An Introduction to Metrics used during Software Development Life Cycle

Define the Metric Objectives

"You can't control what you can't measure." This is a quote from Tom De- Marco's book, Controlling Software Projects, in which he describes how to organize and control a software project so it is measurable in the context of time and cost projections.

Control is the extent to which a manager can ensure minimum surprises. Deviations from the plan should be signaled as early as possible in order to react.

Another quote from DeMarco's book***, "The only unforgivable failure is the failure to learn from past failure,"*** stresses the importance of estimating and measurement.

Measurement is a recording of past effects to quantitatively predict future effects.

How do we define the Metrics?

Software testing as a test development project has deliverables such as test plans, test design, test development and test execution. The objective of this task is to apply the principles of metrics to control the testing process.

A metric is a measurable indication of some quantitative aspect of a system and has the following characteristics:

a) Measurable: A metric point must be measurable for it to be a metric, by definition. If the phenomenon can't be measured, there is no way to apply management methods to control it.

b) Independent: Metrics need to be independent of the human influence. There should be no way of changing the measurement other than changing the phenomenon that produced the metric.

c) Accountable: Any analytical interpretation of the raw metric data rests on the data itself, and it is, therefore, necessary to save the raw data and the methodical audit trail of the analytical process.

d) Precise: Precision is a function of accuracy. The key to precision is, therefore, that a metric is explicitly documented as part of the data collection process. If a metric varies, it can be measured as a range or tolerance.

A metric can be a "result," or a "predictor." A result metric measures a completed event or process. Examples include actual total elapsed time to process a business transaction or total test costs of a project. A predictor metric is an early warning metric that has a strong correlation to some later result. An example is the predicted response-time through statistical regression analysis when more terminals are added to a system when that many terminals have not yet been measured. A result or predictor metric can also be a derived metric. A derived metric is one that is derived from a calculation or graphical technique involving one or more metrics. The motivation for collecting test metrics is to make the testing process more effective. This is achieved by carefully analyzing the metric data and taking the appropriate action to correct problems.

The starting point is to define the metric objectives of interest. Some examples include:

a) Defect analysis: Every defect must be analyzed to answer such questions as the root causes, how detected, when detected, who detected, etc.

b) Test effectiveness: How well is testing doing, e.g., return on investment?

c) Development effectiveness: How well is development fixing defects?

d) Test automation: How much effort is expended on test automation?

e) Test cost: What are the resources and time spent on testing?

f) Test status: Another important metric is status tracking, or where are we in the testing process?

g) User involvement: How much is the user involved in testing?

How do we define the Metric Points?

Following are some metric points associated with the general metrics and the corresponding actions to improve the testing process. The source or the method to compute the metric point is also described.

Sr.	Metric	Metric Point	How to Compute
1)	**Defect analysis**	Distribution of defect causes	Histogram, Pareto Analysis. (**Histogram** – is a graphical representation of measured values organized according to the frequency of occurrence used to pinpoint hot spots **Pareto Analysis** – involves Analysis of defect patterns to identify causes and sources)
		Number of defects by cause over time	Multi-line graph
		Number of defects by how found over time	Multi-line graph
		Distribution of defects by module	Histogram, Pareto Analysis
		Distribution of defects by priority (critical, high, medium, low)	Histogram
		Distribution of defects by functional area	Histogram
		Distribution of defects by environment (platform)	Histogram, Pareto Analysis
		Distribution of defects by type (architecture, connectivity, consistency, database integrity, documentation, GUI, installation, memory, performance, security, standards and conventions, stress, usability, bad fixes)	Histogram, Pareto Analysis
		Distribution of defects by who detected (external customer, internal customer, development, QA, other)	Histogram, Pareto Analysis
		Distribution by how detected (technical review, walkthroughs, JAD, prototyping, inspection, test execution)	Histogram, Pareto Analysis
		Distribution of defects by severity (high, medium, low defects)	Histogram
2)	**Development effectiveness**	Average time for development to repair defect	Total repair time ÷ Number of repaired defects

Sr.	Metric	Metric Point	How to Compute
3)	**Test automation**	Percent of manual vs. automated testing	Cost of manual test effort ÷ Total test cost
4)	**Test cost**	Distribution of cost by cause	By Histogram, Pareto Analysis
		Distribution of cost by application	By Histogram, Pareto Analysis
		Percent of costs for testing	Test testing cost ÷ Total system cost
		Total costs of testing over time	By Line graph
		Average cost of locating a defect	Total cost of testing ÷ Number of defects detected
		Anticipated costs of testing vs. actual cost	By comparison
		Average cost of locating a requirements defect with requirements reviews	Requirements review costs ÷ Number of defects uncovered during requirement reviews
		Average cost of locating a design defect with design reviews	Design review costs ÷ Number of defects uncovered during design reviews
		Average cost of locating a code defect with reviews	Code review costs ÷ Number of defects uncovered during code reviews
		Average cost of locating a defect with test execution	Test execution costs ÷ Number of defects uncovered during test execution
		Number of testing resources over time	By Line plot
5)	**Test effectiveness**	Percentage of defects discovered during maintenance	Number of defects discovered during maintenance ÷ Total number of defects uncovered
		Percent of defects uncovered due to testing	Number of detected errors through testing ÷ Total system defects
		Average effectiveness of a test	Number of tests ÷ Total system defects
		Value returned while reviewing requirements	Number of defects uncovered during requirements review ÷ Requirements test costs
		Value returned while reviewing design	Number of defects uncovered during design review ÷ Design test costs

Sr.	Metric	Metric Point	How to Compute
5)	**Test Effectiveness**	Value returned while reviewing programs	Number of defects uncovered during program review ÷ Program test costs
		Value returned during test execution	Number of defects uncovered during testing ÷ Test costs
		Effect of testing changes	Number of tested changes ÷ Problems attributable to the changes
		People's assessment of effectiveness of testing	By subjective scaling (1–10)
		Average time for QA to verify fix	Total QA verification time ÷ Total number of defects to verify
		Number of defects over time	By line graph
		Cumulative number of defects over time	By line graph
		Number of application defects over time	By Multi-line graph
6)	**Test Extent**	Percent of statements executed	Number of statements executed ÷ Total statements
		Percent of logical paths executed	Number of logical paths ÷ Total number of paths
		Percent of acceptance criteria tested	Acceptance criteria tested ÷ Total acceptance criteria
		Number of requirements tested over time	BY line plot
		Number of statements executed over time	BY line plot
		Number of data elements exercised over time	BY line plot
		Number of decision statements executed over time	BY line plot
		Number of tests ready to run over time	BY line plot
		Number of tests runs over time	BY line plot
		Number of tests run without defects uncovered	BY line plot
		Number of defects corrected over time	BY line plot
7)	**User involvement**	Percentage of user testing	User testing time ÷ Total test time

How do we Define Test Metrics used in Reporting?

The most common Test Report is a simple matrix, which indicates the test cases, the test objectives, and the results of testing at any point in time. The following six tasks define how a test team can define metrics to be used in test reporting. An important part of these tasks is to assure that the data needed (i.e., measures) to create the test metrics is available.

1) Establish a test metrics team: The measurement team should include individuals who:

a) Have a working knowledge of quality and productivity measures
b) Are knowledgeable in the implementation of statistical process control tools
c) Have a working understanding of benchmarking techniques
d) Know the organization's goals and objectives
e) Are respected by their peers and management

The measurement team may consist of two or more individuals, relative to the size of the organization. Representatives should come from management and development and maintenance projects. For an average-size organization, the measurement team should be between three and five members.

2) Inventory existing IT measures: The inventory of existing measures should be performed in accordance with a plan. Should problems arise during the inventory, the plan and the inventory process should be modified accordingly. The formal inventory is a systematic and independent review of all-existing measures and metrics captured and maintained. All identified data must be validated to determine if they are valid and reliable. The inventory process should start with an introductory meeting of the participants. The objective of this meeting is to review the inventory plan with management and representatives of the projects that are to be inventoried.

A sample agenda for the introductory meeting is:

a) Introduce all members.
b) Review scope and objectives of the inventory process
c) Summarize the inventory processes to be used
d) Establish communication channels to use
e) Confirm the inventory schedule with major target dates

The inventory involves these activities:

a) Review all measures currently being captured and recorded: Measures should include, but not be limited to, functionality, schedule, budgets, and quality.

b) Document all findings: Measures should be defined; samples captured, and related software and methods of capture documented. Data file names and media location should be recorded. It is critical that this be as complete as possible in order to determine the consistency of activities among different projects.

c) Conduct interviews: These interviews should determine what and how measurement data is captured and processed. Through observation, the validity of the data can be determined.

3) Develop a consistent set of metrics: To implement a common set of test metrics for reporting that enables senior management to quickly access the status of each project, it is critical to develop a list of consistent measures spanning all project lines. Initially, this can be challenging, but with cooperation and some negotiating, a reasonable list of measures can be drawn up. Organizations with mature processes will have an easier time completing this step, as well as those with automated tools that collect data.

4) Define desired test metrics: The objective of this task is to use the information collected in above tasks to define the metrics for the test reporting process.

Major criteria of this task includes:

a) Description of desired output reports
b) Description of common measures

c) Source of common measures and associated software tools for capture
d) Definition of data repositories (centralized and/or segregated)

5) Develop and implement the process for collecting measurement data: The objective of this step is to document the process used to collect the measurement data.

The implementation will involve the following activities:

a) Document the workflow of the data capture and reporting process
b) Procure software tool(s) to capture, analyze, and report the data, if such tools are not currently available
c) Develop and test system and user documentation
d) Beta-test the process using a small to medium-size project
e) Resolve all management and project problems
f) Conduct training sessions for management and project personnel on how to use the process and interrelate the reports
g) Roll out the test status process

6) Monitor the process: Monitoring the test reporting process is very important because the metrics reported must be understood and used. It is essential to monitor the outputs of the system to ensure usage. The more successful the test reporting process, the better the chance that management will want to use it and perhaps expand the reporting criteria.

How do we Define Effective Test Metrics?

A metric is a mathematical number that shows a relationship between two variables. Software metrics are measures used to quantify status or results. This includes items that are directly measurable, such as lines of code, as well as items that are calculated from measurements, such as earned value. Metrics specific to testing include data regarding testing, defect tracking, and software performance.

The following definitions of metrics are available:

a) Metric: A metric is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.

b) Process Metric: A process metric is a metric used to measure characteristics of the methods, techniques, and tools employed in developing, implementing, and maintaining the software system.

c) Product Metric: A product metric is a metric used to measure the characteristics of the documentation and code.

d) Software Quality Metric: A software quality metric is a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.

Testers are typically responsible for reporting their test status at regular intervals.

The following measurements generated during testing are applicable:

a) Total number of tests
b) Number of tests executed to date
c) Number of tests executed successfully to date

Data concerning software defects include:

a) Total number of defects corrected in each activity
b) Total number of defects detected in each activity
c) Average duration between defect detection and defect correction
d) Average effort to correct a defect
e) Total number of defects remaining at delivery

Some of the basic measurement concepts are described below to help testers use quantitative data effectively.

How do we compare Objective versus Subjective Measures?

Measurement can be either objective or subjective. An objective measure is a measure that can be obtained by counting. For example, objective data is hard data, such as defects, hours worked, and completed deliverables. Subjective data normally has to be calculated. It is a person's perception of a product or activity. For example, a subjective measure would involve such attributes of an information system as how easy it is to use and the skill level needed to execute the system. As a general rule, subjective measures are much more important than objective measures. For example, it is more important to know how effective a person is in performing a job (a subjective measure) versus whether or not they got to work on time (an objective measure). QAI believes that the more difficult something is to measure, the more valuable that measure.

Individuals seem to want objective measures because they believe they are more reliable than subjective measures. It is unfortunate, but true, that many bosses are more concerned that the workers are at work on time and do not leave early, than they are about how productive they are during the day. You may have observed the type of people that always want to arrive at work before the boss, because they believe meeting objective measures is more important than meeting subjective measures, such as how easy the systems they built are to use.

How Do You Know a Metric is Good?

Before a measure is approved for use, there are certain tests that it must pass. QAI has identified the following tests that each measure and metric should be subjected to before it is approved for use:

a) Reliability: This refers to the consistency of measurement. If taken by two people, would the same results be obtained?

b) Validity: This indicates the degree to which a measure actually measures what it was intended to measure.

c) Ease of Use and Simplicity: These are functions of how easy it is to capture and use the measurement data.

d) Timeliness: This refers to whether the data was reported in sufficient time to impact the decisions needed to manage effectively.

e) Calibration: This indicates the movement of a metric so it becomes more valid, for example, changing a customer survey so it better reflects the true opinions of the customer.

What are the Standard Units of Measure?

A measure is a single attribute of an entity. It is the basic building block for a measurement program. Measurement cannot be used effectively until the standard units of measure have been defined. You cannot intelligently talk about lines of code until the measure lines of code have been defined. For example, lines of code may mean lines of code written, executable lines of code written, or even non-compound lines of code written. If a line of code was written that contained a compound statement it would be counted as two or more lines of code, such as a nested IF statement two levels deep. In addition, organizations may desire to use weighting factors; for example, one verb would be weighted as more complete than other verbs in the same programming language.

Measurement programs can be started with as few as five or six standard units of measure, but rarely would exceed 50 standard units of measure.

How do we compare Productivity versus Quality?

Quality is an attribute of a product or service. Productivity is an attribute of a process. They have frequently been called two sides of the same coin. This is because one has a significant impact on the other.

There are two ways in which quality can drive productivity. The first, and an undesirable method, is to lower or not meet quality standards. For example, if one chose to eliminate the testing and rework components of a system development process, productivity as measured in lines of code per hours worked would be increased. This is done frequently in information services under the guise of completing projects on time. While testing and rework may not be eliminated, they are not complete when the project is placed into production. The second method for improving productivity through

quality is to improve processes so that defects do not occur, thus minimizing the need for testing and rework. The QAI Approach uses quality improvement processes to drive productivity.

What are the Categories of Test Metric?

While there are no generally accepted categories of metrics, it has proved helpful to many test organizations to establish categories for status and reporting purposes. The metric categories and metrics within those categories provide an inventory of the metrics that testers will use in status reporting and final test reports.

In examining many reports prepared by testers the following eight metric categories are commonly used:

a) Metrics unique to test
b) Complexity measurements
c) Project metrics
d) Size measurements
e) Defect metrics
f) Product measures
g) Satisfaction metrics
h) Productivity metrics

What are the Metrics that are Unique to Test?

This category includes metrics such as Defect Removal Efficiency, Defect Density, and Mean Time to Last Failure. The following are examples of metrics unique to test:

a) Defect removal efficiency: the percentage of total defects occurring in a phase or activity removed by the end of that activity.

b) Defect density: the number of defects in a particular product.

c) Mean time to failure: the average operational time it takes before a software system fails.

d) Mean time to last failure: an estimate of the time it will take to remove the last defect from the software

e) Coverage metrics: the percentage of instructions or paths executed during tests.

f) Test cycles: the number of testing cycles required to complete testing that may be related to the size of the software system or complexity of the system.

g) Requirements tested: the percentage of requirements tested during testing

How do we measure the Complexity?

This category includes quantitative values accumulated by a predetermined method, which measure the complexity of a software product. The following are examples of complexity measures:

a) Size of module / unit:: larger module/units are considered more complex.

b) Logic complexity: the number of opportunities to branch/transfer within a single module.

c) Documentation complexity: the difficulty level in reading documentation usually expressed as an academic grade level.

Project Metrics: This category includes status of the project including milestones, budget and schedule variance and project scope changes.

The following are examples of project metrics:
a) Percent of budget utilized
b) Days behind or ahead of schedule
c) Percent of change of project scope
d) Percent of project completed (not a budget or schedule metric, but rather an assessment of the functionality/structure completed at a given point in time)

Size Measurements: This category includes methods primarily developed for measuring the software size of software systems, such as lines of code, and function points. These can also be used to measure software-testing productivity. Sizing is important in normalizing data for comparison to other projects.

The following are examples of size metrics:

a) KLOC: thousand lines of code, used primarily with statement level languages.

b) Function points: a defined unit of size for software.

c) Pages or words of documentation

Defect Metrics: This category includes values associated with numbers or types of defects, usually related to system size, such as “defects/1000 lines of code” or “defects/100 function points,” severity of defects, uncorrected defects, etc.

The following are examples of defect metrics:

a) Defects related to size of software.
b) Severity of defects such as very important, important, and unimportant.
c) Priority of defects: the importance of correcting defects.
d) Age of defects: the number of days the defect has been uncovered but not corrected.
e) Defects uncovered in testing
f) Cost to locate a defect

Product Measures: This category includes measures of a product’s attributes such as performance, reliability, usability.

The following are examples of product measures:

a) Defect density : the expected number of defects that will occur in a product during development.

Satisfaction Metrics: This category includes the assessment of customers of testing on the effectiveness and efficiency of testing.

The following are examples of satisfaction metrics:

a) Ease of use: the amount of effort required to use software and/or software documentation.

b) Customer complaints: some relationship between customer complaints and size of system or number of transactions processed.

c) Customer subjective assessment: a rating system that asks customers to rate their satisfaction on different project characteristics on a scale, for example a scale of 1-5.

d) Acceptance criteria met: the number of user defined acceptance criteria met at the time software goes operational.

e) User participation in software development: an indication of the user desire to produce high quality software on time and within budget.

Productivity Metrics: This category includes the effectiveness of test execution.

Examples of productivity metrics are:

a) Cost of testing in relation to overall project costs: assumes a commonly accepted ratio of the costs of development versus tests.
b) Under budget/Ahead of schedule.
c) Software defects uncovered after the software is placed into an operational status.
d) Amount of testing using automated tools.