



PACIFIC NW  
28TH ANNUAL  
SOFTWARE  
QUALITY  
CONFERENCE

OCTOBER 18TH – 19TH, 2010



ACHIEVING  
QUALITY  
IN A COMPLEX  
ENVIRONMENT

*Conference Paper Excerpt  
from the*  
CONFERENCE  
PROCEEDINGS

---

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

# Driving Product Quality towards Release Goals

**Bhushan Gupta**  
NIKE Inc.

## Abstract

The software quality assurance activities strive to achieve utmost product quality at release given the scope, schedule, and resources. Normally, a program management team, comprising of all stakeholders, establishes a release criteria that functions as a measuring stick for the product quality at release. At a minimum, the release criteria includes test execution and defect goals from the quality perspective.

This paper describes a set of metrics that a software quality assurance group can provide to the program management team to assess the product quality and make release decisions. The author divides the quality activities during the product development into two phases; “New Functionality” testing and “Regression” testing and discusses test coverage such as total test cases planned, percentage of planned test cases executed and defect characteristics; such as number of defects by severity and their trends in each phase. As one would expect the defect find rate is a function of multiple factors including testing phase, test effectiveness, execution rate, and product complexity. Tracking each factor provides insight into on-going product quality and therefore an understanding of progress towards release. The author applies the concept of “code volatility” to further validate the defect characteristics. The trends in defect behaviors are explained with regards to the influencing factors highlighted above.

## Biography

Bhushan Gupta has twenty five years of experience in software engineering, fifteen of which have been in the software industry. Bhushan recently joined the Global Store Solutions group at NIKE Inc. as a Software Quality Lead after a thirteen year tenure at Hewlett-Packard. He joined HP as a software quality engineer in 1997 where he was responsible for identifying key process areas to reduce rework. Based upon Hewlett-Packard quality parameters he developed quality goals, quality plans, and software validation strategies for several products. As a software process architect at HP, he customized the agile lifecycle for different groups and developed productivity metrics.

From 1995-97 Bhushan Gupta worked as a Systems Analyst at Consolidated Freightways. He was a faculty member of the Software Engineering Technology department at Oregon Institute of Technology from 1985 to 1995.

As a change agent, Bhushan volunteers his time and energy for organizations that promote software quality. He has been a Vice President, a Program Co-Chair, and a member of the board of directors of the Pacific Northwest Software Quality Conference.

Bhushan Gupta has a MS degree in Computer Science from New Mexico Institute of Mining and Technology, Socorro, New Mexico, 1985.

## Introduction:

Data-driven decision-making takes out the gut-feels, hunches, and emotions from the decision making process and provides an analytical approach to make informed decisions. A critical decision an organization makes is when to release a product. Releasing an inferior quality product often has higher support cost. Delaying a release to achieve perfection incurs development cost and lost opportunities causing a revenue loss. It is only prudent to strike a balance between the support cost and the cost of delayed release. The balance is maintained by enforcing a release criteria once the business goals have been established. Once adopted by the stakeholders, the release criteria becomes the measuring stick and is a static control document throughout the product development.

Typical release criteria for a software product includes requirements coverage, defect find trend – normally declining for a certain period after the new functionality has been tested, and number of high severity defects that might adversely impact the support cost. The scope of this paper is the quality assurance metrics and actions that will drive the product quality towards a scheduled release.

## Test Execution Phases:

To establish common understanding and clarity, it is necessary to understand the nature of activities involved in testing and how they can impact the quality metrics. In a broad sense, the testing lifecycle can be divided into two phases: testing new functionality and regression of the existing functionality. The two phases have very distinct activities as described below:

**Testing New Functionality:** This is the most prevalent reason for validating a new or a follow up release and has the following primary activities:

- Enhanced understanding of the product functionality
- Building test strategy and developing test cases
- Enhancing test cases for effectiveness and efficiency
- Clear and concise defect logging
- Helping developers with defect reproduction and further characterization

**Regression Phase:** In this phase the testing goal shifts to verifying defect fixes and determine any adverse effects caused by the fix on the already working functionality. The activities in this phase are:

- Re-execution of test cases to verify a defect fix
- If a fix fails, reopen the defect
- Re-execution of a subset of test cases to assure that a fix has not broken the existing functionality. Determination of the sub-set of test cases requires careful consideration.
- Finding new defects – the probability of finding new defects is low.

## Release Criteria:

Formal or informal, each product development effort has a release criteria. A release criteria manifest is a list of conditions agreed upon by the program management team that should be met for market release. Its purpose is to achieve marketing goals with no or minimal support cost. Once established, it becomes a control document and guides the product development. The release criteria can only be altered if the business conditions have changed. Rothman [Rothman 2002] has discussed the mechanics of developing a release criteria and its use to determine the software release readiness.

The level of sophistication of a release criteria also depends upon the scope of the product being delivered; a hot fix, a patch, re-release, or a brand new product. At a minimum, a release criteria includes:

- Requirements coverage
- Percent test coverage
- Defect severity find rate declining over a period of time

A more sophisticated release criteria would have additional aspects such as:

- Maximum allowable number of defects that have been fixed but not verified
- Maximum allowable number of open defects by severity
- Declining code volatility trend
- Release Readiness
  - Training and support plans readiness
  - Completion of invention disclosures

The criteria may vary between organizations and within an organization. There may be other items in the release criteria if the software product has life threatening potential. Tabulated below is an example of a typical release criteria manifest:

**Table 1: A Release Criteria Manifest Example**

Criterion	Description	Owner
Functionality	100 Percent of the planned requirements completed	Development Manager
Test Coverage	100 percent test cases executed	QA Manager
Defects	<ul style="list-style-type: none"> <li>• 100 percent “high severity” defects addressed as either:               <ol style="list-style-type: none"> <li>1. Fixed, verified, and closed</li> <li>2. Workarounds available where possible</li> <li>3. Support cost estimated and agreed upon</li> </ol> </li> <li>• 100 Percent customer impacting “medium severity” defects understood</li> <li>• 100 percent defect fixes verified</li> <li>• Find rate declining for 3 consecutive weeks</li> <li>• No more than 10 percent increase in overall number of open defects in the application since last release</li> </ul>	Program Manager
Support Readiness	<ul style="list-style-type: none"> <li>• Release notes up to date with workarounds where available</li> <li>• Documentation/Training material ready</li> </ul>	Support Manager
Marketing Readiness	<ul style="list-style-type: none"> <li>• Rollout plans ready and communicated</li> </ul>	Marketing Manager
Development	<ul style="list-style-type: none"> <li>• Intellectual property activities completed</li> <li>• Open defects moved to next release</li> </ul>	Program Manager

The rest of the paper describes the metrics that can be used to track the quality of the product during product development to meet the release criteria.

### **Is testing on track? Requirements Coverage:**

The requirements define product functionality which in turn establishes the product marketability. Once established, an organization’s goal is to develop a product that meets all the planned requirements. It is imperative that the organization tracks the product requirements that have been fulfilled. For a quality assurance group it is important that all the requirements are validated. There is often a one-to-many mapping from the requirements to test cases. From the quality assurance perspective, the requirements coverage translates to test coverage and thus a QA organization measures the requirements in terms of test coverage. Quality assurance tool such as “Quality Center<sup>®</sup>” provides mechanisms to build traceability between requirements and test cases. In the absence of a tool, a QA team often uses a spreadsheet. The set of test cases are simply associated to a functionality and are tracked by it.

The requirements coverage is measured by the number of test cases that have been executed at any time during the product development. It is traditional to assess the development progress each week.

The QA also measures progress by the number of test cases executed each week. A program manager who is interested in a timely completion of the program will often ask QA ‘Is testing on track?’. A development manager will be interested in finding which functional aspects of the product are complete. The QA manager can address both of these questions using a simple metric shown in below:

**Table 2a: A Simple Test Coverage Metric**

Functional Area	Test Cases Planned	Planned for execution till date	Executed till date	% of Total planned Executed	% Test Cases Passed	Test cases blocked
User Interface	45	35	32	71	98	3
Database	195	45	12	27	75	8
Output Display	25	20	15	60	99	2
Reporting	40	10	10	25	50	1
<b>Total</b>	<b>305</b>	<b>110</b>	<b>69</b>	<b>22</b>	<b>87</b>	<b>14</b>

The metric emphasizes the following points:

- How is each functional area progressing towards release? The delta between planned for execution till date and executed till date is the measure of progress towards release. If the delta is positive or zero the test coverage is well on schedule.
- What is the quality of each component? Percent test cases passed signifies the quality of each component. The lower the rate of test cases passed, the lower the quality and attention is needed.
- What is the outlook for the future? The extent of blocked test case adversely impacts the future progress. A higher number of blocked test cases means more work is passed on to the future thereby back loading testing activity and potentially impacting the overall quality and the release schedule. In addition to assessing the quality of each functionality, the metric also provides a measure of overall progress and current quality of release.

Based upon the answers to these questions the organization can adjust resources in the test group or provide additional support to the development group.

Normally the data is collected at regular intervals and can be compiled as trend chart for each functional area. The overall result can be presented as a Status Dashboard for the program management and the development team to evaluate the release status as shown below. The data is derived from the test coverage metric (Table 2a).

**Table 2b: Status Dashboard Showing Quality of Various Components of a Product**

Functional Area	User Interface	Database	Output Display	Reporting
<b>Status</b>	Green	Red	Yellow	Red

The color notation has standard meaning and communicates the following:

Green – the quality is progressing as expected

Yellow – the product quality is not progressing as expected but the progress will improve

Red – The product quality is not progressing as expected and will require specific measures to improve

In the example above, both the Database and the Reporting functional areas are RED. For the database functionality only 27% of the planned test cases have been executed and the failure rate is relatively high (25%). In the case of Reporting, although the execution rate is 100%, the failure rate is high (50%). The two functional areas require immediate attention and a course correction.

## Does the product have desired quality?

So far we have only considered test coverage as a component of quality. The other equally significant and might be even more critical component is the defects that have been discovered during testing. At any instant, we may have completed the planned testing but we are still finding a substantial number of defects. In addition, there may be an excessive number of defects that are yet to be resolved and still a significant number of defects fixed but yet to be verified to make sure that the fix is effective and has not caused any side effects.

The succeeding paragraphs describe some effective metrics used to address these concerns and make progress towards release:

### Defect Find Rate:

The following graph illustrates a weekly defect find rate.

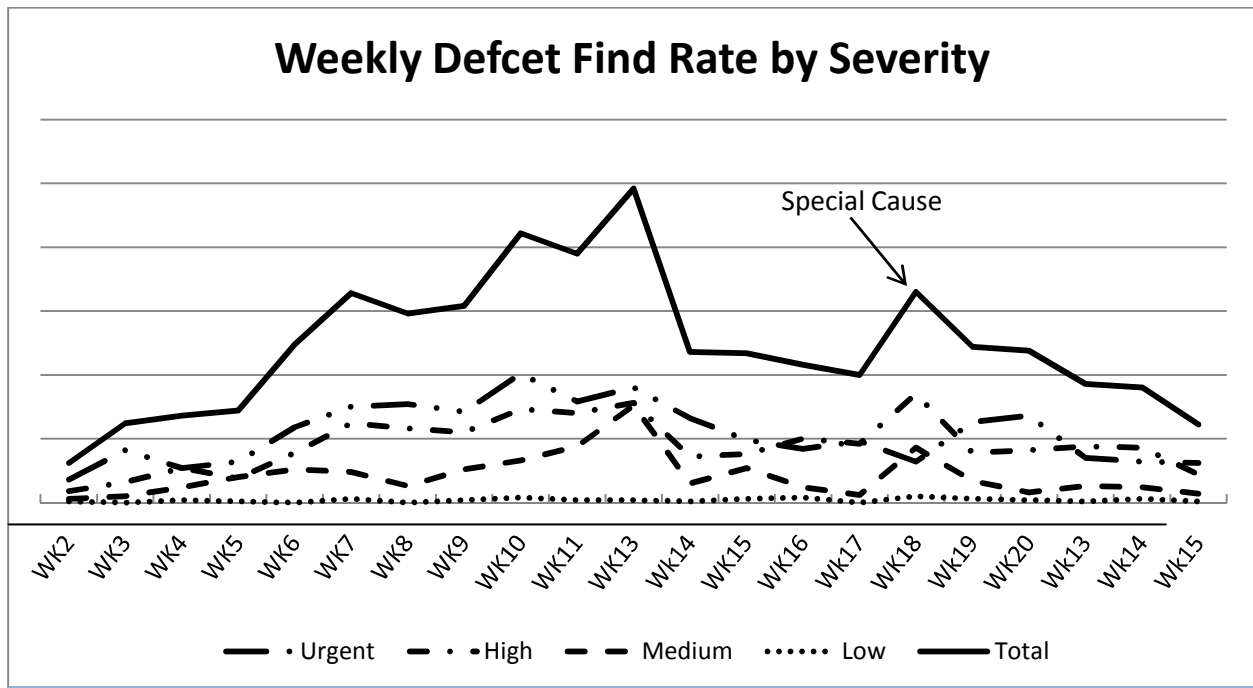


Figure 1 – Defect Find Rate by Severity

Initially, find rate is usually low due to:

- Limited understanding of the functionality by the test group on what the expected behavior is. This particularly is the case when the application workflow is complex or has been significantly changed in case of product versioning
- Planned functionality not yet complete
- Although the functionality is complete but not working end-to-end and thus certain aspects of the functionality are not reachable. This will be signaled by a high number of blocked test cases

As the testing progresses, the QA group better understands the functionality and the find rate increases. This rate peaks out when all the new functionality has been tested and the testing moves to regression. During the regression phase, only regression tests are run, fewer defects are found, and find rate starts to decline. As the development progresses, the find rate continues to taper off. It is not practically possible to find all the defects in the code so normally the release date sets the point in time when the testing should stop. Simmons [Simmons 2000] has modeled the defect arrival using the Weibull curve to answer the question “When Will We Be Done Testing?” Gupta [Gupta 2004] has utilized the concept of defect density to predict defect find rate.

While this is an ideal behavior there are normally fluctuations due to special causes such as test-blocking defects, extent of new code added during a typical period, and change in testing hours due to change in resources. Any steep fluctuation must be evaluated and a corrective action must be taken to address it as a special cause.

### Find and Fix Rate:

Although the Find Rate plays a critical role in this metric, it only indicates the level of quality at a given time and fixing the defects actually enhances the quality. Of course, knowing the shortcoming of the product increases the chances of better risk mitigation. Fixing the defects is essential not to just enhance the quality but also reduce the support cost.

The following graph shows Fix Rate in combination with the Find Rate.

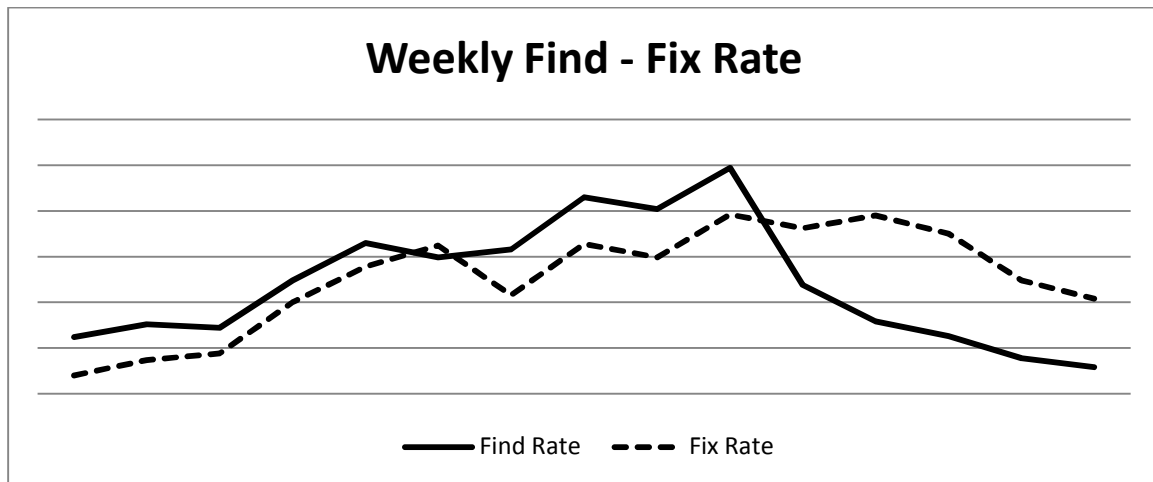


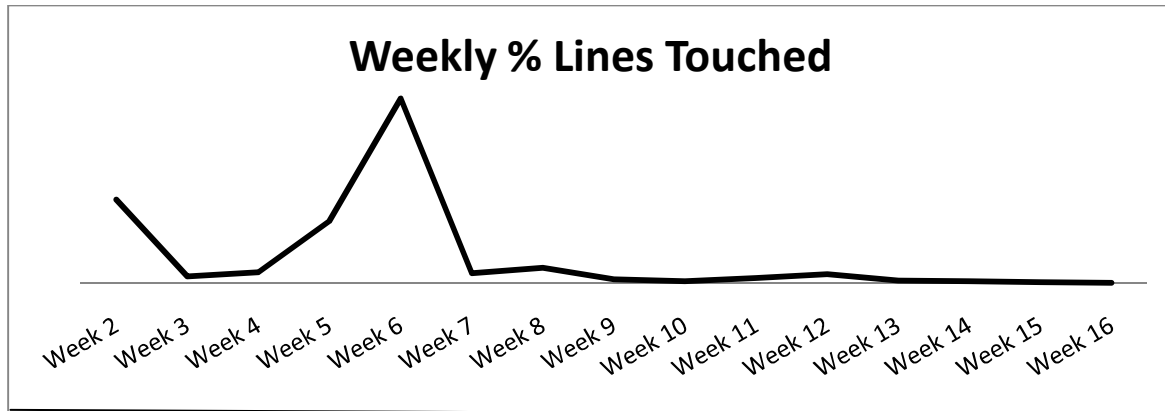
Figure 2: Find and Fixed Rates in a Test Cycle

In the beginning the Fix Rate trails the Find Rate. Once all the new functionality has been developed, the development efforts shifts towards defect-fixing resulting into a higher Fix Rate. At some point the Fix Rate surpasses the Find Rate. That is when one can mathematically compute the time required to fix all the known defects and the product quality at release.

Like the defect Find Rate, the defect Fixed Rate may be inflicted due to late feature creep resulting in new code. It is important that such behaviors are understood by the program management team.

### Code Volatility aka Code Turmoil – the mother of all trends:

Any defect patterns that are used to make release decisions are inherently related to code volatility (sometimes also referred to as code turmoil). This is the number of lines of code that changes from one drop to another. After all, the amount of change in the code directly impacts the number of potential defects. The code volatility can be measured by accounting for new lines of code, modified lines of code and the deleted lines of code. A large number of deleted lines of code resulting due to removing a feature or re-structuring code has a small potential for introducing new defects. One needs to be diligent when inferring the number of deleted lines of code.



**Figure 3: Code Volatility Trend in a Test Cycle**

Once again, this trend provides a cross reference to the Defect Find Rate. As one would expect, the code volatility will be high during the development of the new functionality and will decrease as the development moves from the new functionality to the regression phase.

Although not commonly explored it is possible to correlate the number of defects to the code volatility as the development proceeds. Once established for a stable environment, it is entirely possible to predict the number of defects that are to be expected in the next code drop. This has the potential to evaluate the following facts while testing a new code drop:

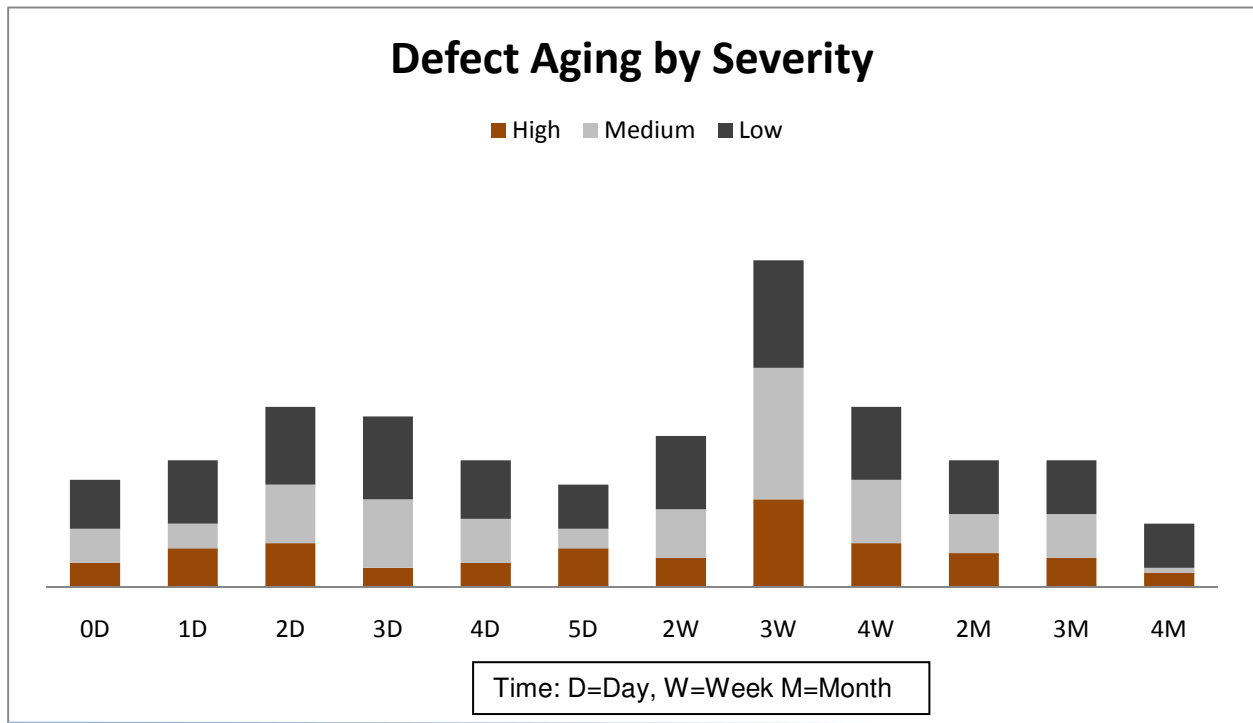
- The number of defects anticipated
- The amount of efforts required to test
- Testing effectiveness measured by the number of anticipated versus actual defects

In some organizations a maximum allowable code volatility is imposed to keep the number of new defects to a manageable level. This becomes increasingly relevant as the release date approaches.

## Defect Resolution for Scheduled Delivery and Quality

In order to release an intended quality product on time, it is important that the defects found in testing are resolved and verified in a timely manner. Defect resolution may not always require fixing a defect and a business reason should drive a fix. If a defect fix does not support a legitimate business reason, it does not have to be fixed at the cost of slipping schedule.

The defect aging metric helps provide an understanding of the defects that have not been worked upon for a longer than the expected period. Figure 4 shows how the defects with varying priority have been delayed for resolution.



**Figure 4: Open Defects Aging Chart**

The graph shows how the open defects accumulate over time for each severity. In this example a significant number of defects are 3 weeks old. Letting the defects age makes it more difficult to fix as characterization becomes harder to reproduce. The defect verification activity also suffers with the same dilemma although not to the same extent since the test cases have been documented along with the expected results. An early decision on which defects should be fixed for the current release is beneficial to avoid spending longer than usual time on these defects.

## Proactive QA Approach

The quote “Prevention is better than cure”, applies well in the software quality arena. The QA group has an important role to play by means of work product reviews in the early phases of product development. Once the product has entered the validation phase the QA group can proactively facilitate a better quality product by taking some specific measures both prior to and after the product enters the development phase. Some of these measures include:

- Get familiarized with the product functionality – this can be achieved via reviews and working closely with the development teams on test development. Requesting an early drop of the code into QA is an excellent way to get familiar with the environment, installation and setup, and product workflow.
- Understand the high risk functional components from the development perspective and ensure that test case design and execution effort is appropriate to the risk.
- In the situation where test coverage is not on schedule understand the factors such as “test blocking defects” and “incomplete functionality” and coordinate that with the development group.
- Ensure that adequate resources are made available for testing. An understanding of testing capacity always helps to establish the need for testing resources.
- Provide as much assistance as possible to the development group with the defect root cause analysis by a clear defect description, providing screen captures and logs, and being available to answer questions.

- Apply alternatives to structured testing approaches when the test schedule is at risk. Testing is always on critical path and any proven test methods such as Exploratory Testing [Bach] will help the overall test schedule.
- Last, but not least, foster the “team approach with the development” and not “us vs. them”.

## **Conclusion:**

The goal of a quality organization is to support the delivery of a high quality product as defined in the release criteria. This can be better achieved by measuring the progress towards the goal along the product development. Gut feels and hunches do not lead to educated decision making and an organization using the right metrics will be in a better position to answer the question “Are We Ready To Release?”. The author has extensively used these techniques and benefitted with their usage. A QA team should emphasize prevention over cure but cure is an important complement of prevention.

## **Acknowledgements:**

The majority of the experience the author has shared in this paper comes from his thirteen year tenure at HP. The author has maintained a general approach and has especially refrained from sharing the data in its entirety.

## **References:**

1. Rothman, Johanna; Release Criteria: Is this Software Done?, <http://www.jrothman.com/Papers/releasecriteria.html>
2. Simmons, Erik; When Will We Be Done Testing?; Software Defect Arrival Modeling Using the Weibull Distribution; 18<sup>th</sup> Annual Pacific Northwest Software Quality Conference, Portland, Oregon, October 2000
3. Gupta, Bhushan B.; Guiding Software Quality by Forecasting Defects using Defect Density; Pacific Northwest Software Quality Conference, Portland, Oregon, October 2004
4. Bach, James ; Exploratory Testing Explained, <http://www.satisfice.com/articles/et-article.pdf>