period}}$
- $\frac{\text{Number of critical failures (severity 3&4) found in corresponding time period on previous project}}{\text{Total number of failures found in corresponding time period on previous project}}$

Once initial measurements are defined using the GQM paradigm, it is essential to verify that the metrics align with the departments(teams) that make up the organization.

2.2 Collect Input from Affected Teams

Many GQM efforts consist of following the described GQM method with intact work teams. On the Process Metrics team, however, we were asked to define metrics for an entire testing

organization. As a result, some members of the Process Metrics team engaged in GQM for the organization. At the same time, other members of the team worked on determining what current metrics are being used by teams and what metrics are needed in the future. The following matrix is an example of how this type of data was collected from the System Testing group.

Owner	Process / Activity Being Measured	Metric, Effectiveness/ Efficiency, In Place Now / Planned for Future	Customer(s), Goals, Decisions Made
System Test Lead	Install Testing	Number Defects Found, Effectiveness Measure, In Place Now	R&D / Marketing, Find defects with install testing, Adjust testing based defect finding results
System Test Manager	Overall System Testing Process	Number of field defects prevented, Effectiveness, Planned for Future	Warranty / Customer Center / R&D, Marketing, Prevent more defects from getting into the hands of customers, increase the intensity of System Testing

After data was collected from each test department, the results were examined to make sure the metrics outlined by specific departments aligned with GQM results.

2.3 Six Prioritized Measurements

GQM results, completed surveys from testing teams within the organization, and prioritized voting led to the following measurements:

Metric	Importance	Timeframe	Difficulty	TPI Mapping
<u>Tests That Find Defects</u>	High	Short Term	Low/Medium	II
<u>Defects Found Per Week</u>	High	Short Term	Low	II
<u>Testing Coverage</u>	High	Medium/Long Term	Medium/High	II
<u>Testing Benefit vs. Cost</u>	High	Short/Medium Term	Medium/High	I
<u>Automation Count</u>	Medium/High	Medium Term	Medium	II
<u>Tests Configured, Executed per Unit of Time</u>	Low/Medium	Medium Term	Medium	I

It should be noted that the TPI mapping of the metric is not the primary consideration as to whether the measurement is of high importance. Rather, the TPI mapping helps determine the level of maturity of the metrics program. When assigning importance, there are other organizational considerations in addition to TPI. Each metric's importance is based on three attributes; the timeframe in which the metric can be implemented, the relative difficulty of establishing the measurement, and its TPI mapping. Based on each of these attributes, the Process Metrics team decided on the importance of each metric. The recommended focus is on High and Medium importance metrics first. At this point, six metrics have been identified. What is not known are the details of each metric. The necessary detail will be described in the design of each metric.

3.0 Designing The Metrics

The GQM process, gathering input from other teams, and prioritizing the metrics is roughly equivalent to metrics requirements gathering, with a hint of high level design when considering primitive vs. computed metrics. The next step in the lifecycle is detailed design. For each metric, a template was completed. For the purposes of this paper, the important parts of the template are represented in the following matrices.

3.1 Measurement Design Matrices

3.1.1 Measurement Basis (GQM)

Goal	Questions	Metric
By end of FY 1999, determine the ratio of “clean” to “dirty” tests. Reduce execution frequency of “dirty” tests that are no longer finding critical defects.	<ul style="list-style-type: none"> - What is the definition of a “dirty” test? - If a test corresponds to one or more requirements, is it “clean”? - What is the definition of each level of criticality? 	<u>Tests That Find Defects</u>
By the end of FY 1999, illustrate the importance and value of Software Test Section activities in the eyes of customers.	<ul style="list-style-type: none"> - What types of defects is the Software Test Section finding? - How many and what types of defects are being found by organizations outside the Software Test Section? - What is valuable to customers? 	<u>Defects Found Per Week</u>
By the end of FY 2000, improve the thoroughness of testing at testing levels that are the responsibility of the Software Test Section.	<ul style="list-style-type: none"> - What types of coverage are applicable at different test levels? - How is coverage data gathered? - What coverage data exists, what does not? 	<u>Testing Coverage</u>
By end of FY 1999, improve the ability to communicate the value of testing to customers by reporting specific benefits vs. the costs of testing.	<ul style="list-style-type: none"> - What are the key components of cost that should be measured? - How does the customer measure the value of testing (requirements tested, defects found, etc.)? - How much, on average, is spent for testing on a particular project? 	<u>Testing Benefit vs. Cost</u>
By end of FY 2000, improve the ability to repeat tests thereby increasing the level of accuracy in predicting and designing project testing schedules.	<ul style="list-style-type: none"> - What criteria must be met to automate a test? - What are the levels of automation? 	<u>Automation Count</u>
By end of FY 1999, improve schedule planning by gaining a better understanding of how much effort it takes to accomplish testing tasks.	<ul style="list-style-type: none"> - What are appropriate units of time for this measurement? - What are the current test phase activities? What are the ideal test phase activities? 	<u>Tests Configured, Executed per Unit of Time</u>

3.1.2 Definition

Metric Description	Calculation Formula	What it is	What it's not	Success & Failure Indicators
<u>Tests That Find Defects</u> A list of tests that find defects and a list of tests that do not.	Count the number of tests in the defect/test management system that have found defects at three levels of criticality (High, Medium, and Low).	A way to prioritize tests based on historical performance.	An absolute measure of a test's effectiveness.	Evaluate “Dirty” tests that aren't finding defects as potential candidates for reduced execution frequency. “Clean” tests should not be judged solely on this metric, other indicators are necessary.
<u>Defects Found Per Week</u> The number of product defects found per week by members of the LSG Software Test Section.	Count the number of defects found each week by Software Test Section Tests per criticality categories (ex. High, medium, and low). A week is defined as a calendar week wherein test execution has occurred.	One indicator of test efficiency.	This metric should NOT be used by itself to determine the efficiency of testing efforts.	Once historical data is available, similar projects “may” be compared.

Metric Description	Calculation Formula	What it is	What it's not	Success & Failure Indicators
Testing Coverage A measure of the thoroughness (depth) of testing.	1st Phase Only <u>High Level Component Testing</u> # Rqmts. Tested / Total # Rqmts. <u>System Testing</u> # Use Cases Tested / Total # Use Cases Top Tier Configs Tested / Total # Top Tier Configs	Several measures of the completeness of testing.	An indication of whether or not defects are being found.	Based on definitions in testing model.
Testing Benefit vs. Cost The number of requirements tested or defects found, by criticality, per dollar spent on testing resources plus effort.	Requirements Tested or Defects Found per month / Dollar cost per month of: 1. Resources – Non labor expenses, contract labor, occupancy costs, capital depreciation, categorized or apportioned to lowest level project:activity PLUS 2. Effort – Labor for each test lab individual categorized / apportioned to lowest. Level project:activity	A measurement of the value of testing as compared to the resources and effort spent.	Intended to be precise to the \$1.00. Especially in the area of effort, costs will be estimated based in information entered by the worker.	There is no inherent numerical criteria for cost.
Automation Count The number of testing activities that meet particular automation criteria.	Count of tests in each of the following categories, using supplied definitions, standards, and “engineering judgement”: 0: not automated 1: partially automated 2: substantially or fully automated NA: Not (applicable/unable to automate)	A measure of opportunity in the test lab to be realized by automation of our test activities	A standalone measure; it is probably best combined with the Cost measure to evaluate opportunities.	All activities classified either NA or Automated.
Tests Configured, Executed per Unit of Time The number of tests configured, executed, per unit of time (month, week, hour).	Number of tests configured, executed / Unit of Time (month, week, hour)	An indicator of testing efficiency.	A way to measure individual performance.	This is more of an indicator to let people know how testing resource is spent. It is difficult to say that you are successful or not based on activities accomplished per hour.

3.1.3 Assumptions & Dependencies

Metric	Assumptions	Dependencies
Tests That Find Defects	<ol style="list-style-type: none"> The designation of a test as “clean” or “dirty” is a new concept in this organization. There is currently no distinction in the defect management system between a “Failure” and a “Fault”. 	Internal - Common criticality categories. - Recording of necessary defect/test data. - Way to link defects to tests. - Common process for defect management. - Getting agreement that certain tests should be run less frequently. External - “Common” defect management.
Defects Found Per Week	<ol style="list-style-type: none"> The quality of software under test may affect this metric more than the rigor put into testing. 	Internal - Common criticality categories - Way to link defects to Test Lab Tests. External - Common process for defect management.
Testing Coverage	<ol style="list-style-type: none"> All the necessary tools may not be in place to measure coverage. Additional tools may need to be investigated and purchased. Measuring coverage requires significant effort and resource. 	Internal - Availability of resources. - Generation of accurate data. External - Many types of coverage measurements require in-depth knowledge of the product source code and integral partnership with R&D organization developers.
Testing Benefit vs. Cost	<ol style="list-style-type: none"> There is not one uniform method for gathering effort cost information among different departments. 	Internal 1. This depends on practices, presumably different right now, in use by departments. 2. Will require “multiplier” or other method (see proposal) to convert effort to \$.

Metric	Assumptions	Dependencies
		3. To capture all costs, some method of defining and apportioning "overhead" will be needed. 4. There are currently few methods to differentiate between defects found in testing and defects found in development or some other way. 5. There is currently no linkage between the Requirements Management and Test Management systems. <u>External</u>
<u>Automation count</u>	1. Interviewing and engineering judgement is necessary in determining the level of automation for tests. 2. Each department may have an overall different level of automation.	<u>Internal</u> Depends on owners of automation projects to report their results. <u>External</u>
<u>Tests Configured, Executed per Unit of Time</u>	1. Hours are being tracked per activity. This assumes a lot.	<u>Internal</u> Process to provide effort data. <u>External</u>

3.1.4 Usage

Metric	Decisions Made / Benefits	Costs
<u>Tests That Find Defects</u>	<ul style="list-style-type: none"> - Determine which tests are finding defects and which are not? - Way to prioritize tests. - Determine where testers are focusing their efforts. - Helps identify tests that should be blocked from selection/execution until the defect(s) they found have been resolved. These are good candidates for automation. 	<ul style="list-style-type: none"> - Team to define a common defect management process. This includes a way to link defects to tests and common criticality categories. - Effort to enhance defect and test tools to record new data. - Effort to implement reporting/measurement tools.
<u>Defects Found Per Week</u>	<ul style="list-style-type: none"> - Level of success at finding defects. - Adjust testing based on defect find rates. - If few defects (or non-critical defects) are being found, may substantiate the need for new tests. - May be used in conjunction with "Coverage" metrics to predict when a project is ready for release, or testing effort is complete. - Customer Center data could be used to learn what defects are getting past the Test Lab. This would be a better measure for "efficiency" as opposed to the number of defects found. 	<ul style="list-style-type: none"> - Team to define a common defect management process. This includes a way to link defects to an organization (i.e. Test Lab) and common criticality categories. - Effort to enhance defect (or other) tools to record organization name. It may be that a maintained list of Test Lab personnel is all that is needed. - Effort to implement reporting tools.
<u>Testing Coverage</u>	<ul style="list-style-type: none"> - Determine whether our customer usage profile fits what the customer actually executes. - Determine what interfaces are and are not being tested. - Identify bottlenecks and, thereby, potential opportunities for system performance tuning. - Based on industry standards, is our coverage sufficient? 	<ul style="list-style-type: none"> - Tool Investigation - Tool Purchase - Partnership costs with R&D and Technical Marketing. - Tool Implementation, Training, and ongoing Support (i.e. dedicated resource).
<u>Testing Benefit vs. Cost</u>	<ul style="list-style-type: none"> - Outsource or in house testing. - Can we complete this project within budget? - Are we spending too much for value added? Is our testing efficient? - What is the Cost/benefit of automating? - Where are our major costs; our major opportunities for cost reduction? - Basis for decisions, proposals, and estimates is more visible and easier to communicate. - What is it costing us to find defects / test requirements? - How can we find more critical defects / test more requirements for less money? 	Monthly Startup to Collect Cost information: <ul style="list-style-type: none"> - Agreement on tool/practice for each department to collect costs (range of 5-20 hrs /dept.; suppose 60 hrs total.) - Method to collect data from each department (20-40 hrs) - Initial report (10 hrs) total about 100 hrs.
<u>Automation Count</u>	<ul style="list-style-type: none"> - Identification of opportunities for resource utilization with automation. - Identification of opportunities for lessening schedule impacts. - Improving repeatability of tests. - Estimation of schedule and resources for new projects. - Increased efficiency in the Test Lab. - Reduced schedule impacts and increased responsiveness - Increased credibility and precision of reported results. - Better ability to plan and to make sourcing decisions. 	Implementation: (assume a phased implementation as increasing numbers of activities are identified. Also assume that area managers can categorize broad testing areas together and that automation is the exception right now.) 8 hours , most of which will be used to define criteria for the automation categories; an itemized database solution will be more costly. Ongoing costs: As efforts are applied to automate

Metric	Decisions Made / Benefits	Costs
	- More effective use of resources.	activities, these will be reclassified. Total effort of reporting and tracking this result should be less than 2 hours / month (an itemized solution will be more costly).
<u>Tests Configured, Executed per Unit of Time</u>	- Per activity, the amount of time involved in testing a project. - What is the estimated vs. actual time spent on these testing tasks? - Information for developing a contingency plan to handle changes to testing schedules, product and process. - Provide an actual account used for estimates on future projects. - Quick & efficient response to changes in testing schedules, product, and process.	The effort needed for test workers to enter the time on these testing tasks. The cost of implementing a tool to allow such data to be entered.

3.1.5 Data To Collect (Input)

Metric	Internal Data Needed	External Data Needed						
<u>Tests That Find Defects</u>	- <u>Test Name</u> – Name of the test. - <u>Test ID</u> – Unique identifier. - <u>Execution Date</u> – Date test was executed. - <u>Defect ID</u> – Unique identifier. - <u>Tool Name</u> – Name of defect tracking tool (ex. DMS, CRMS, ClearQuest) - <u>Criticality</u> – Defect criticality (ex. High, medium, low). - <u>Project Name</u> – Name of the project being tested.	N/A						
<u>Defects Found Per Week</u>	Defect id, criticality, date found, project name.	Submitter's organization.						
<u>Testing Coverage</u>	- Tests Designed to meet Requirements / Use Cases / Top Tier Configs - Total Number of Requirements / Use Cases / Top Tier Configs	Different types of coverage will require various types of data from external partners.						
<u>Testing Benefit vs. Cost</u>	- Identify and prioritize the activities/outputs/deliverables for each department, each month, for each project. - Identify monthly expenses and capital depreciation, and "overhead". - Periodically update the loaded cost of a month of test lab personnel effort (and for categories of effort, such as engineering Vs management, if these are used.) - Link between requirement and the test that verified that requirement. - Defects found, by criticality.	Cost, defects found, requirements tested comparisons from third party testing vendors.						
<u>Automation Count</u>	1. Identification of activities needing automation classification 2. Reclassifications, as they occur. Examples: Summary form: Number of TCP Network Test Executables: 70, Number classified NA: 20, Number Classified Automated: 5, Number Classified Partial: 15, Number Classified Not Automated: 30. Itemized form: Assumes that each test executable has an entry in a database of Test Lab activities. An attribute of each test, possibly in a table denoted "Automated" would be the automated status as coded in 1.1. A typical record in this table might look like (assuming that "cost" has been recorded in the same table): <table border="1"> <thead> <tr> <th>Test Activity</th> <th>Automated Status</th> <th>Activity Cost</th> </tr> </thead> <tbody> <tr> <td>DLC: Banner01</td> <td>NA</td> <td>\$1000</td> </tr> </tbody> </table>	Test Activity	Automated Status	Activity Cost	DLC: Banner01	NA	\$1000	May want to compare with R&D development test systems already automated.
Test Activity	Automated Status	Activity Cost						
DLC: Banner01	NA	\$1000						
<u>Tests Configured, Executed per Unit of Time</u>	Activity ID Activity Description # Activity Instances Product ID Project ID Hours Estimated Hours Spent (entered at regular intervals) Example - Localization Tests Executed per hour - Query "Execute Localization Tests" Test Activity - Pull "Hours Spent" field - Query "Test Instance" Data Objects and count Localization Tests Executed - Divide Number Counted by "Hours Spent"]	This same type of data can be requested from 3 rd party test vendors.						

3.1.6 Data To Display (Output)

A picture is worth a thousand words. Another important part of designing a metric is roughing out what the initial report, graph, or chart will look like. It is not possible to provide examples of this for every measurement defined by the Process Metrics team, but a few example reports and graphs are represented below:

Metric = Tests That Find Defects

1) Tests That Have Found Defects

Test Name	Test ID	Number of Defects Found			
		High	Medium	Low	Total

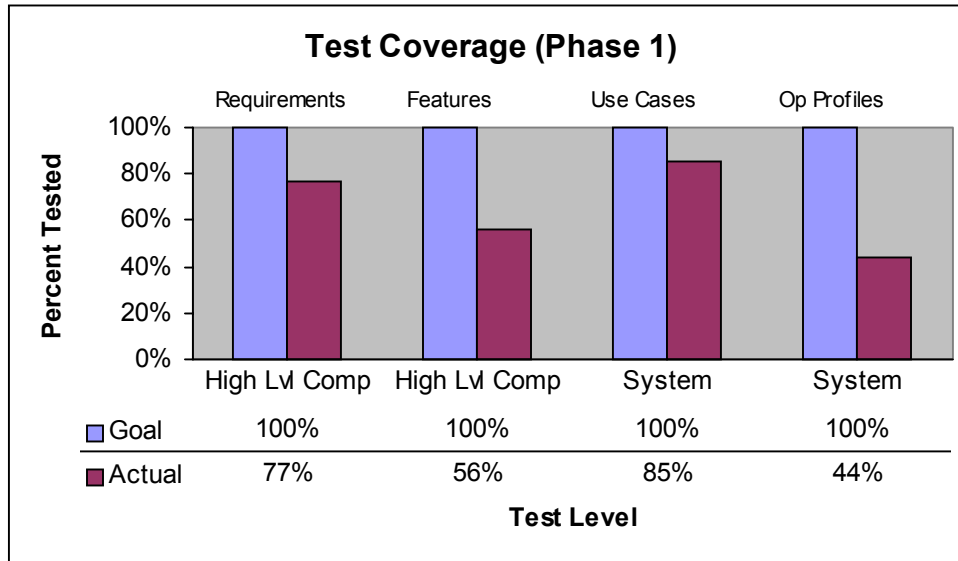
2) Tests That Have Found Defects - More Detail

Test Name	Test ID	Date Executed	Defect ID (and Tool Name)	Defect Criticality	Project Name

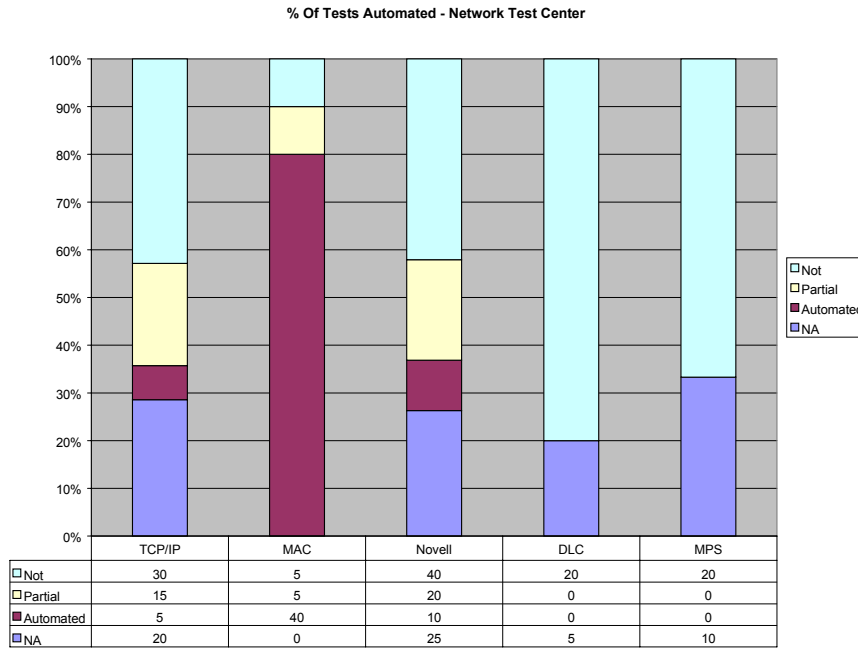
3) Tests That Have Not Found Defects

Test Name	Test ID

Metric = Testing Coverage



Metric = Automation Count



4.0 Implementing a Successful SW Testing Measurement Program

When managers and co-workers notice the rigor and time spent designing software testing metrics, they are much more likely to support a measurement implementation plan. Successful measurement programs have several characteristics such as an attractive value proposition, consistent management sponsorship, adequate dedicated resources, an automated way to report the metrics, and a way, with the metrics, to track progress towards goals set by the organization.

4.1 Value Proposition & Management Sponsorship

It is essential that managers and individual contributors understand the importance of a measurement effort to their bottom line. One powerful message that is consistent across most measurement programs is the ability to refocus resources. With a successful measurement program, engineers previously responsible for generating metrics for their project can focus on software product testing instead of generating metrics. This focus of resources away from metrics toward activity directly benefiting customers is an influential message in the eyes of sponsors. Another way metrics can benefit the bottom line is in the area of knowledge recovery through program understanding. When there are metrics available on a project, it is much easier to learn what is needed about a project. Metrics give new resources the ability to get up to speed and working on a project faster than if few metrics are available. When test and development engineers work together to create a quality product, substantial rework can be avoided. Rework can be substantially reduced when metrics provide an objective medium of communication between the test and development community. This enhanced communication is another benefit of a metrics program that should be communicated to sponsors.

4.2 Resources

Just like any other project, a measurement program is destined to fail if resources are not dedicated to it. Project planning techniques such as task breakdown and other resource allocation techniques should be used to plan the number and types of resources needed to maintain a successful measurement effort.

4.3 Automated Tool

Metrics gathering and reporting can be a time consuming effort. Many people focused on measurement spend days out of their work week creating and distributing metrics. There is often a simpler way to provide metrics to important customers. There are automated measurement tools available on the market. A decision should be made whether to purchase or create a tool specific to the needs of the organization. One important thing to remember, whether a tool is made or bought, is ensuring an open architecture. This means a common data standard must be in place. This standard can receive information from a variety of sources, all of which can be processed by the tool. Generating and displaying metrics automatically, by means of a tool, on a regular basis, is an effective way to gain a customer base for your measurement effort.

4.3 Track Progress Toward Goals

It is essential to constantly revisit goals and illustrate, in front of sponsors, how the measurements are used to track progress toward them. Emphasize what business decisions are made based on the metrics. Sponsors will soon forget the importance of measurement unless metric champions are reminding them on a regular basis.

5.0 Conclusion

The Process Metrics team and what is contained in this paper identifies all the tools necessary to specify, design, and implement a successful Software Testing Measurement program. An argument outlining the need for Software Testing Metrics is described, a useful Software Testing Measurement maturity model has been introduced, a process for generating and designing effectiveness and efficiency metrics is outlined, and experienced based suggestions for instituting a successful measurement program have been discussed. By using the tools and processes described in this paper, a Software Testing organization can better communicate the value of Software Testing and the importance of testing activities in the Software Development process.

6.0 Acknowledgements

I would like to thank the members of the Process Metrics team; Troy Brunson, Tom Menten, Karen Bellegante, and Jerry Smart. This paper would not have been possible without your efforts.

Thanks go to my manager Mary LePori. It's great to have a manager that believes people do a better job when they are given the opportunity to pursue their passion.

I would like to thank my colleagues Susan Davis and Mike Dunlap for providing valuable comments and suggestions in relation to the contents of this paper.

I would also like to thank Bob Grady, author of three books and numerous articles, for introducing me to Software Measurement, Failure and Root Cause Analysis. His comments on my papers, his insights and suggestions, have proven accurate and important.

7.0 References

1. DeMarco, Tom, Controlling Software Projects, New York: Yourdon Press, 1982, p. 6.
2. Grady, Robert B., "Successful Software Process Improvement". Prentice Hall, Inc., (1997), pp. 60-61.
3. Grady, Robert B., "Successful Software Process Improvement". Prentice Hall, Inc., (1997), pp. 62-67.
4. Pol, Martin. Koomen, Tim. "Test Process Improvement". Kluwer BedrijfsInformatie, (1998). Software Quality Week, San Francisco and Europe, 1998.
5. Basili, Victor, and D. M. Weiss, "Tailoring the Software Process to Project Goals and Environments," IEEE Proceedings of the Ninth International Conference on Software Engineering, Monterey, CA, (Apr. 1987), pp. 345-357.
6. Hewlett Packard Company. "Guide to Performance and Development Planning," Americas Geographic Operations Education Division, May, 1996.