Useful Automated Software Testing Metrics

By Thom Garrett
IDT, LLC

Adapted from the book "Implementing Automated Software Testing," by Elfriede Dustin, Thom Garrett, Bernie Gauf

Author Bio: Thom Garrett

Thom Garrett has twenty years of Information Technology experience in planning, development, testing and deployment of complex processing systems for U.S. Navy and commercial applications. Specific experience includes rapid introduction and implementation of new technologies for highly sophisticated architectures which support users world-wide. In addition, he has experience in managing and testing all aspects of large scale complex networks used in 24/7 environments.

Thom currently works for Innovative Defense Technologies (IDT), LLC, and has previously worked for companies such as America Online (AOL) and Digital System Resources (DSR), Inc., supporting system engineering solutions from requirements gathering to production roll-out.

Thom received a Bachelor of Science degree in Mathematics / Computer Science from Virginia Commonwealth University and a Master's degree in Information Systems from the University of San Francisco.

Useful Automated Software Testing Metrics

> ***"When you can measure what you are speaking about, and can express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind."***
> -- Lord Kelvin, a physicist.

As part of a successful automated testing program it is important that goals and strategies are defined and then implemented. During implementation progress against these goals and strategies set out to be accomplished at the onset of the program needs to be continuously tracked and measured. This article discusses various types of automated and general testing metrics that can be used to measure and track progress.

Based on the outcome of these various metrics the defects remaining to be fixed in a testing cycle can be assessed; schedules can be adjusted accordingly or goals can be reduced. For example, if a feature is still left with too many high priority defects a decision can be made that the ship date is moved or that the system is shipped or even goes live without that specific feature.

Success is measured based on the goal we set out to accomplish relative to the expectations of our stakeholders and customers.

Automated Testing Metrics

Metrics can aid in improving your organizations automated testing process and tracking its status. These metrics and techniques have successfully been used by our software test teams. As the quote at the beginning of this article implies, if you can measure something, then you have something you can quantify. If you can quantify something, then you can explain it in more detail and know something more about it. If you can explain it, then you have a better chance to attempt to improve upon it, and so on.

As time proceeds, software project become more complex due to increased lines of code because of added features, bug fixes, etc. Also the task is asked to be done in less time with fewer people. The complexity over time will have a tendency to decrease the test coverage and ultimately affect the quality of the product. Other factors involved over time are the overall cost of the product and the time to deliver the software. Metrics can provide insight into the status of automated testing efforts.

When done properly, implementing automation software testing will reverse the negative trend. Automation efforts can provide a larger test coverage area and increase the overall quality of the product. Automation can also reduce the time of testing and the cost of

delivery. This benefit is typically realized over multiple test cycles and project cycles. Automated testing metrics can aid in making assessments as to whether progress, productivity and quality goals are being met.

What is a Metric?

The basic definition of a metric is a standard of measurement. It also can be described as a system of related measures that facilitates the quantification of some particular characteristic.[1] For our purposes, a metric can be looked at as a measure which can be utilized to display past and present performance and/or used for predicting future performance.

What Are Automated Testing Metrics?

Automated testing metrics are metrics used to measure the performance (e.g. past, present, future) of the implemented automated testing process.

What Makes A Good Automated Testing Metric?

As with any metrics, automated testing metrics should have clearly defined goals of the automation effort. It serves no purpose to measure something for the sake of measuring. To be meaningful, it should be something that directly relates to the performance of the effort.

Prior to defining the automated testing metrics, there are metrics setting fundamentals you may want to review. Before measuring anything, set goals. What is it you are trying to accomplish? Goals are important, if you do not have goals, what is it that you are measuring? It is also important to continuously track and measure on an ongoing basis. Based on the metrics outcome, then you can decide if changes to deadlines, feature lists, process strategies, etc., need to be adjusted accordingly. As a step toward goal setting, there may be questions that need to be asked of the current state of affairs. Decide what questions can be asked to determine whether or not you are tracking towards the defined goals. For example:

- How much time does it take to run the test plan?
- How is test coverage defined (KLOC, FP, etc)?
- How much time does it take to do data analysis?
- How long does it take to build a scenario/driver?
- How often do we run the test(s) selected?
- How many permutations of the test(s) selected do we run?
- How many people do we require to run the test(s) selected?
- How much system time/lab time is required to run the test(s) selected?

[1] http://www.thefreedictionary.com/metric

- etc.

In essence, a good automated testing metric has the following characteristics:

- is Objective
- is Measurable
- is Meaningful
- has data that is easily gathered
- can help identify areas of test automation improvement
- is Simple

A good metric is clear and not subjective, it is able to be measured, it has meaning to the project, it does not take enormous effort and/or resources to obtain the data for the metric, and it is simple to understand. A few more words about metrics being simple. Albert Einstein once said

> ***"Make everything simple as possible, but not simpler."***

When applying this wisdom towards software testing, you will see that:

- Simple reduces errors
- Simple is more effective
- Simple is elegant
- Simple brings focus

It is important to generate a metric that calculates the value of automation, especially if this is the first time the project has used an automated testing approach. The test team will need to measure the time spent on developing and executing test scripts against the results that the scripts produced. For example, the test team could compare the number of hours to develop and execute test procedures by the number of defects documented that would not likely have been revealed during a manual test effort.

Sometimes it is hard to quantify or measure the automation benefits. For example, often defects are discovered using automated testing tools, which manual test execution could not have discovered. For example, during stress testing 1000 virtual users execute a specific functionality and the system crashes. It would be very difficult to discover this problem manually, using 1000 test engineers. Another way to minimize the test effort involves the use of an automated test tool for data entry or record setup. The metric, which applies in this case, measures the time required to manually set up the needed records versus the time required to set up the records using an automated tool.

Consider the test effort associated with the system requirement that reads, "The system shall allow the addition of 10,000 new accounts". Imagine having to manually enter

10,000 accounts into a system, in order to test this requirement! An automated test script can easily support this requirement by reading account information from a file through the use of a looping construct. The data file can easily generated using a data generator. The effort to verify this system requirement using test automation requires far fewer number of man hours than performing such a test using manual test methods.[2]

Automated software testing metrics can be used to determine additional test data combinations. For example, with manual testing you might have been able to test 'x' number of test data combinations; with automated testing you are now able to test 'x+y' test data combinations. Defects that were uncovered in the set of 'y' combinations are the defects that manual testing may have never uncovered.

Percent Automatable

At the beginning of an automated testing effort, the project is either automating existing manual test procedures, starting a new automation effort from scratch, or some combination of both. Whichever the case, a percent automatable metric can be determined.

Percent automatable can be defined as: of a set of given test cases, how many are automatable? This could be represented in the following equation:

$$PA\,(\%) = \frac{ATC}{TC} = \left(\frac{\#\ of\ test\ cases\ automatable}{\#\ of\ total\ test\ cases}\right)$$

PA = Percent Automatable
ATC = # of test cases automatable
TC = # of total test cases

In evaluating test cases to be developed, what is to be considered automatable and what is not to be considered automatable? Given enough ingenuity and resources, one can argue that almost anything can be automated. So where do you draw the line? Something that can be considered 'not automatable' for example, could be an application area that is still under design, not very stable, and much of it is in flux. In cases such as this, we should:

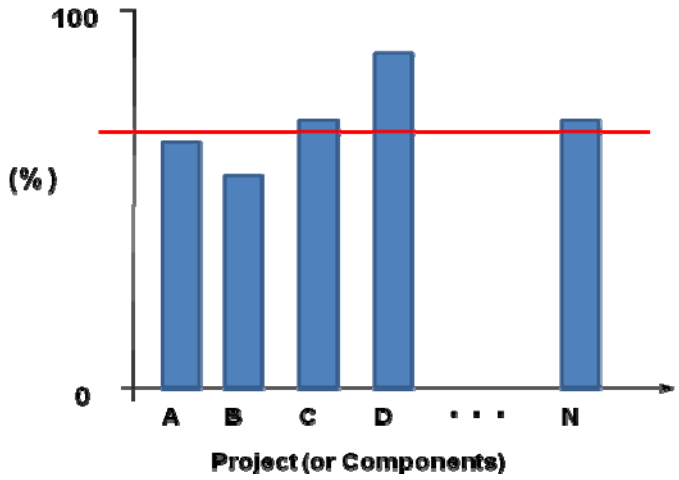
"evaluate whether it make sense to automate"

[2] Adapted from "Automated Software Testing" Addison Wesley, 1999, Dustin, et al

We would evaluate for example, given the set of automatable test cases, which ones would provide the biggest return on investment:

"just because a test is automatable doesn't necessary mean it should be automated"

When going through the test case development process, determine what tests can be AND makes sense to automate. Prioritize your automation effort based on your outcome. This metric can be used to summarize, for example, the % automatable of various projects or component within a project, and set the automation goal.



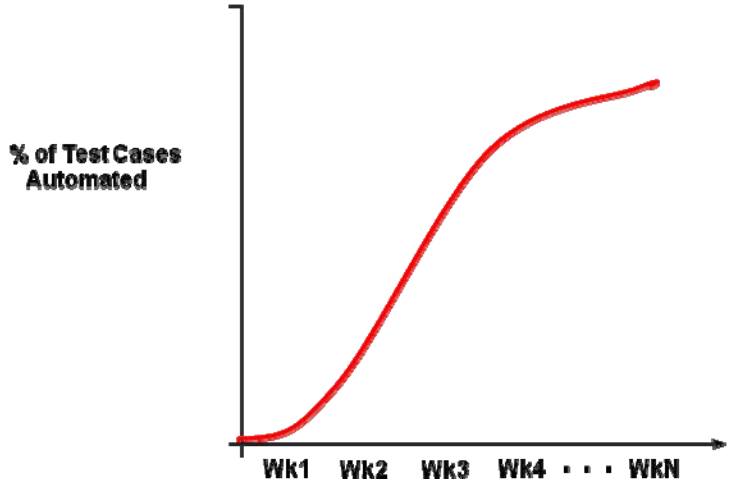
Automation Progress

Automation Progress refers to, of the percent automatable test cases, how many have been automated at a given time? Basically, how well are you doing in the goal of automated testing? The goal is to automat 100% of the "automatable" test cases. This metric is useful to track during the various stages of automated testing development.

$$AP\ (\%) = \frac{AA}{ATC} = \left(\frac{\#\ of\ actual\ test\ cases\ automated}{\#\ of\ test\ cases\ automatable} \right)$$

AP = Automation Progress
AA = # of actual test cases automated
ATC = # of test cases automatable

The Automation Progress metric is a metric typically tracked over time. In the case below, time in "weeks".

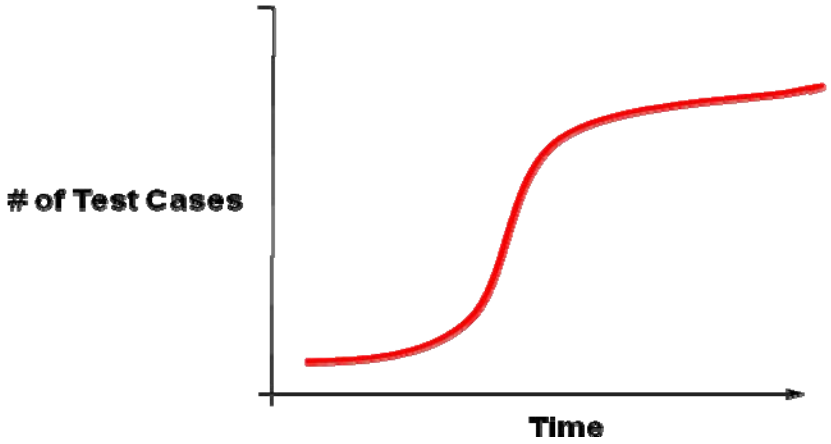


A common metric closely associated with progress of automation, yet not exclusive to automation is Test Progress. Test progress can simply be defined as the number of test cases attempted (or completed) over time.

$$TP = \frac{TC}{T} = \left(\frac{\text{\# of test cases (attempted or completed)}}{\text{time (days/weeks/months, etc)}} \right)$$

TP = Test Progress
TC = # of test cases (either attempted or completed)
T = some unit of time (days / weeks / months, etc)

The purpose of this metric is to track test progress and compare it to the plan. This metric can be used to show where testing is tracking against the overall project plan. Test Progress over the period of time of a project usually follows an "S" shape. This typical "S" shape usually mirrors the testing activity during the project lifecycle. Little initial testing, followed by an increased amount of testing through the various development phases, into quality assurance, prior to release or delivery.



This is a metric to show progress over time. A more detailed analysis is needed to determine pass/fail, which can be represented in other metrics.

Percent of Automated Testing Test Coverage

Another automated software metric we want to consider is Percent of Automated Testing Test Coverage. That is a long title for a metric to determine what test coverage is the automated testing actually achieving? It is a metric which indicates the completeness of the testing. This metric is not so much measuring how much automation is being executed, but rather, how much of the product's functionality is being covered. For example, 2000 test cases executing the same or similar data paths may take a lot of time and effort to execute, does not equate to a large percentage of test coverage. Percent of automatable testing coverage does not specify anything about the effectiveness of the testing taking place, it is a metric to measure its' dimension.

$$PTC(\%) = \frac{AC}{C} = \left(\frac{\text{automation coverage}}{\text{total coverage}} \right)$$

PTC = Percent of Automatable testing coverage
AC = Automation coverage
C = Total Coverage (KLOC, FP, etc)

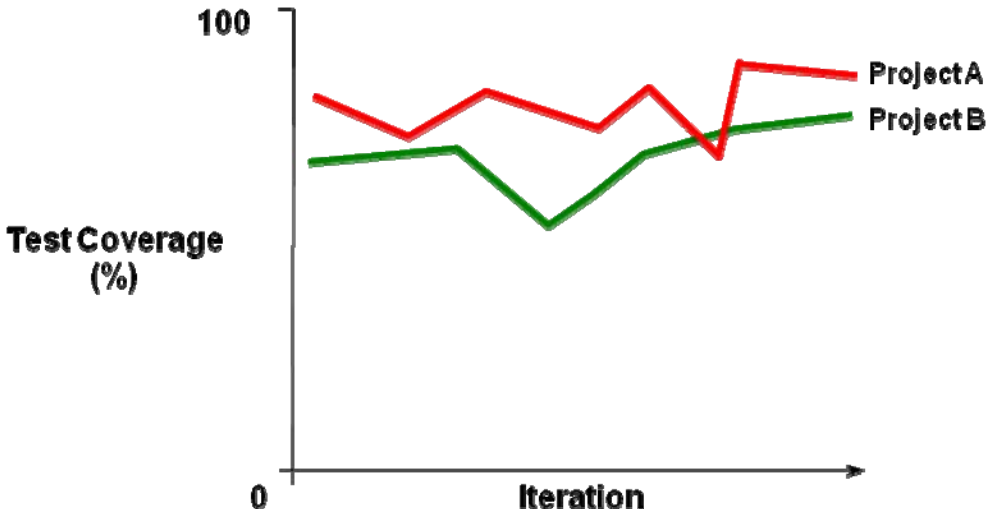
Size of system is usually counted as lines of code (KLOC) or function points (FP). KLOC is a common method of sizing a system, however, FP has also gained acceptance. Some argue that FPs can be used to size software applications more accurately. Function Point Analysis was developed in an attempt to overcome difficulties associated with KLOC (or just LOC) sizing. Function Points measure software size by quantifying the functionality provided to the user based logical design and functional specifications. There is a wealth

of material available regarding the sizing or coverage of systems. A useful resourse is Stephen H Kan's book entitled "Metrics and Models in Software Quality Engineering" (Addison Wesley, 2003).

The Percent Automated Test Coverage metric can be used in conjunction with the standard software testing metric called Test Coverage.

$$TC(\%) = \frac{TTP}{TTR} = \left(\frac{\text{total \# of TP}}{\text{total \# of Test Requirements}} \right)$$

TC = Percent of Testing Coverage
TTP = Total # of Test Procedures developed
TTR = Total # of defined Test Requirements



This measurement of test coverage divides the total number of test procedures developed, by the total number of defined test requirements. This metric provides the test team with a barometer to gage the depth of test coverage. The depth of test coverage is usually based on the defined acceptance criteria. When testing a mission critical system, such as operational medical systems, the test coverage indicator would need to be high relative to the depth of test coverage for non-mission critical systems. The depth of test coverage

for a commercial software product that will be used by millions of end users may also be high relative to a government information system with a couple of hundred end users. [3]

Defect Density

Measuring defects is a discipline to be implemented regardless if the testing effort is automated or not. Josh Bloch, Chief Architect at Google stated:

> ***"Regardless of how talented and meticulous a developer is, bugs and security vulnerabilities will be found in any body of code – open source or commercial.", "Given this inevitably, it's critical that all developers take the time and measures to find and fix these errors."***

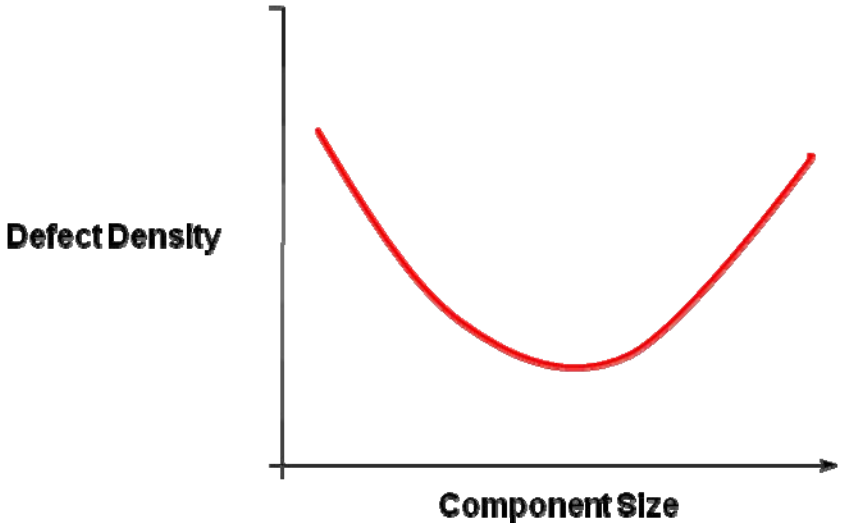
Defect density is another well known metric not specific to automation. It is a measure of the total known defects divided by the size of the software entity being measured. For example, if there is a high defect density in a specific functionality, it is important to conduct a causal analysis. Is this functionality very complex, and therefore it is to be expected that the defect density is high? Is there a problem with the design/implementation of the functionality? Were the wrong (or not enough) resources assigned to the functionality, because an inaccurate risk had been assigned to it? It also could be inferred that the developer, responsible for this specific functionality, needs more training.

$$DD = \frac{D}{SS} = \left(\frac{\text{\# of known defects}}{\text{total size of system}} \right)$$

DD = Defect Density
D = # of known defects
SS = Total Size of system

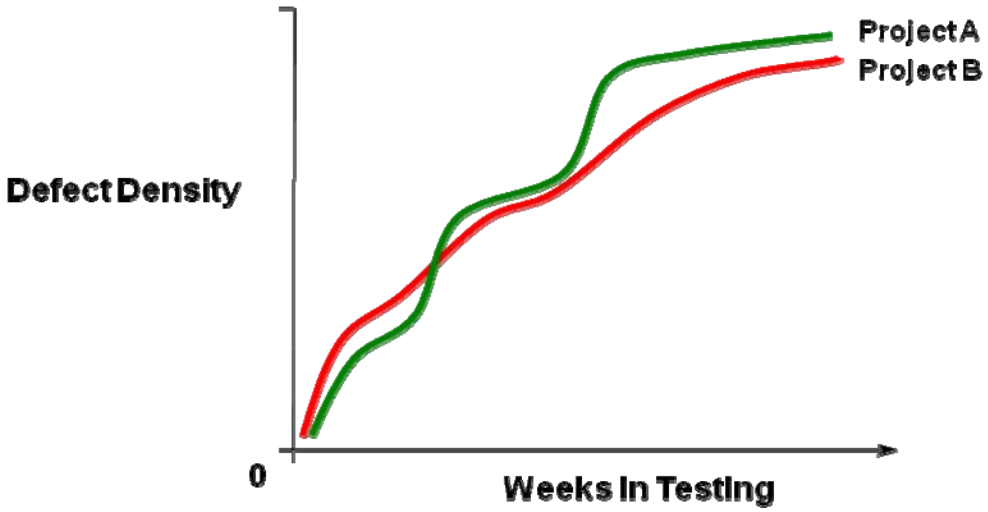
One use of defect density is to map it against software component size. A typical defect density curve that we have experienced looks like the following, where we see small and lager sized components having a higher defect density ratio as shown below.

[3] Adapted from "Automated Software Testing" Addison Wesley, 1999, Dustin, et al



Additionally, when evaluating defect density, the priority of the defect should be considered. For example, one application requirement may have as many as 50 low priority defects and still pass because the acceptance criteria have been satisfied. Still, another requirement might only have one open defect that prevents the acceptance criteria from being satisfied because it is a high priority. Higher priority requirements are generally weighted heavier.

The graph below shows one approach to utilizing the defect density metric. Projects can be tracked over time (for example, stages in the development cycle).

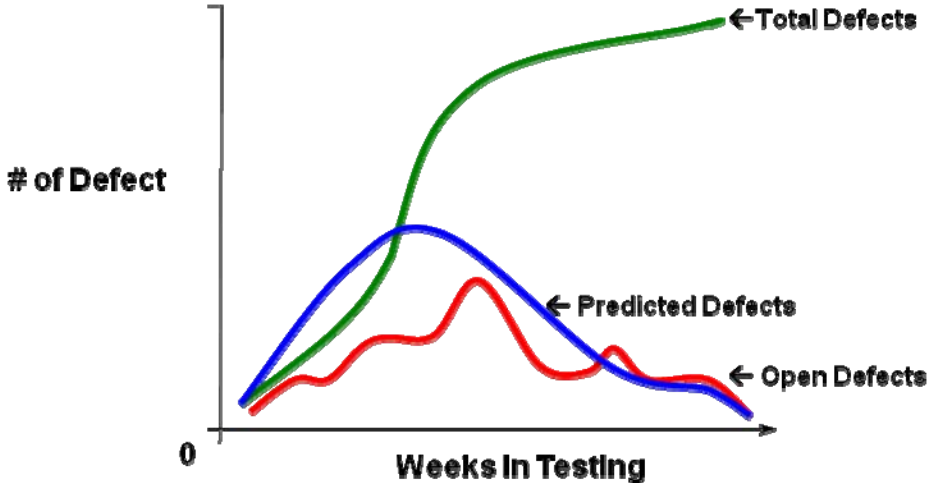


Another closely related metric to Defect Density is Defect Trend Analysis. Defect Trend Analysis is calculated as:

$$DTA = \frac{D}{TPE} = \left(\frac{\#\ of\ known\ defects}{\#\ of\ test\ procedures\ executed} \right)$$

DTA = Defect Trend Analysis
D = # of known Defects
TPE = # of Test Procedures Executed over time

Defect Trend Analysis can help determine the trend of defects found. Is the trend improving as the testing phase is winding down or is the trend worsening? Defects the test automation uncovered that manual testing didn't or couldn't have is an additional way to demonstrate ROI. During the testing process, we have found defect trend analysis one of the more useful metrics to show the health of a project. One approach to show trend is to plot total number of defects along with number of open Software Problem Reports as shown in the graph below.



[4]

Effective Defect Tracking Analysis can present a clear view of the status of testing throughout the project. A few additional common metrics sometimes used related to defects are as follows:

- **Cost to locate defect** = Cost of testing / the number of defects located
- **Defects detected in testing** = Defects detected in testing / total system defects

[4] Graph adapted from article: http://www.teknologika.com/blog/SoftwareDevelopmentMetricsDefectTracking.aspx

- **Defects detected in production** = Defects detected in production/system size

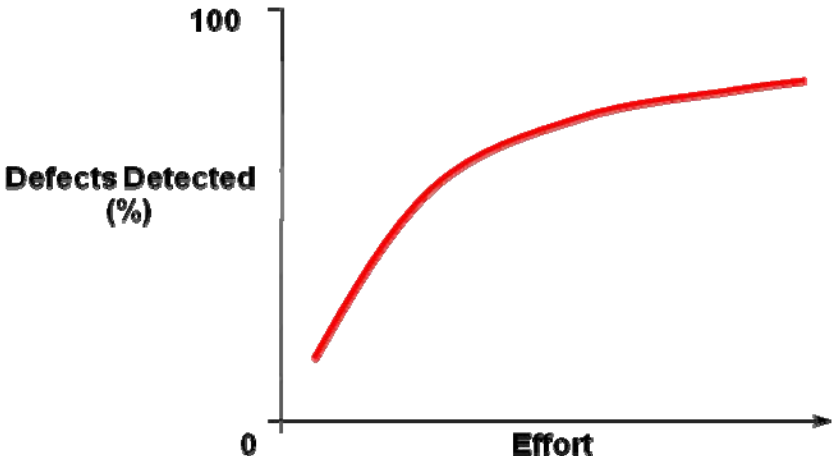
Some of these metrics can be combined and used to enhance quality measurements as shown in the next section.

Actual Impact on Quality

One of the more popular metrics for tracking quality (if defect count is used as a measure of quality) through testing is Defect Removal Efficiency (DRE), not specific to automation, but very useful when used in conjunction with automation efforts. DRE is a metric used to determine the effectiveness of your defect removal efforts. It is also an indirect measurement of the quality of the product. The value of the DRE is calculated as a percentage. The higher the percentage, the higher positive impact on the quality of the product. This is because it represents the timely identification and removal of defects at any particular phase.

$$DRE(\%) = \frac{DT}{DT + DA} = \left(\frac{\text{\# of defects found during testing}}{\text{\# of defects found during testing} + \text{\# of defect found after delivery}} \right)$$

DRE = Defect Removal Efficiency
DT = # of defects found during testing
DA = # of defects acceptance defects found after delivery



The highest attainable value of DRE is "1" which equates to "100%". In practice we have found that an efficiency rating of 100% is not likely. DRE should be measured during the different development phases. If the DRE is low during analysis and design, it may indicate that more time should be spent improving the way formal technical reviews are conducted, and so on.

This calculation can be extended for released products as a measure of the number of defects in the product that were not caught during the product development or testing phase.

Other Software Testing Metrics

Along with the metrics mentioned in the previous sections, here are a few more common test metrics. These metrics do not necessarily just apply to automation, but could be, and most often are, associated with software testing in general. These metrics are broken up into three categories:

- **Coverage:** Meaningful parameters for measuring test scope and success.
  .
- **Progress:** Parameters that help identify test progress to be matched against success criteria. Progress metrics are collected iteratively over time. They can be used to graph the process itself (e.g. time to fix defects, time to test, etc).

- **Quality:** Meaningful measures of excellence, worth, value, etc. of the testing product. It is difficult to measure quality directly; however, measuring the effects of quality is easier and possible.

Metric Name	Description	Category
Test Coverage	Total number of test procedures/total number of test requirements. The *Test Coverage* metric will indicate planned test coverage.	Coverage
System Coverage Analysis	The *System Coverage Analysis* measures the amount of coverage at the system interface level.	Coverage
Test Procedure Execution Status	Executed number of test procedures/total number of test procedures This *Test Procedure Execution* metric will indicate the extent of the testing effort still outstanding.	Progress
Error Discovery Rate	Number total defects found/number of test procedures executed. The *Error Discovery Rate* metric uses the same calculation as the defect density metric. Metric used to analyze and support a rational product release decision	Progress
Defect Aging	Date Defect was opened versus date defect was fixed *Defect Aging* metric provides an indication of turnaround of the defect.	Progress
Defect Fix Retest	Date defect was fixed & released in new build versus date defect was re-tested. The *Defect Fix Retest* metric provides an idea if the testing team is re-testing the fixes fast enough, in order to get an accurate progress metric	Progress
Current Quality Ratio	Number of test procedures successfully executed (without defects) versus the number of test procedures. *Current Quality Ratio* metric provides indications about the amount of functionality that has successfully been demonstrated.	Quality
Quality of Fixes	Number total defects reopened/total number of defects fixed This *Quality of Fixes* metric will provide indications of development issues.	Quality
	Ratio of previously working functionality versus new errors introduced The *Quality of Fixes* metric will keep track of how often previously working functionality was adversarial affected by software fixes.	Quality
Problem Reports	Number of Software Problem Reports broken down by priority. The *Problem Reports Resolved* measure counts the number of software problems reported, listed by priority.	Quality
Test Effectiveness	Test effectiveness needs to be assessed statistically to determine how well the test data has exposed defects contained in the product.	Quality
Test Efficiency	Number of test required / the number of system errors	Quality

Common Software Test Metrics[5]

Summary

Metrics are an important gauge of the health, quality, and progress of an automated software testing effort. Metrics can also be used to perform past performance, current status, and future trends. Good metrics are objective, measureable, meaningful, simple,

[5] Adapted from "Automated Software Testing" Addison Wesley, 1999, Dustin, et al

and have easily obtainable data. Traditional software testing metrics used in software quality engineering can be applied and adapted to automated software testing. Some metrics specific to automated testing are:

- Percent Automatable
- Automation Progress
- Percent of Automated Testing Coverage

In the test case requirements gathering phase of your automation effort, evaluate whether it makes sense to automate or not. Given the set of automatable test cases, determine which ones would provide the biggest return on investment. Consider that just because a test is automatable doesn't necessary mean it should be automated.

Acronyms:

AA	-	# of actual test cases automated
AC	-	Automation coverage
AP	-	Automation Progress
ATC	-	# of test cases automatable
D	-	# of known Defects
DA	-	# of defects acceptance defects found after delivery
DD	-	Defect Density
DRE	-	Defect Removal Efficiency
DT	-	# of defects found during testing
DTA	-	Defect Trend Analysis
FP	-	Function Point
KLOC	-	Lines Of Code (Thousands)
LOC	-	Lines Of Code
PR	-	Percent Automatable
PTC	-	Percent of Automatable testing coverage
ROI	-	Return On Investment
SPR	-	Software Problem Report
SS	-	Total Size of system to be automated
T	-	Time (some unit of time (days / weeks / months, etc))
TC	-	# of total test cases
TP	-	Test Progress
TPE	-	# of Test Procedures Executed over time