BIO

PAPER

T1

November 7, 2002
10:30 AM

Pressure-Cooker Testing: What to Do When the Squeeze is On

Geoff Horne
iSQA

International Conference On
Software Testing Analysis & Review
November 4-8, 2002
Anaheim, CA USA

Geoff Horne

Geoff Horne is the Managing Director of iSQA (Independent Software Quality Assurance) which is based in New Zealand and enjoys an international clientele. He has over 25 years experience in IT including software development, sales and marketing and IT and project management. Eight years ago, almost by accident, he found himself involved in testing a complex fault management system which led to further testing and QA assignments covering a wide range of applications and tools. Geoff then founded iSQA to bring a full range of testing consultancy services to the IT industry. With iSQA, Geoff specialises in testing strategies, methodologies, planning and project management, taking on selected assignments personally. He has also written a variety of white papers on the subject of software testing and has been a regular speaker at the Star conferences. Geoff is married with four children and in his spare time (which there is not a lot of) enjoys writing and recording original contemporary Christian music.

Pressure Cooker Testing: What To Do When the Squeeze Is On

© August 2002
Geoff Horne
Managing Director
iSQA
geoffh@isqa.com

Overview

This document looks at the issues that can arise and suggests solutions when a testing project becomes constrained by deadlines. Whilst this is true of most testing projects, there are some with farther reaching consequences than others especially if the deadline is immovable.

I hope to impart a few key points along with a few tips and techniques on what to do when you find yourself in this situation. This list is not exhaustive and the ideas presented here may need to be adapted to your own specific project. However when the chips are down and you need to bring in some semblance of software that is tested, these ideas may be a good starting point.

This paper is intended for Testing Managers but the points and tips contained within it will be of use to anyone one involved in planning for a deadline fixed project.

Precept

In an ideal world all IT projects would be carefully planned well ahead of time with adequate resources (time, people, money etc.) made available.

The more typical story is that a project starts off with the best of intentions, finds itself in difficulty some ways down the track and then sees panic set in when the real size and scope is finally unearthed. Then its all hands to the pump, long days and nights, and it falls gasping over the line (after the line has been moved a number of times).

Curiously enough, I have found that testing projects or the testing phase of a larger IT project seem to suffer most from under-estimation both in scope and the resources required - the "oh did we actually need to budget for testing?" syndrome. As a general rule of thumb we usually recommend allowing between 30%-60% of the development budget for testing, more if the risks are greater.

Deadlines are fact of life, not just in IT. Without them no project could ever be properly controlled and the art of good project management would quickly disappear. However some deadlines cannot be moved and if you find yourself with one of these and in trouble, what can you do?

Let’s look first at the different types of deadline…

The most challenging deadline is, of course, one that cannot be moved due to an absolute or a force of nature eg. Y2K. Fortunately there are precious few of these.

Next is the project that is being forced by law or legislation eg. new tax laws, industry compliance legislation etc. - an ever-increasing occurrence in our continually changing governmental and financial systems.

Then there are those that are brought about by changing business initiatives and requirements eg. to make best use of a business opportunity, reduce costs, increase efficiencies etc.

Following these are those projects that the sponsors are just dead keen to get out of the way within a certain timeframe eg. to ensure no more money than is absolutely necessary is spent on silly things like software. This is where many projects sit.

Then there are those that really need to be completed within a certain timeframe however there will not be any major impact if they slip by a week or two on the grounds that earlier implementation could cause problems. Most sensible managers would rather have it late than compromised.

Lastly are those projects that are non-critical or ones where the sponsors and project managers are so laid back that the project can be completed just about when everyone feels like it. And believe it or not, I have seen a few of these.

Whatever the reason for your deadline, there are techniques that can be deployed to give your testing project the best possible chance of success. In a few cases, there may be genuine reasons to say no way on earth can this be done on time and if all other aspects are non-negotiable, then there is a strong case to take a walk. However, there are probably more walks taken than really necessary and although in most cases, you will never get to the ideal, you may be able to get closer to it than you might think.

So let’s take a ride…

What Are We Aiming For?

So what constitutes a quality delivery? A project delivered on time, on budget, to expectations, beyond expectations, low level of defects outstanding when you go-live, low level of apres-implemente defects, software that does what its supposed to do and doesn’t do what its not supposed to do, test team still breathing, test manager not in an asylum? All of the above and more.

All testing whether it be unit, system, UAT or whatever is ultimately concerned either directly or indirectly, with what the software is going to be used for regardless of whether the need is business-, scientific-, research- or whatever-based.

And to be successful, we must have something to aim for, something that says this is what it should do or this is what should happen. Without knowing where or even what the target is, the archer cannot hope to hit it… and if it is hit, its purely by fluke.

No matter how dramatic a deadline is, if the target is not well-defined then the chances of getting anywhere close are greatly reduced. Hopefully somewhere in the system development lifecycle, specifications have been produced to an appropriate degree of detail for all areas of concern.

We’re not going to delve here into tips for developing specifications or bringing them up to scratch. We’ll assume that you know pretty much what you’re aiming at and have at least some sort of specification or business requirements to springboard from.

It should also be noted that the ideas presented here will work best in a project that is planned and potential shortfalls identified early. Their effectiveness when you’re a week from D-day and only 20% complete is going to be limited although not necessarily without benefit.

We’re also not going to get into the whys and wherefores of test automation. By automation I’m referring to use of the popular record/playback tools and not something that is required to execute your tests. If automation is to work, it requires significant time and investment so I’ve assumed that if you have a tiger-by-the-tail project, that automation is not viable. This is not to say that the principles identified here cannot be applied to an automation project.

What Do We Have At Our Disposal?

- **Documentation –** the better the documentation, the better chance there is of success. No doubt about it, I've seen it proven time and time again. In order to develop test requirements you must know what the software is supposed to do.

- **People** - testing projects, like all others, will not happen without people. Even in the most highly automated test environments, people are still paramount. We should also include leadership under this heading however more on this later.

- **Time** – no matter how much business executives would like to think IT projects can be performed overnight, they take time. In theory, the more time available, the better the quality however this is only true if the project is quality focused, well managed and the best possible time utilisation techniques are employed.

- **Budget –** IT projects cost, and many of them cost more than the business might be able to justify. The theory says the more money you throw at a problem, the greater the chance of solving it. However as per above, this is only true if the money is wisely expended.

- **Environment/Non-Human Resources** – as much as we would like to, we cannot test without something to test on and this includes your application-under-test. Again prudent use of these resources can enhance the chances of success.

If you like analogies, consider this:

The documentation is the roadmap, time is the roadway, the application-under-test is the car, people are the drivers and passengers and the budget is the fuel.

The more accurate the roadmap, the better chance there is of reaching your destination. The longer the roadway, the further the car can travel. The bigger the car, the more people can ride in it. The better the driver, the more chance you have of getting where you need to go. The more fuel you put in the car, the longer it will go for.

So at one end of the scale, we have a project with a thimbleful of fuel in an old Model T Ford on a narrow, dirt road that leads to nowhere and no map.

What about the other end; a petrol tanker equipped with a GPS and nitrous oxide on an autobahn with the passengers tied to the roof and Schumacher at the wheel!

What Can We Do To Maximise the Potential for Success.

Making the best possible use of the resources at your disposal can increase your chances of delivering a successful project. This will require a certain amount of creativity and a never-say-die approach on your behalf.

Risk Analysis

The documentation will (should) tell us what we're aiming for. However this alone does not usually qualify us as experts in what the software should be doing. We often need to get our callous-less IT hands just a little dirty by digging into the requirements and/or the business and unearthing the things that are really important.

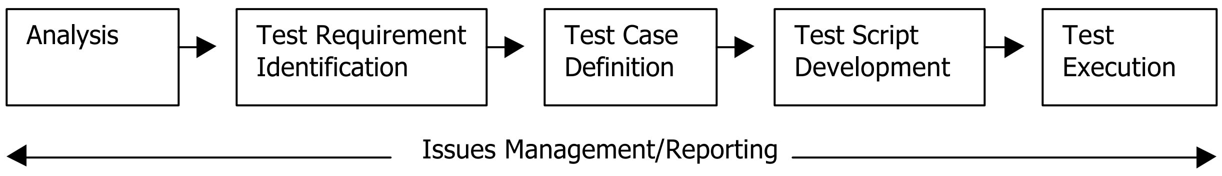
In addition, looking at past exercises can help determine what has gone on before and help to determine where the likely problem areas might be.

Lets look in a bit more detail….

If the deadline is looking insurmountable then focusing on the areas that really matter can assist in bringing a semblance of reality to the effort. This is commonly referred to as Risk

Based Testing or in simpler terms, Do-The-Important-Things-First Testing. Let's have a look at some of the key points.

Before we can assess risk, we must know what our test initiative will comprise of. This can be determined by following the traditional method of testing, which in simplistic terms, looks something like this:



I do not wish to trivialise this process, it should be at the heart of all testing projects, however we won't be going into the semantics in this session. I'm sure we've all been exposed to it in some form or another and needless to say, the more closely this process is adhered to, the greater the chance of success.

Risk assessment is best started as early as possible in the system development lifecycle then broken down further through the rest of the process. Usually in testing the first area for risk assessment is the test requirements, which can often be taken from the business risk assessment (provided one has been performed). Be sure to cover the lower levels, for example, it may be critical to determine the overall risk covering a certain test requirement however the test cases within that requirement may be assessed with differing priority levels.

As a loose guide however the following points should be included in any risk analysis (I use the word 'module' below to mean an entire system, a sub-system, module, function or even a single calculation):

- **Customer impact and visibility** - what will be the impact on the customer if this module doesn't work or is flawed?
- **Business operational risk** – will problems in this module present any risk to internal processes. There may not be an immediate impact on the customer or he may not ever know there is a problem.
- **Frequency of use** - is this module used frequently and/or repetitively?
- **Prior defect history** – has this module had a history of problems?
- **New functionality** – is this module completely new?
- **Modified functionality** - has this module had a reasonable level of modification made to it or does it contain new functionality somewhere within it?
- **Continuity** – will problems affect the continuity of use of the module? Users are often more interested in ensuring the software does everything it did before than they are in new bells and whistles.
- **Relative complexity** – is this module complex by its very nature?
- **Prior testing initiatives** – has this module been exhaustively tested in previous releases or test phases? Conversely - has this module been only cursorily tested in previous releases or test phases?

In your own project(s) you may well have others.

Methods and strategies of risk assessment may differ from project to project however the points below should provide a good starting point:

- **Old test logs and bug reports** – check to see where the problem areas have traditionally been (if its not a new application).

- **Exploratory observations** - perform "smoke" or exploratory tests first to determine where the problems are likely to come.

- **Confidence level** – listen to where the developers, testers and users etc. suggest the software is relatively clean and conversely where they suggest the problems may be.

- **Level of test coverage** – do the test requirements and cases cover all the significant features, functions, modules, areas etc?

- **Positive/negative conditions** – do the test cases cover both positive and negative test conditions?

- **Miscellaneous** – for example, where new or unproven technologies have been introduced, areas that relying upon interfaces, where quality practices have been poor, where specifications are loose or ambiguous, where a variety of people have been doing the coding, where a particular developer renown for short-cutting has been working, areas that have had high staff turnover etc.

Using these methods you should be able to bring the correct focus to your risk assessment.

Assessing each requirement and case against these criteria will help you build a priority profile. You may even wish to add a weighting to reflect importance of expected issues. This can be done for each requirement/case by assessing what the likelihood of problems is and what the impact is if problems do arise. For example:

Requirement	Impact	Likelihood	Risk Factor
TR1	4	2	8
TR2	1	6	6
TR3	3	5	15

Once we have completed the risk assessment then test requirement/cases can be assigned the appropriate priority. By starting with the high priorities first and estimating the time for test case development (where necessary) and execution, you can quickly get a rough picture of whether you can meet your deadline. Test estimating is a whole separate field which I won't go into here however be sure you include adequate time for test data set up, defect fixes and retests, defect logging and reporting etc. as these are often omitted from estimates.

Now that you have your prioritised set of test requirements and with any luck, test cases, and your estimates of how long its going to take to test (allowing of course for a certain failure rate or level of defects) you can set about setting the appropriate expectations with your project managers and sponsors. You might need to do a bit selling here, as their expectations are nearly always that the product will be delivered fully tested with no bugs. Use your test plan as the basis of setting the expectations and selling the plan for what can be achieved in that time with which resources. Make sure that in one form or another, you obtain their agreement (in writing is preferable) as to what your testing initiatives will and will not cover.

There may also be occasions where you have play the role of the obnoxious son-of-a-camel and challenge the business (or whomever) over their views on potential risks and priorities. You may even have to the resort to asking if they had one requirement that they needed covered first, what would it be? Then second, third etc. Always emphasise what you can do rather than what you cannot. It never ceases to amaze me how quickly some business managers backtrack when challenged over risk and priority. Sometimes I even wonder whether they deliberately overstate the risk just to make sure that we are kept on our toes although I'm sure that no reasonable manager would ever do this?!

However the risk analysis exercise doesn't end there. You need to be constantly reviewing your priorities. Sometimes we can start out in a reasonably healthy position but for one reason or another find ourselves quickly behind. Re-assessing on a regular basis will keep your efforts focused on the real issues.

Prioritisation doesn't just apply to test requirements and cases. In fact it's more commonly used in defect analysis. By using the same principles as above you can very quickly determine which problems really need fixing straight away and which don't (and avoid the 'everything-is-a-P1 syndrome' that users are renown for). Re-assessing on a regular basis will ensure that only the really important bugs are focused on first.

You may find your prioritisation criteria will change as the project progresses. Always ensure that you consult your wider project personnel (including your end-user) before this happens as they may have a different view than you as to what is important and what is not. And again you may find yourself having to challenge.

The simple message of risk analysis and subsequent prioritisation whether it be for test requirements/cases or defects, is to make sure you do the important bits first and that the determination of what the important bits are, is tracked, managed and where necessary, challenged.

People

The key resource for testing. Without them, and this includes the testing manager, testing won't happen.

However what if you haven't got the right people or not enough of them or precious few people at all? What if you have a good team but your deadline is so tight that you'll have to burn them out to get there.

If you have a people shortfall and your project manager cannot provide you with bucketfuls of cash for you to go and hire more, there are ways to plug the gap, albeit not necessarily ideal.

Firstly, make sure you are making the best possible use of the existing personnel in your team. Don't always assume that if your test team is finding it tough going that more people is necessarily the answer. Some simple staff management techniques can be employed to determine if your team could be better deployed eg:

- Meeting one-on-one regularly to discuss issues etc.
- Regular team meetings for communication, airing issues etc.
- Making sure objectives and roles are clearly defined
- Delegating effectively
- Ensuring that the appropriate effort is being expended
- Implementing appropriate measurement and evaluation criteria

However, if its clear that more people would enhance your success potential and you need them in a hurry, there are a few good places to start:

- **Support Staff** - if you can get management to buy into the idea (and wild promises of better quality software usually does the trick) these people know first hand what sort of problems are experienced by end users and where they're likely to be found. Sometimes you might be able to secure their time on a partial basis only however it might be better than nothing especially if they're prepared to work outside of hours.

- **End Users** – there's no better advocate for better quality software than a frustrated end-user. Again you may need to get management buy-in or accept only part time hours but with careful management this should not be a hindrance. Beware though, if your end user works for a client company, this could result in a conflict of interest.

- **Extramural Staff** – sometimes you can run your testing into the night and on weekends. If so you might be able to make use of people who need a second job, either internal or external. You will need to ensure though that these people can last the distance, as there's no greater bug sieve than a tired tester.

- **University Students** – the obvious place to start is with IT students however this does not always turn up good testers. I've found engineering students make good testers and believe it or not, so do music and psychology students. Make sure your students can commit the time though, as when exams roll around they will be decidedly scarce. You may also have the perfect timing for your project and have it running during their summer break.

Methods of acquisition can include advertisements in local papers, soliciting candidates from business managers, use of 'intelligence', advertisements on company noticeboards, calling student job search agencies, beg, steal or borrowing in any way you can.

However there is one major trap. You have to make sure that your tester is up to the task and obviously the risk is much greater with untried personnel. A few pointers for identifying potentially good testers:

- Pessimism but the type that is not confrontational and keeps within the bounds of searching for the worst to achieve the best.

- The ability to hone in on the important issues and gravitate towards the areas most likely to turn up issues – the proverbial tester's nose.

- A curiosity that wants to try the unexpected as well as the routine.

- A polite firmness with the ability to raise the red flag when its required.

- A degree of computer literacy (not being able to use a mouse might be a problem).

- At least some sort of exposure to business software.

- A total and complete lack of fear of hard work and long hours!

You can minimise your risk in a number of ways:

- **Do a characteristic/skill assessment** - your HR department may be able to help or a friendly recruitment agency (for which you may need a good excuse as to why you're not hiring from them). As a test manager you know (or should know) what attributes make a good tester, make a list and ask potential candidates to rate themselves. If they're internal people, get their manager to rate them also.

- **'Good Buddy'** system – put the casual alongside an experienced practitioner. Have the casual learn by watching the masters (assuming you have one or two).

- **Training** – maybe you can run an ad-hoc training course for your casuals. If you have the budget, you can bring a trainer in or send them on a public course, if they're available.

If you want casuals to help you through the execution phase of a test cycle then you need them to be able to do two things:

- Execute a test script
- Write a defect report

Assuming that your test scripts have already been developed and are clear and concise, they must able to follow the instructions and when something goes amiss, describe accurately what happened. I won't go into the details of writing defect reports here, suffice to say the more detail that is provided and the more accurate it is then the better the chance of a fast remedy.

Don't assume though that your casuals will be restricted to execution. If you need them to develop test scripts, many will be quite capable. However there are two other attributes necessary:

- Understand how business software works (trivially said but no mean feat)
- Articulate a requirement and put it into a series of steps to follow (same)
- Write clear instructions

Again appropriate coaching, mentoring and training will bring your casuals up to speed although a much longer learning curve will be evident.

It should be noted though that there might be a conceptual issue to be grappled with having external people involved in certain types of testing. UAT for example, is designed to give the users the opportunity for acceptance of the software and external personnel may not be the most appropriate resource. You may need to balance convenience with compromise.

I remember a few years back running a system testing project that required a large number of test scripts to be written and executed in a short space of time (to meet an unrealistic deadline as usual). Fortunately we were running alongside the university summer break and we brought in 25 students (after sitting an aptitude test) who developed 2,500-odd test scripts over a 3-month period. Some of these people were eventually offered full time jobs with the company and progressed into development and management, others went back to full time study (in fact, one is now working for us at iSQA as a test manager 3½ years later). Of them all, only one turned out to be a dud (he was the guy we found asleep under his desk one Monday morning, much the worse for wear from the previous night). This exercise opened my thinking up to the value and potential of untried people if you give them a chance and the right input.

The message here again is quite simple. If you can go out and hire professional people, go for it. If you cannot, there are other options.

I should maybe mention leadership and team spirit et al here also. There's only so much to be said for wise management of resources and as a leader of people, the test manager is in no different position than any other manager. A good leader impassions his people with his vision and imparts an enthusiasm that is infectious. If you can effectively communicate your goal for your team to your team and obtain their buy-in then you will have fewer problems with their availability for extramural work and giving that added extra. Now it has been said in some quarters that trying to impassion a test team can be like coaxing a cat into water and there is a popular misconception that testing can be mundane and boring (I really cannot understand why). So it is imperative that good team dynamics are employed and everyone is kept 'on the edge' so to speak. I won't go into detail on this here as I'm sure that any good book on leadership can you provide you with much better examples than I can.

Time

Not enough time causes stress, too much leads to laxity. Project managers have a propensity to avoid the latter then finding the former increases more than the latter decreases! Great! Not that a little bit of stress doesn't help to keep the gun loaded however too much can be counter-productive.

Lack of time on an IT project usually leads to long nights and even longer weekends. These are the obvious areas to use if you're behind and in many cases a few late nights and weekends might be all that is needed to get on track. However if we are to sustain this over an extended period of time then we either need super-human staff or more of them (see previous section). It is also worth noting here that I really do recommend evenings and weekends are only included as contingencies. It can be very demoralising for project personnel to embark on a project that already looks doomed to fail - and it will if you schedule evenings and weekends up front.

There are a few other initiatives that can be taken to make the best possible use of time:

Multiple shifts – depending on your team availability and that of your test environments, you may be able to steal a march on your deadline by requesting that your team go to multiple shifts (and providing they agree).

This is where some of your team, rather than work standard office hours, are scheduled for the duration of the test project (in full or in part) to work in the evenings, eg. 4:00pm to midnight. If you're really up against it, you might even consider the 'graveyard' shift – midnight to 8:00am or similar.

'Round the clock testing' can provide you with significant gains in time however there are a few 'gotchas' to watch for:

- Be sure you don't burn your team out, they'll be of no use to anyone further down the track if you charge too early. The key is to plan and pace, plan and pace. Make sure that you're always revisiting your plan, as a project manager (which is in essence what you are in this role – with a twist, of course) you should be doing this as a matter of course anyway. Ensure that the intense effort is going in where it has the best effect.

- There's no point in multiple shifts if any defects that are logged cannot be attended to until the next day. One of the keys to testing is the timely turnaround of defect fixes and if you're at the stage of multiple shift testing then I'd suggest that leaving high-priority defects longer than a few hours before they're at least investigated is going to cause you problems. This does mean however that you will need to get your support/development team to buy in to this idea as well, which might not always be easy. You may also have to accept the fact that you may have impassioned your test team into these crazy working hours but the development team might not see things quite the same.

- Make sure that by going to multiple shifts, that you are not merely doing after hours what could be done inside hours with better utilisation of resources eg. test environments (refer to the discussion on duplicate test environments). I once had a test manager who went to a second shift and all his second team did was perform testing that could have been done inside hours with a second test environment. This may be less costly however a problem usually has more than one solution; the trick is to choose the best.

- Ensure that the effort of the additional shifts is monitored and the output quality-checked. Unless you're a total insomniac, you are probably not going to be able to manage this hands-on yourself. It might be worth your while appointing the more senior members of your shift teams as shift leaders and request a report each day of the previous night's activities and status.

Appropriate scheduling – I think we all tend to put down eight hours (or so) per day for testing just as we do for any other task. However reality dictates that you're probably going to get no more than six (or so) productive hours on average per day. Once you have allowed for meetings, interruptions, down-time, paperwork, sick leave etc., even six might be ambitious.

There's also the question of ensuring that you have allocated the appropriate team members to the tasks that they're going to be the most productive on. Although it might be a good idea longer term to give Joe Bloggs the Inventory module so he can come up to scratch on it however if he's the Invoicing guru then this could hinder your progress on both modules. Tiger-by-the–tail testing projects are not good training grounds!

Test Case Maximisation

When selecting test cases to develop and/or execute, start with those that will give you the best possible coverage. Consider the following:

Test Case	Test Condition
1	A
2	B

3	C
4	A & B
5	A & C
6	B & C
7	A, B & C

If you start with #7 and it passes then you may choose to safely assume that the remainder of the cases will also pass. If #7 fails, then do #4 through #6 and only if these fail, do #1 through #3. The best case scenario is that you only need to execute one test case, worst case is all seven however you may escape with only four.

Unstructured Testing - while I'm not a great fan of this approach as a substitute for a good methodology, I believe it has its place especially when testing new or heavily modified software. This approach involves a team of testers tackling an application (or part thereof) totally unstructured and crashing through ad-hoc test conditions based on the knowledge and expertise of the testers and doing it within a defined time frame. The concept is that with the testing effort capped and the scattergun nature of the approach, you turn up as many defects as possible in the shortest amount of time.

If you're going to use this approach, ensure that you record your test steps. While this might reduce the amount of hands-on testing time, it will provide you with the ability to repeat the tests and provide an audit trail. The documentation produced may then be used as a base for more formal test scripts.

Use "Real" Data - as opposed to test data. This approach can always be relied upon to turn up a defect or two - hundred! Using test data is all well and good if you do not have access to the real data the application will use however even testers have a tendency to construct test data to make the test pass which is not always what we're after. Real data will provide the application with real scenarios and make the testing more applicable to the way the application will be used once in Production. Please note that when using real data that I'm not advocating performing testing in a Production environment. If it is to be used, a copy will be required.

Test Phase Overlap - if your application is modular, you may be able to overlap the test phases and commence the next phase before the previous one is complete. This may be particularly helpful when running integration tests which test the ability of the various units to function together, with functional testing which tests the end-to-end functioning of all units.

Test "Stub" Creation - this idea is particularly useful when testing applications requiring interfaces. I once had an exercise where a module relied upon an outbound interface file being manipulated external to the application and then returned as an inbound interface file. The application was ready to test but the external manipulation application had not been developed. Rather than wait for this to be done, we developed a small piece of external code that let the main application know that the outbound file had been sent. We then manually created an inbound file and placed it in the location where the application was to pick it up. The net result was that we were able to test the main application without waiting for the extra code to be developed.

The "stub" approach is not a substitute for proper testing once the interface has been developed, more of a "fudge" to enable the earlier testing of software that would otherwise had to have waited. You will need to ensure that you don't forget to go back and retest that area once the interface has been delivered.

Achievement re-evaluation – as you progress through your timeline, you will need to continually be reviewing your deliverable. There will be many occasions where due to circumstances beyond your control, eg. poor defect turnaround, that your deliverable will need to change. Make sure that you communicate to everyone what the revised expectations are and as you will no doubt receive challenges from your sponsors and project manager, be ready to justify. Ensure that you gain agreement, preferably in writing.

If you are able to identify your shortfall(s) early enough, the following suggestion can be a major time optimiser:

Early involvement - this can be easier said than done but the IT project that sees testing as the last step of the process will always place unbelievable pressure on the poor test team and on the remainder of the project team as they fight to fix up all the defects that could have been found earlier. A good test manager recognises the benefits of getting himself and his team involved right up front even at the analysis and requirements specification phase. You may well find that you've entered the foray too late in the day for this method to be effective however any opportunity you can get to deploy it will be of benefit.

A tester's nose is a valuable asset in the early stages of a project as it can sniff out potential "gotchas" before they eventuate - prevention is always the best medicine. Sometime we find that analysts and developers bring their special talents in such a way that focuses on the positive (and nothing wrong with that) whereas the balancing nose of the tester, while not intended to necessarily be negative, can bring more of a quality equipoise to the process.

Because the time and cost of fixing defects increases dramatically through the system development lifecycle, defects and issues identified early are easier and less costly to fix. It will also reduce the incidence of related or regression-type issues further down the track so it pays more quality dividends than just improved time utilisation.

Earlier in this session we talked about risk analysis. Starting risk analysis back at the requirement specification phase can help identify the risks associated with resulting test requirements and cases. If the test manager can gain agreement for test team involvement earlier in the development lifecycle, much of the risk analysis can be covered much earlier meaning more time for productive testing.

Early involvement not only assists in making best use of time but also assist in selling the project on the estimate for testing both in time and dollar terms and promotes testing as an integral part of the development lifecycle. It has the potential to improve the overall quality of the deliverable and of the process providing that deliverable.

Test Environments

As discussed earlier, we cannot test without something to test on. This should be a dedicated environment that is 'owned' by you as the test manager. No-one should be permitted to update of change these environment without your say-so otherwise you will very quickly lose control of them. Be especially aware on this point as I have had a number of occasions where project personnel have played lets-try-to-fool-the-test-manager!

Sometimes the nature of our projects and the availability of environments might dictate that we have no movement or possible 'creativity' to make better use of the facilities available. However if this is not the case then some of the ideas below could be of benefit:

Duplicate test environments – if the nature of your application or environment means that your test team is constrained by only having a single test system, then a second or even multiple environments could help. You will need to be aware of any conflicts eg. common error logs etc. however these can mostly be managed. You will also need to make sure that you appropriately split the effort across the multiple environments (see below).

If your test system resides on a separate machine, you may not necessarily need another machine for a second environment. See if there is space and/or processing power available on another machine. Make sure any compromises do not affect the integrity of your testing though.

I had a project a while back that used hand-held devices. I was told by the IT team that we could only have one test environment because we only had one hand-held gateway and the test environment could only support one anyway. Problem was solved when I managed to source another gateway and hey presto, we could have a second environment.

Isolated environments – there may be an opportunity within your test environment to split off a certain section, module etc. into a separate isolated environment. This environment may not support everything to perform a complete test cycle however may be sufficient to cover a subset of requirements. You will need to balance this with any consequences of not having the entire system tested in a single environment and how this reflects reality.

Managed demarcations – this is where a constraint is created by having everyone logged in, all testing simultaneously and the conflicts cause delay. I once had a project where the testers covering the batch processes would run their tests and the response time for everyone else dropped fivefold. While this was a legitimate concurrency test, it slowed performance to the point of causing delay so we changed the schedule to run the batch tests after hours. We could have run them in another environment or selected certain times of the day etc.

Whatever demarcations you arrive at, make sure that they do not compromise the integrity of your testing.

The management and creative use of your test environments often goes hand-in-hand with your people and time availability. It may not be possible to separate one from the other so ensure that if you do go for more people and/or more hours, that you can source any additional environments in a timely manner. You do not want to gain a few days advantage if it's going to take a week to get another test environment up and going.

Budget

Unless you hold the purse strings for your testing project then its unlikely you will be able to make any direct call on the moolah. So probably at best you will be restricted to going cap-in-hand to the powers that be for the much-needed funds to do the job.

There are a few tips and techniques that can help with this great solicitation:

Make sure you have a strategy and plan (including contingencies) – a well-thought out and deliberated plan will convey to your management that you know exactly what you're doing and that the only hindrance is lack of funds. Cover off any potential 'gotchas' and ensure that you include in your plan what you plan to do if your plan doesn't go according to plan, got that?

Explain the testing initiative – make sure you appropriately convey the testing issues. Clearly explain the benefits of proper testing, including an ROI if possible, to all stakeholders and maybe even involve them in the planning. I'm sure most of us know how to create fear, uncertainty and doubt over the potential calamities caused by a lack of or under-cooked testing. However beware that you don't oversell otherwise it might be seen that you're going for the Rolls Royce when the Ford is what you need to get over the line. Stakeholders are rarely akin to spending lots of money (not on testing anyway) however if you can show that funds wisely invested now may save a huge outlay fixing it all up when it goes wrong later, then you will stand a better chance of getting what you need.

Set appropriate expectations – if I go to the powers that be, promise the earth then don't deliver, my credibility is in tatters especially if I have been given the funds requested. Make sure that you convey the realities of testing and not pie in the sky. Set an expectation that you know you can achieve and, all things being equal, back yourself. If it looks like you might not be able to deliver as expected, keep your sponsors and stakeholders informed of the issues and why they're coming about. If necessary, reset the expectations but ensure you have valid reasons as you may be asked to explain.

Some ideas on smart use of the dosh once it has been approved

Down time – if your project suffers from environment down time, ask your team to go home (if practical) and come back in when its up, after hours if necessary. This especially useful when dealing with contractors, there's no point in paying $x/hour for someone to sit around twiddling thumbs waiting for the test system to come back up. You may need to counter this with any extra costs of people working after hours.

Unnecessary equipment – if you have to rent or hire additional equipment eg. printers, hand-helds etc. make sure you time their acquisition according to your test plan. Having an expensive machine hanging around for a few weeks while you catch up can be costly. See if you can get the supplier to maintain a certain amount of flexibility and maybe hold off delivering until the last minute. Watch the counter-affect though, as you don't want to be waiting around while the IT team spends a week configuring a new box when you needed it yesterday.

Outside resources – if you have the budget, you may use external resources (other than contractors) to complement your internal ones. Such resources may include outsourcing some of your testing, buying time on someone else's equipment/network, bringing in key experts when required etc. Again careful timing of these can ensure that you maximise the benefits and optimise the expenditure on such resources.

There's an old proverb that goes along the lines of this: 'a fool and his money are soon parted'. Spend it wisely or you'll loose it and when the accountant comes calling asking you where that last $100,000 has gone, you could be in a bit of a spot if you haven't delivered.

Summary

So you've got the immovable deadline. You need to maximise your efforts and give your team the best possible chance of success. In summary:

- Carefully prioritise you test requirements, test cases and defects according to risk making sure the high priority areas are tackled first.
- Ensure that you keep abreast of any changes in risk(s) or requirement(s) that might affect your assigned priorities.
- Look for additional testing staff outside of the usual channels especially if your budget is tight.
- Keep team spirit up with regular meetings and maybe the occasional social activity.
- Obtain buy-in to get the test team involved as early in the development lifecycle as possible.
- Consider going to multiple shifts if the nature of the project allows.
- Look at establishing multiple test environments again if the nature of your project allows.
- Set appropriate expectations up, down and across the team, resetting as required and as early as possible.

There are no easy answers to the immovable deadline. As mentioned, sometimes its better for one's reputation to politely decline an opportunity that seems like you will need to push many barrels up a hill. However in other cases, with some creative management you might get closer than you think to a quality delivery.

As mentioned, some of the ideas here may need to be tempered to your particular environment and/or application. Nearly always you will need to deploy combinations of them rather than the odd one here and there. Remember to keep setting and resetting expectations both upstream and downstream, no-ones like surprises.

Hopefully these will help with your next tiger-by-the-tail project. If I can be of any further assistance, please do not hesitate to email me at geoffh@istq.co.nz.