

# Avoiding the “Test and Test Again Syndrome”

## A Lesson From The School of Hard Knocks

**Douglas Hoffman**

BACS, MSEE, MBA, ASQ-CQMgr, ASQ-CSQE, ASQ Fellow  
QA Program Manager<sup>1</sup>  
Hewlett-Packard  
408-285-2408 (W)  
408-741-4830 (H)  
douglas.hoffman@HP.com

### Abstract

I’ve heard that a frog won’t jump out of boiling water if the water is slowly heated from room temperature to boiling and the frog was placed in it before heat is applied (and I tend to believe it without needing to test it for myself). It seems that a frog does not react to slow changes in temperature, even when its life is threatened. Over the span of my career, I’ve worked with projects where the test team seemed to wake up one day to realize that it was time for the final testing push, but they’ve been so busy running tests that they haven’t had time to prepare properly. The test team had become embroiled in what I call the “Test and Test Again Syndrome” (the Syndrome).

Over the years I have worked through the issues with varying degrees of success. To successfully deal with such a situation, several questions need answers: What are some of the forces behind the Syndrome? What does it cost us? How can testers successfully deal with the Syndrome? How might it be avoided? What are some of the approaches that have failed to deal with it? The session delves into answering these questions based on some of the lessons learned through the school of hard knocks.

In commercial software development, test execution usually begins on new software while it is still being written; before it is complete. This happens for a variety of good reasons. This early testing can be difficult and time consuming because not everything works and the test team must differentiate between “it’s not ready” and “it’s broken.” These early investigations can be extremely valuable, but a lot of extra work can go into investigating and documenting the software characteristics. In the worst-case scenario, the early tests are all that ever get run and developers train the code to pass the tests; potentially leading to very defective code.

Once testing has begun there may be significant pressure from many sources to continue testing. Some of the pressure comes from management, developers, and frequently from the test team itself. Sometimes, this early testing can lead to situations where a test team gets into a Test and Test Again Syndrome. This is a situation where testers and developers begin a dance where the developers provide product for test, the test team explores and documents their findings, the

---

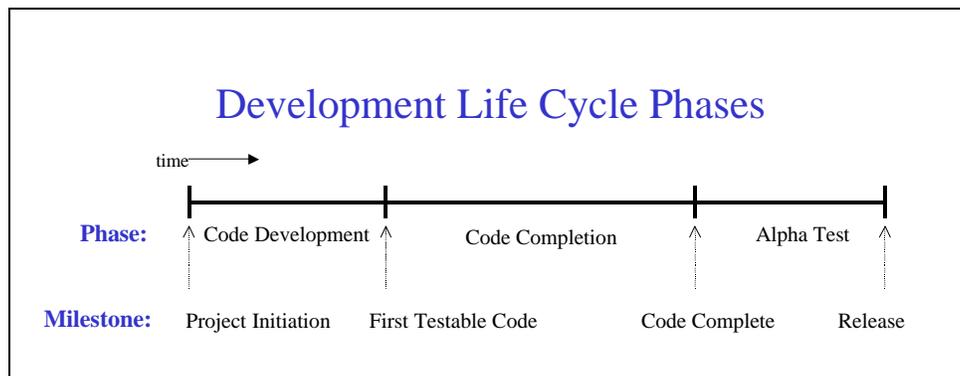
<sup>1</sup> The experience and opinions described in this paper are based on work completed prior to Douglas’ joining Hewlett-Packard and are in not representative of Hewlett-Packard’s organization, processes, or opinions.

developers fix the problems and release a new version for testing, and the test team then reruns and documents their new findings. This pattern cycles over and over, with the product slowly gaining features and stability toward its eventual release. The test team may dedicate all of its effort toward running tests, reviewing results, and reporting their findings. Although these are required activities for a successful test team, they are not the only ones. The test team in this situation is often working full tilt; running tests, investigating and isolating faults, writing up defects, rerunning the tests, and verifying fixes. As time goes on, the product evolves one step at a time until finally it is complete. The test team is all the while trying to do the testing while learning about the upcoming features and planning for future tests.

As a consultant I have worked with many test organizations and have encountered the Syndrome repeatedly. The Syndrome was normal for many of those test organizations. I see sub-optimal use of resources when the test team is so busy running tests that other aspects of analysis and planning are squeezed out. This paper describes many of the aspects of the Syndrome and what we, as testers and managers, can do to recognize and improve the situation.

## Introduction by Example

In one typical case, I was brought in to manage the QA function for a startup developing a commercial software application. Versions of the product had been deployed at several customers for evaluation, and the company was bringing together a more complete feature set for the expected release 1.0. New features had been under development for several months and the project was in the development Alpha Test Phase (See Figure 1).



**Figure 1: Life Cycle at Example Company**

I noticed some things that concerned me:

- All the test team’s time was spent doing testing and reporting problems
- There was no time for in-depth analysis, good planning, or creating new tests
- There were lots of open problems and unverified fixes
- The project was getting close to the final test cycle for release
- The test team was not ready to do final acceptance testing
  - They were behind in analysis, design, and implementation of tests

- They were especially not prepared for the latest features introduced
- They had no overall plan for the final test cycle

I also observed:

- The test team was working hard and long hours testing and reporting their results
- Daily versions were released to testing by the developers
- The test team was working very closely with developers
- The code was really defective
- The test team reported lots of problems
- Developers were very responsive to fixing the problems
- Many of the reported problems were already known by development

Independent of the testing itself, I had serious doubts about the software being ready for release. However, my biggest concern was about the test team. They were not going to be ready if good software was delivered on schedule. Defective code and missed code completion dates should not leave a test team unprepared. It was their choice of priorities that led to the lack of preparation time.

On this project, the test team started testing as soon as the code compiled. This is not necessarily a bad thing, and should probably be part of the plan, as James Bach and I described at last years’ CAST. As in this instance, though, the test team may become mired in testing, reporting, and retesting to the exclusion of everything else. In this organization they became too seriously involved in testing cycles too early in the development cycle, hence leading to the Syndrome.

### **The Test And Test Again Syndrome**

- A test team gets an early product version and begins testing it
- They report the problems they find
- Before they are through testing, a new version is available
- Testing begins again on the new version
- Some fixes are verified and new problems are found and reported
- Before remaining tests can be run, another new version is available
- Testing and retesting of new versions continues in a cycle that has a growing number of open problems and unverified fixes
  
- There isn’t time for more test planning or design
- Often, more and more effort goes into fix verification and less and less into looking for new defects
- The test team is ultimately not prepared for final testing

### **The Dilemma Leading to Test and Test Again**

Prior to complete development, parts of the new software are capable of being tested incrementally in the periodic development builds. These early software builds usually occur

faster than complete test cycles can be run. They also start to be available long before all the new tests have been designed or implemented. Once the test team begins testing, whenever a new software version containing fixes and partially completed new functions becomes available they have to make the choice between finishing with testing the current version or installing and testing the new version.

Continuing testing allows completing all planned tests and fix verification. However, it puts testing out of sync with the developers because developers are running on more recent versions. This means communication of questions and potential problems is more difficult. The test team may report problems that developers may no longer care about because the developers may have already fixed the problem in a newer version. Developers may not accept what is found as relevant because of the possibility that it may no longer be a problem. The test team may need to test for the problem in the latest build before the developers accept it.

Testing the new version keeps the test team synchronized with developers. However, it incurs the overhead of reinstalling the software and setting the tests up again. It cuts off testing before all the tests are run, so testing doesn't provide all the planned information. It also restarts the testing cycle, causing repeated running of some of the same tests and possibly repeated verification of fixes to confirm they remain fixed.

An endless test cycle can easily develop if the teams are not careful about testing of the early builds. Early development builds provide ample opportunity for defects to be found and reported. An expectation that the test team will run complete test cycles, or even complete defect fix verification, frequently leads to the Syndrome. Early testing is inefficient while the test team learns about the product and it has missing and broken capabilities. Early testing is superficial due to time and product capability constraints. In order to provide any indication of product quality at release, the early testing must be repeated after the product is completed.

### Some Advantages of Testing Early

1. When defects are found soon after being introduced, the cost to fix them is minimized
2. Early testing can provide insights into product strengths and weaknesses that are valuable for both development and the test team
3. The knowledge of product characteristics gained through using the product is useful for planning and creating tests
4. Early exploratory testing can maximize the time spent performing tests to learn about product quality

### Some Advantages of Waiting to Test

1. Some potentially high costs to the test team for early testing can be saved by waiting:
  - The overhead of setting up new versions of software
  - The time spent tracking down what turn out to be known deficiencies
  - Time spent doing fix verification and retesting are increased by fresh, incomplete code
  - Intermediate development versions are inherently unstable and difficult to work with

2. More useful information about product quality is usually derived from testing completed products than is gained by testing incomplete products
3. Waiting provides time to plan, design, and create excellent testware in advance of test execution, which increases the value of the information gained through testing

The problems created by the Syndrome may best be seen by clarifying the role of testing in the organization. Quality cannot be put into software through testing since testing can only identify some portion of defects already built into the product. Testing by itself does not change the software; it only changes what we know about it (which is very powerful information). Although the test team’s feedback about defects and ongoing measures of project process are very important during development, these contributions are not unique. All early users of the product can provide feedback (e.g., developers, technical writers, trainers, technical support specialists, etc.). The test team’s unique (and therefore most valuable) contribution is independent, unbiased analysis and reporting about the quality of the final release candidate. No other group can provide that information. The information gathered through testing usually forms the technical basis for release decisions. And, the value of that information depends on the focus and quality of the tests run by the test team on the final product.

A key concept in the commercial software development context is that the test team must qualify the release version. The software has to work for the users, and that leaves a burden on the test team to look hard for defects in the specific code that ships. New versions are different from the previous ones, and even tiny changes in software can have wild, non-intuitive consequences. Test results from earlier versions are not guaranteed to be reliable results for the updated version. If the test team could only test once, it would have to be on the version released.

In the commercial software context, the test team must be prepared to test the last release candidate quickly and effectively in order to provide a thorough evaluation in a reasonable time. Test planning is critical for all aspects of this tall order. Time must be invested in understanding of the testing context and risks to prioritize their work. The test team needs powerful tests, which requires in-depth analysis of the product and time to design and implement tests and test data. These will not be achieved if the time planned for them is used instead for running and rerunning tests.

## The Costs of Test and Test Again

What could be wrong with a test team doing testing? Testing provides critical information about quality so developers and management can make good decisions and produce good quality software. However, there is a matter of test team efficiency and effectiveness. In commercial software testing contexts, providing less information at a higher cost than necessary is inefficient, and providing less valuable information for the same cost is ineffective. If the goal is to find defects or measure quality, the test team must plan to provide the most valuable information for their efforts. That means making efficient use of their time while running effective tests.

At one extreme, a test team might devote all their efforts to test execution activities. At the other extreme, a test team could spend all their time planning tests and creating test infrastructure. The

first case is not likely to be effective at designing complex test cases to test against specific risks. The second case is not likely to be effective at gaining a broad picture of the product quality. A good test team must spend sufficient time working with the product to get the breadth and depth they need for test design, data generation, and test preparation. They must also spend sufficient time preparing tests so they can understand and effectively test the most important aspects of the product.

Getting into the Syndrome is seductively easy. It usually begins innocently with a little testing to see what the product is going to look like. Some defects are found and reported, and the team retests when developers deliver a version with fixes. The cycle time between development versions is often very short. The test team verifies fixes and reports a new batch of defects. The situation spirals downward because, as product functionality grows, the amount of testing also grows. Likewise, there are increasing numbers of defect fixes to verify as time goes on.

When the test team gets bogged down in the Syndrome they become locked in perpetual testing mode to the exclusion of everything else. The problem this situation causes usually goes unrecognized. The test team is obviously busy testing and working with the defects. People rationalize that this is what the team is supposed to be doing. However, testing is not efficient because of the amount of repeated work, and testing is not effective at delving deeply into the product due to lack of time. The usual time for planning and development of effective tests is consumed by repeated (redundant) retesting of new versions of the product. The large number of defects encountered takes substantial amounts of time to investigate, report, and verify after they are fixed. This is not the best use of the team’s time.

Early testing can be both inefficient and ineffective. It can incur high costs for the test team, including:

- Wasted tester and developer time analyzing, isolating, and reporting of defects and incomplete code when the developers already know about the problems
- Very little incremental useful information from rerunning tests that have already passed and retesting of code that has already been tested
- Superficial testing due to unfamiliarity with the product and lack of opportunity to concentrate on any area in depth
- Wasted time doing retesting and fix validation of defects that otherwise would have been found and fixed by developers before code completion anyway
- Product functions tested before completion need to be retested when they’re completed in order to have confidence in their quality
- Programs are guaranteed to fail some tests before they are completed. Effort spent on such false alarms provides no useful information about product quality
- Even the time spent doing the test that generated a false alarm is wasted because they must be investigated and some, if not all, retested later
- Lost opportunity for analysis, planning, and preparation for testing
- Exhausted testers

## Some Underlying Forces Behind Test and Test Again

Managers and others within the project team may hold the view that the job of Software QA/Testing team is simply to run tests. In extreme cases the test team is perceived to be unproductive if they aren't running tests and reporting problems. Successful developers don't just write code, they must also understand requirements, architect, design, document, and review their work. The same applies to the test team.

There are several other factors in some organizations that tend to bias testers toward running tests to the exclusion of these other tasks:

- Testers like to run tests because testing and finding defects is fun. It provides immediate gratification in getting the test results and discovering defects. (In my experience, most novices tend to avoid planning, analysis, design, and reporting if they can.)
- Using a product is one obvious way to learn about it
- The test team's time and contribution is often perceived as being less valuable than development's time. (It isn't more or less valuable – the roles are different but both are necessary.)
- The test team frequently is doing part of development's product debugging.<sup>2</sup> (Developers must take responsibility for creating and delivering good quality to begin with.<sup>3</sup>)
- There is often the expectation that the product is close to ready once testing begins

## Successfully Dealing with Test and Test Again

Since recognizing the problems due to the Syndrome, I have learned many ways to deal with it. I think the best way is to avoid getting into testing cycles too early, but that isn't always an available choice. Either way, I work with development and the test team to minimize the Syndrome's effects.

### **Plan to Avoid Beginning the Test and Test Again Syndrome**

Having a clear understanding of testing's role in the organization is a huge step toward avoiding the Syndrome. As described above, there is real value in the test team using products during development. Early feedback from testing is valuable to the developers, but early testing's real value for the test team comes from their increased understanding of the product.

The test team needs to shy away from too much early testing so they have time to engineer excellent tests and testware. Some early testing can be safely achieved when the test team focuses on learning about the product and they consider the defect discoveries as bonus byproducts. Once the test team has learned enough to design excellent tests for a part of the product, they should move on to other tasks. The test team should do the early testing for their

---

<sup>2</sup> *Debug. To detect, locate, and correct faults in a computer program.* IEEE Std. 6110.12-1990(R2002), page 21.

<sup>3</sup> Debugging is only necessary because errors have been introduced before code is completed. Debugging by the test team should be avoided unless their charter includes alpha testing or they are recognized as “sub-contractors” for that part of development's job.

convenience, not as a necessary duty. They must also keep in mind that testing incomplete code is not efficient and will have to be redone when the product is completed.

The plan for testing needs to be clear from the start (e.g., schedule, resources, strategy, tools, priorities, etc.). It should allow the test team sufficient time for planning and preparation before formally beginning test cycles. The planning and preparation time will overlap development of the product and should include some use of system resources for early experience with partially completed software.

One way to communicate a plan to allow sufficient test preparation time is to officially begin testing only after development claims the product is complete. This is usually a major project milestone (typically called Alpha Test), so it is easy to identify in the development schedule. The initial delivery should be the first release candidate – complete and adequately tested so there is some chance it could be released (although this has yet to happen in my experience). Prior to that time there is no chance that this will be the last test cycle and the testing is redundant. Therefore, testing will need to be run again on the final release candidate if testing is going to qualify it. The beginning of testing should be decided by the test team by running a subset of their tests to determine that the product is complete enough and acceptable for testing.

### **Recognizing the Test and Test Again Syndrome**

Recognizing that the test team is in the Syndrome is the first step toward correcting it. As discussed above, some of the observable behaviors of the Syndrome are:

- The test team is testing full time
- The code is not yet complete
- The test team does not have time to analyze, design, or implement tests for new functions
- There are many problems being found and reported in the new code
- Fix verifications are consuming significant amounts of the test team’s time
- Many problems found by the test team are already known or in unfinished code

### **Breaking Out of a Test and Test Again Syndrome**

There are several alternatives when the test team (or any team member) is caught up in the Syndrome. Generally, the test team gets out by interrupting the cycle and going back to the plan. (If not the original plan, the best alternative moving forward.) Not all of the methods I’ve seen have worked, however. Below are three methods that have been successfully used, followed by three methods that I have not found successful.

### **Some Successful Approaches to Breaking Out of Test and Test Again**

The successful approaches to breaking out of the Syndrome require the test team to step away from test execution to a large degree. The first two are most useful when the major difficulty is too frequent deliveries of new versions. The third approach also reduces the testing required for each revised version. There are risks and potential drawbacks for each, but reducing the impact of the Syndrome more than makes up for them.

## Don't Accept Intermediate Versions

The overhead cost of setting the test environments and the restarting and repeating of tests has a high impact. When frequent deliveries of new versions keep the test team from making progress toward completing even one pass through the planned tests and the team thinks it's necessary to continue testing, consider ignoring some of the development versions. Plan a schedule for delivery of versions that will allow completion of planned tests. This also may give development more opportunity to stabilize the product and provide better quality in the versions they deliver.

Some care should be taken with this approach. Where too frequent deliveries creates too much overhead setting it up and retesting, longer times between deliveries generates long feedback loops between the test team and development. The longer the product remains in development, the larger the differences between versions being tested. The longer the time between defect creation and defect discovery, the more difficulty development has dealing with them.

This approach also gives development time to do interesting new things, stabilize, and gain confidence in product readiness between versions the test team accepts. While the product is still being created, the characteristics of sequential builds can fluctuate wildly. The test team is more efficient when testing stable, good quality versions instead of unstable, defective ones.

It also limits the number of different versions having problems reported against them. This is much easier on everyone, since fewer versions of the source code and executables need to be kept for defect isolation and fixes.

## Only Test Completed Functions

Another way of dealing with too frequent version deliveries is to only test those parts of the version that are thought to be complete. Until a feature or component is complete, it's broken and some tests will fail. Those failures that are due to known deficiencies, and these defects can be time consuming for both the test and development teams, ultimately providing almost no benefit. Testing prior to code completion is, in reality, part of debugging, and prior to code completion debugging should be a development function.

Good developers will test their work when they think code is complete. However, many developers do little or no testing prior to code completion because they know tests will fail on incomplete code. When the test team waits until after the developers have completed their testing there are fewer defects to find and fewer false alarms.

## Postpone Verifying Fixes Until Late in Testing

In a Test and Test Again situation the number of defects often grows out of hand. The number of fixes in a version can become large and the task of verifying them overwhelming for the test team. The question becomes whether to verify the fixes immediately, later, or not at all.

I have differing thoughts about when to retest fixes. On one hand, the sooner a fix is checked, the easier it is to remember how to test for it. When the fix doesn't work, providing immediate feedback to development significantly increases the likelihood and their ability to fix it correctly. The code involved and their thought process about fixing the problem are still fresh in their memory. These problems are also relatively inexpensive to test for, since they have been encountered and documented before.

On the other hand, if development properly fixed the defect, retesting only confirms that fact. Rerunning the test rarely finds another, different problem. If development requires testing to know if a problem is fixed, they really have a different issue because they apparently don't know what the problem is. Tests are most powerful the first time they're run; i.e., they are more likely to find a new problem the first time they are run than after that. Code is not likely to wear out, so running the same test repeatedly has diminishing chances of finding the problem it was designed for. (It's not that they won't find any problems; only that the likelihood of finding a problem diminishes the more times the test passes. The test team's time may be better spent finding new defects rather than retesting parts of the product with tests that aren't likely to find new defects.

Yet, the project (and user) takes the risk of having the defect remain if the test team does not verify fixes at all. This is a high risk, as some non-trivial percentage of fixes either don't work or introduce new defects. (My experience has run from around 20% to 110% - 20 defects per 100 fixes to 110 defects per 100 fixes.) Skipping all fix verification is not recommended.

Ultimately, the test team should choose when to verify fixes. When the amount of time spent verifying fixes gets in the way of other critical activities, I think verifications should be postponed and the time invested in new and better tests instead. The test team should re-verify all fixed problems in the release version anyway to assure that the fixes did not regress during development.

### **Some Unsuccessful Approaches to Dealing with Test and Test Again**

In my career I have witnessed some failed attempts to deal with the Syndrome. The test team is bogged down, is not going to be prepared, is overwhelmed with defects, or foresees schedule slips. I consider the approaches failures because the testing and the product were not improved. Below are three approaches that I think are worth avoiding. (There is a common theme in these failed approaches; they are attempts to deal with some of the symptoms without addressing the root causes.)

#### **Asking for More Resources**

Asking for more resources is a lose-lose approach. Adding resources to the test team does not address the underlying problems. The causes for the Syndrome are likely to continue to spiral and consume any added resources. Testing can always consume the available resources without making a dent in the number of valid, useful tests that might be created and run. If resources are not available the test team has the same problems and, at the least, has admitted their inability to work to their plan.

## Controlling by Fiat

Trying to control the release by fiat has not worked. This is a failing approach even when the test team has the power to direct changes and control the project. The test team is part of the overall project team, but they aren't the project planners. For example, changing development priorities, changing schedules, reallocating resources, or redefining the contents of a release does not address the causes or problems of the Syndrome. These types of changes are highly disruptive to the organization and do not endear the test team to the other people they have to work with.

## Rigid Enforcement of Process

Focusing on doing things “by the book” does not improve the situation, either. Requiring exact adherence to the methodologies, standards, reviews, etc. stifles communication and usually causes additional rework and delays. This is especially difficult when the process has been loosely enforced historically, when schedules are particularly tight, or when quality is poor. This approach often causes the opposite effect without addressing the underlying cause. Greater flexibility can be helpful when problems surface from the Syndrome.

## Summary

The Test and Test Again Syndrome is a situation where the test team becomes prematurely dedicated to full time testing of an incomplete product. A cycle of Test and Test Again is costly to the test team and the project. Frequently the cost comes in the form of test team inefficiency and ineffectiveness, both of which are nearly invisible. Although the test team is working full tilt, their testing is inefficient and less effective than it should be. The ultimate result is less thorough final testing and a potential for more product defects.

There are several ways to deal with the Syndrome and avoid its potential costs. It is possible to avoid the Syndrome through test strategy planning, including: resisting deep involvement in early testing of development versions, having a late “official start” of testing. Watch out for the Syndrome and break the cycle if you see it happening. Some successful techniques for breaking the cycle are testing only selected versions from development, testing only completed functions, and delaying fix verifications. Try to avoid trying to break the cycle by asking for more resources, controlling by fiat, or rigidly enforcing processes, since these have tended not to work.

## Experience and qualifications:

Douglas Hoffman has over thirty years experience in software quality assurance and has earned degrees in Computer Science, Electrical Engineering, and an MBA. He is currently employed by Hewlett-Packard as a QA Program Manager. He is a Founding Member and a past Director of the Association for Software Testing. He has been a participant at dozens of software quality conferences and Program Chairman for several international conferences on software quality. He was among the first to earn a Certificate from ASQ in Software Quality Engineering (ASQ-CSQE), has been certified in quality management (ASQ-CQMgr), and is an ASQ Fellow. He is

active as a Fellow of the ASQ, participating in the Silicon Valley Section, Software Division, and the Software Quality Task Group (SSQA), and is also a member of the ACM and IEEE. He is current Auditor and Past Chairman of the SSQA and is the Immediate Past Chairman of the Silicon Valley Section of the ASQ.