

Combating the Test Schedule Killers

David Petrella, Sr. Project Manager, SysTest Labs, LLC
216 16th Street, Suite1380, Denver, CO 80202
303-575-6881, 303-575-6882(fax)
dpetrella@systest.com

This information will be presented at StarWest 2003.

Tired of ‘stressing-out’ trying to fit adequate testing into unrealistic timelines? We all know that most project schedules skimp on testing. By implementing one new and two not frequently enforced test practices, you can end long hours and be confident your testing will be adequate for your customer.

I have been testing software in various work environments for quite a long time. I have seen and been through extreme test cycles that worked my teams and myself ragged. There are several things that can affect and bog down a test cycle, but on the projects I have worked there seem to be just a few that are very common. These are: Not Enough Test Schedule, Test Environment Not Ready, and Not Enough Time for Regression.

Not Enough Test Schedule

There are many factors that result in an inadequate schedule for testing. These can range from project managers dictating the test schedule to Marketing needing (wanting) to have the product launched by a given date. And you know development won’t reduce their schedule. Sometimes, management tries to trick you and asks for your input.

Here is the typical scenario. You are assigned to a major software development project. You assess the concept documentation and in your professional view you give an estimate of six weeks of testing. Being a little inexperienced as well as honest, you do not pad your estimate. After the project manager and whomever else they conspire with gets hold of it, they tell you that now you only have 3 weeks to complete your testing. Oh, and by the way, you need to do it with existing resources. You begin to stress out, because you truly felt that it would take six weeks. Now you will need to work mega-overtime or cut out some necessary testing.

Test environment not ready for test to begin

This is a common slayer of the testing schedule as well. The testing environment is made up of several pieces. These include the physical hardware, the software, data, interfaces, tools, other supporting or downstream systems, as well as test case procedures and scripts. Many of these environment artifacts are out of the testing group’s control. Obviously Development owns the software. Often Development or a ‘Systems’ group controls the physical hardware. Data is administered by the DBAs or the business, and other groups control supporting systems. Typically it is the software that is not ready.

Scenario Number Two: It is Friday and testing is scheduled to start on Monday. You have worked diligently for weeks getting your test cases ready. You have worked closely with the DBAs to ensure that you have the needed data from the business and the project manager pulled some strings and you have your own dedicated test system. Development reports that they are 99% complete with the code, but still have some minor tweaks. Being a team player, you agree to give them a few days of your test schedule to complete the code. You think, since everything else is in place, giving Development a couple more days is cool if you are going to get a final product instead of pieces. Tuesday comes and goes and still no code. Development says that they will give you what is complete to start testing and give you the rest by the end of the week. You begin testing what you have by starting the application and logging in. You click the button on the web home page only to get the dreaded “404 – page not found” error. You quickly learn that no one in Development even smoke tested the code. You now have just over half the time you originally scheduled for testing. You begin to stress out.

Not Enough Time for Regression.

The last schedule killer is one that afflicts us due to bad code or good intentions. Delays in testing due to bad code are obvious. There are constant fixes and rebuilds being made to correct defective code. These seem to continue endlessly. Other factors that cause delays are more sincere in their origins. The project team, including test, tries to be accommodating to the business. The team allows additional desired functionality to be added to the delivery at the last minute. Or, the test team agrees to continuous bug fixes and re-builds right up until rollout. All of these causes, result in little or no time to perform adequate regression testing of the application.

Nice Tester scenario: You are one week into a three-week test window. Testing has been mediocre at best. You have been informed that Development has agreed to incorporate some minor changes. You acquiesce because a major client requested it. This adds a couple more tests to your suite. As the second week ends, the new code is complete and you are about 80 percent complete with all testing. The defect rate is going down, but there are still new defects creeping up each day. You begin to doubt the quality of code, but the developers insist everything is fine and bug repairs are ‘just a one line change.’ As the final week progresses, those ‘one line changes’ turn into 4 days of repairs. You think that you will probably have time to re-test the bug fixes and some of the major features, but only enough time to regression about 50 percent of the code. You begin to stress out.

Combating these common Killers

Over the years, I have learned to use three simple tools to defeat these schedule killers.

- Not enough time for testing – Implement the Scope and Risk Assessment
- Test environment not ready – Use the traditional Test Phase Entrance Criteria
- Not enough time for regression – Enforce a Code Freeze

The Scope and Risk Assessment

In our first scenario, the test estimate was slashed in half. Someone who doesn’t know or understand testing likely did this. They probably do not know why you needed so much

time to perform testing. Using a Scope and Risk Assessment, you may just get some of that time back. This tool is not only used when schedule is an issue, it can be implemented when other resources are inadequate as well.

#1 Scope and Risk Assessment

- Introduced during Planning Phase
- Provide Scope of Testing
- What Can and Can't be Tested
- Give Recommendations
- Get Concurrence



Introduced during the Planning Phase (as early as possible), the Assessment is targeted to project management and more specifically the customer or client. It provides the reader with the background information they need to make a valid determination of the need for testing.

The Scope and Risk Assessment begins by briefly describing the Scope of the effort (see sample below). The system being developed and tested is overviewed first. This overview describes the nature of the system, software, and any history the reader may need to better understand the project. This is followed by an overview of the scope of testing. This should provide 'what' will be tested and may include items such as Applications, Functions, the Types of Tests, etc.

A Risk Assessment then follows the Scope. The Risk Assessment section explains what testing will be *limited* or *not performed* and what the associated risk this will have on the project. The reason a test is being omitted or limited should accompany each item. The omission or limitation may be due to any restrictions imposed on test. This may include inadequate or reduced schedule, lack of resources including personnel or equipment, or any other valid reason. After the limitation, assessed risk, and reason are given, a solution should be presented. Your proposed solution may be more schedule time, more resources, or even outsourcing.

When implemented on a recent project, use of the Scope and Risk Assessment document resulted in two weeks being added to our test schedule.

Scope and Risk Assessment

Overview

This document

- Presents an overview of the application or system to be tested,
- Describes the scope of testing,
- Identifies what cannot be tested given existing project constraints,
- Lists the risk(s) and risk levels associated with not performing listed tests, and
- Explains why the listed test is limited or not performed.

Scope

System Overview

The Gas Tank project scheduled for Release 15. The Gas Tank Project will provide a visual display of data usage to the customer by displaying an easy to understand graphic representation. The Gas Tank functionality will be visible to the user.

Testing Overview

The System Level test effort will ensure accurate functionality of the web page as it relates to the Gas Tank project. All basic functionality as described in the preliminary project documentation will be verified. In addition, all interfaces will be verified.

What will NOT be tested

Please see the Risk Table below for those areas that System Test will **NOT** perform testing. An explanation and the potential risk to the customer or company are also given.

Risk Assessment

The risk assessment presented in the following tables was made after reviewing all available documentation describing the desired application, the proposed schedule, and available resources. This assessment places emphasis on areas of high risk to the customer or company.

Risk Level Descriptions

Risk Levels are defined in three categories as noted in the table below:

Risk	Description
High	A condition where risk is identified as having a high probability of occurrence and the consequence would affect program objectives, cost, and schedule. The probability of occurrence is high enough to require close control of all contributing factors, the establishment of risk milestones, and an acceptable fallback position.
Moderate	A condition where risk is identified as one that could possibly affect program objectives cost or schedule. The probability of occurrence is high enough to require close control of all contributing factors.
Low	A condition where risk is identified as having little or no effect or consequence on the program objective; the probability of occurrence is low enough to cause little or no concern.

Risk Table

The table below describes the types of testing or functional areas that **WILL BE LIMITED OR NOT PERFORMED** for the XXX project. It includes the potential risks, the risk level, an explanation 'why' this test is not being performed, and a potential solution to allow the test to be performed.

Testing Type, Area or Function	Potential Risk(s)	Risk Level	Explanation	Solution
Load/Performance	Poor system performance and poor response time	High	Hardware not available. Inadequate time allocated.	Need dedicated load test environment and an additional 2 weeks of test

Sample Scope and Risk Assessment Document

Entrance criteria

The next schedule killer – Test Environment not Ready - can be combated by implementing a method that has been around for years – the Test Entrance Criteria. Although this tool has been around a long time, it is often not implemented or adhered to. The Entrance Criteria document establishes when the Test ‘clock’ will start.

#2 Establish Entrance Criteria

- Introduced during Planning Phase
- Test Hardware
- Test Software
- Test Data
- Get Concurrence



The test environment is generally comprised of Test Hardware, Software, Data and Test Cases. If any of these items are not ready when testing is scheduled to begin, you will immediately be behind your original planned schedule. The first three of these are typically supplied by other organizations and out of your control.

Test Cases however are the responsibility of the Test organization and should be complete when testing is ready to begin.

Introduced after the test schedule has been established, the Entrance Criteria document states that testing will be impacted if the criteria is not met by the time testing is to begin. Entrance Criteria should be established around those areas out of your control – Software, Hardware, and Data.

Examples of Entrance Criteria

- 100% of base code complete
- Code is adequately controlled
- Component or Unit Testing completed and results reviewed by System Test
- System Test environment built-out and ready
- Project Data is available
- Supporting systems are ready and available
- Pre-Test build checkout by development

Software should be complete enough so that you can begin testing and also be assured that any delinquent code will be delivered soon. The code should be unit tested and controlled to reduce the chance of old code being re-introduced into testing. All hardware, interfaces and supporting systems should also be ready and available as well. Test Data is sometimes the most difficult to obtain, but is crucial to test success.

The final criterion that should be met is an application checkout by the Development group to ensure the software has been properly loaded and functioning as expected. In our second scenario, the surprise of discovering a login or initial failure could have been alleviated if a ‘pre-test’ checkout was executed.

If you are preparing a document, include a ‘Statement of Risk’. This statement gives the ramifications of what could happen if the criteria are not met.

Statement of Risk: “If the above criteria are not met in full, there will likely be risk to the quality of the product or a negative impact to the project schedule”

If at all possible, get the project team to accept the entrance criteria.

Testing Entrance Criteria Standard

For an application to be tested effectively and efficiently, several conditions must be met. If any of the following are omitted tests will likely need to be rerun or delayed until all are satisfied. The result to the project would be inadequate testing or delayed delivery of the application to the next phase of development.

Iteration

The following is a list of the criteria that should be met prior to testing a code "iteration" for the <PROJECT NAME> Project.

- 100% of iteration code complete.
- Code will be controlled via a CVS version control system.
- Component or Unit Testing completed and results reviewed by System Test.
- System Test environment built-out and ready.
- No Severity 1 defects against the project that affects the given iteration.
- Iteration deployed to System Test environment and has completed a Development Cycle 0 test.

Statement of Risk: If the above criteria are not met in full, there will likely be risk to the quality of the product or a negative impact to the project schedule (explain what the risk may be).

System

The following is a list of the criteria that should be met prior to a project entering or being considered in the "Certify or System Testing" Phase for the <PROJECT NAME> Project.

- 100% of base code (all iterations and interfaces) is complete. Only bug fix code from iterations may be pending completion.
- Code will be controlled via the CVS version control system.
- All iterative testing completed.
- System Test environment built-out and ready
- SystemA and Test Bed 2 systems are ready and available.
- Customer Account data is available.
- No Severity 1 defects against project.
- Build and deployment schedule established.
- Application deployed to System Test environment and has completed a Development Cycle 0 test.

Statement of Risk: If the above criteria are not met in full, there will likely be risk to the quality of the product or a negative impact to the project schedule (explain what the risk may be).

Entrance Criteria Template

The Code-Freeze

In our last scenario, our accommodating testers found themselves in a tight spot since they did not leave enough time for regression testing. This could have been easily avoided if they used the Code-Freeze.

#3 Establish a Code-Freeze

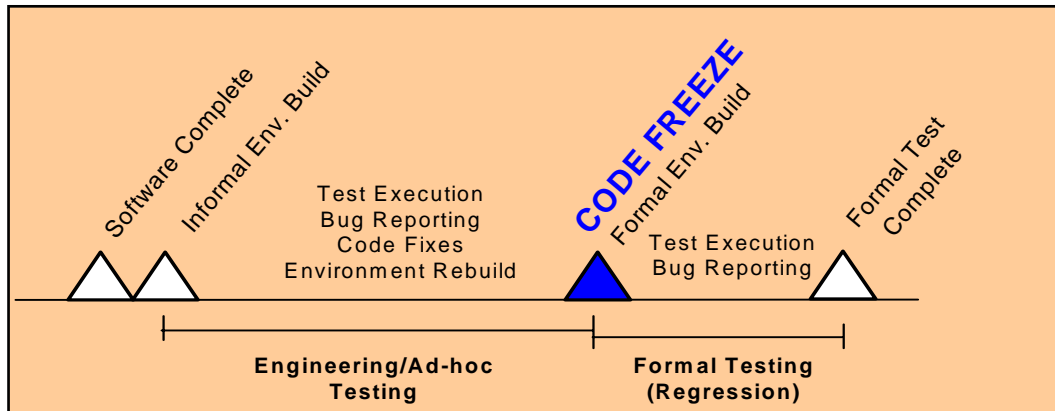
- Engineering or Ad-hoc Testing
 - Find Defects
 - Rebuild
 - Re-Test
- Establish Freeze
- Regression Test



The last tool for combating schedule killers is the Code-Freeze. Simply put, a Code-Freeze is the point where no changes in software can occur. The

code freeze ensures that there is adequate time to perform a regression test of the code or application. The code freeze is established during the Planning or scheduling phase but implemented during the Test Phase of the project.

During planning, a fixed 'Freeze' date is established as part of the Test cycle. This date should be just prior to the Test completion date, but leave enough time to run through all the test cases one more time. The figure below shows the testing timeline.



Once the software code is complete and handed over to the test team, an informal build is completed on the test hardware. This begins the ad-hoc, 'engineering', or informal testing stage. During this time, tests are executed, bugs are reported and fixed, and the code is rebuilt. This occurs repeatedly until the code-freeze date or until no more defects are discovered (whichever comes first). The build process may be a fixed cycle or just periodic as needed.

During this period, the number, frequency and severity of defects can be trended to establish or get a feel for the quality of the software. The project team can evaluate this condition and determine if the scheduled production date is adequate or should be delayed. This does not mean that there are no defects, just that defects are known and not severe enough to delay delivery.

On the 'code-freeze' date, the current version of the software is formally rebuilt in the test environment. This begins the final regression or 'formal' Testing stage. During this time, all test cases are rerun to ensure the code is free of new defects. If defects are discovered, they can be evaluated by the project team to determine if the application should go to production or be reworked.

Implementing one or all of these methods may be difficult in your situation. However, if you can implement even one, it will reduce your stress level and lead to better quality of your final product.

Biography

David is a software test professional with over 17 years of software industry experience, 15 of that specifically in testing and quality assurance. He is currently a Senior Project Manager for SysTest Labs in Denver, Colorado. David has managed teams or performed testing on a multitude of projects for various industries. He has also led test process improvement efforts at several companies. David has been a speaker at both PSTT and STAR conferences and published in the Journal of Software Test Professionals.