Foundations for Successful Testing

Author Ruth Keys, Independent Testing Consultant

Am Weissen Berg 1H
61389 Schmitten
Germany

www.keys-consult.de
ruth@keys-consult.de

Abstract For those new to testing or those without lots of testing success stories to share, this paper aims to help lay the foundations for successful testing.
It points out some of the stumbling blocks, which lie on the path of the unprepared tester and shows how to turn them into building blocks for a less painful testing experience.
Some things considered are:

- Getting your test planning embedded in the project plan.
- Who do you need on your side?
- What's needed to turn the test plan into reality?

Building on Success

For those new to testing or those who don't have lots of testing success stories under their belt, the EuroSTAR 2003 theme of "Building on Success" may be somewhat discouraging. However as Lloyd Roden the Conference Programme Chair says in his opening remarks to the Call for Papers "This year's theme picks up the challenge of encouraging one another through our successes. I believe encouragement is a powerful thing – we need to be able to build not only on our own successes, but also on other people's successes". Similarly in the British Government's Schools Green Paper from 2001 also entitled "Building on Success" the then Secretary of State for Education and Employment David Blunkett laid out the aims of paper in the foreword which included "to learn from the past in order to contribute to the future".

With this in mind, this paper sets out to share some of my experiences in software testing with an aim to helping others lay the foundations for successful testing. It will point out some of the stumbling blocks, which lie in the path of the unprepared tester and show how to turn them into building blocks for a less painful testing experience.

Four Cornerstones

So what are the foundations for successful testing? There are probably as many opinions on this as there are people involved in testing. Established models such as the Test Process Improvement model (TPI) developed by Martin Pol and his associates [1999] offer starting points for determining which steps to take towards improving the testing process. Lee Copeland [2001] offers a catalogue of best practices, which he calls the "Seven Habits of Highly Effective Testing Organizations".

I have chosen four cornerstones:

- No business involvement:
  No satisfied customers
- No requirements:
  No test
- No plan for dealing with bugs:
  No sleep at night
- No stable environment:
  No release without risk.

The wording may seem a little provocative, so let me explain. A colleague of mine likes to say "No pain – no change". Certainly I can attest to painful experiences in all four areas, which have led to my own desire for change. In the following sections I will expand on these statements and hopefully justify them.

1. No business involvement: No satisfied customers

Too many projects in the ICT industry are run with minimal involvement of the business stakeholders. Once the business requirements have been documented then the development department takes over and the only contact with the customers of the system being developed or enhanced is the odd telephone call or email exchange. So to paraphrase an old adage: we may build the system right, but we don't build the right system.

There is a simple remedy to this problem and that is to involve the business stakeholders in as many project and test activities as possible. This may seem to be stating the obvious, but as with many simple ideas they can be difficult to implement. As the following quote from a business project manager shows, there are many bridges still to be built between the technical and the business worlds. The project had just completed the definition of the business requirements and the requirements specification document had been signed off by business. The business project manager was heard to say "Well that's the business phase finished. We'll come back when the IT phase is over to perform the User Acceptance Test."

How to get business on board

Why the reluctance of people outside software development to become more involved in projects? Some of the more commonly heard reasons are:

- We don't have time for it. It's stopping us doing our real work.
- That's all IT stuff. It's too technical.
- We don't understand all that IT-speak.

Reading between the lines one can discover several ways to encourage more contributions from the customers.

Firstly, establish a win-win situation where the potential benefits of an active participation in the project become apparent. We are all more motivated if there's something in it for us! For the users of the system this may be the chance to become familiar with the new system or new functionality before they have to work with it. For the client this could be the opportunity to influence what will be delivered before the system is finally put into production, i.e. to ensure that the *right* system gets delivered, thus avoiding costly change requests and enhancements at a later stage. In short be aware of the vested interest of the particular stakeholder you want on board in your project team and talk to his concerns.

Secondly, it is important remove the barriers to good communication. Establish a project terminology, which is understood by all those involved. With respect to testing it may not be only the business stakeholders who need some explanation of testing terms. Even within the development team there may well be confusion concerning, and a lack of clear understanding of, all testing terms.

Communication is a two way process so have the business people define their terminology too! The important goal is to talk the same language. I have found that it often falls to the testers to act as interpreters between business and development – so get your dictionary defined for you to make your life easier or even to make yourself redundant in this role.

Finally don't underestimate the truth in the saying "a picture is worth a thousand words". When trying to explain something it often helps to move the context into an area that everyone can understand. If you aren't good at coming up with metaphors yourself then collect all the good ones you hear. One of my favourites to explain testing comes courtesy of Henry Muccini [2002] where he uses the analogy of a musician playing in an orchestra. Another excellent example is Elisabeth Hendrickson's Vasa Story [2002].

And where best to use them

Once you've got all those business people committed where do you use them? These are some of the areas of testing where I have had good experience in obtaining help from the business community:

- Reviewing the business requirements specification
  Have different people review the document from the authors!
  Try using reading techniques to review the document from the customer perspective [van Veenendaal 2002].
- Reviewing the functional specification
  Have the business requirements been translated into functions in the system which support the business processes?
- Reviewing screen layouts / prototypes
  Are the fields arranged into logical groups for the user?
  Is the sequencing of events correct from a business perspective?
- Writing test cases for the User Acceptance Test
  Input from the business is indispensable here. Their knowledge helps them choose the most likely set of combinations of inputs from all possible combinations, which results in a more realistic test of the business processes. Similarly they also know the exceptions that prove the rule and often come up with test cases, which are very effective at finding defects.
- Defining realistic test data
  Good test data is one of the key elements of successful testing. (For more information on this topic see James Lyndsay's article [2001]). The knowledge of the business stakeholders is indispensable in arriving at realistic test data.
- Performing the System Test
  Not normally considered the domain of the customer but there are benefits to be gained from having some business testers involved. They obtain an earlier insight into the system, which often leads to better User Acceptance Testing, as the experience gained is translated into test cases, which thoroughly exercise the system.

- Performing the User Acceptance Test
  This can be seen as a " train the trainer " opportunity. The users who perform the UAT are the future super-users, who can assist their colleagues during the introduction phase as the new system goes live.
- Performing Usability Testing
  Who else but the users can adequately assess the usability?

2. No requirements: No test

The first two sentences in the book "Mastering the Requirements Process" by Suzanne and James Robertson [1999] are:
"Requirements are the things that you should discover before starting to build your product. Discovering the requirements during construction, or worse, when your client starts using your product, is so expensive and so inefficient, that we will assume that no right-thinking person would do it..."
Unfortunately there appear to be quite a few wrong-thinking people developing software. It still seems to be common practice to be discovering requirements during construction. Judging by the number of papers given at conferences and to be found on the Internet about how to test without requirements, I am not alone in having made this observation.

The need for requirements

Why are high quality, signed-off requirements so important for testing? Without requirements every system behaviour is permissible or not assessable and it is not possible to define the expected results for test cases. Of course those elusive requirements do exist somewhere. They just haven't been committed to paper. They exist in the heads of various people associated with the project. To make matters worse they probably exist in multiple variations in those heads because there has been no review process to build a consensus as to exactly what the system should do and how it should behave. The tester is faced with the dilemma not to test or to take on the thankless task of collecting together the information necessary to specify the expected results for his test cases.

Signed-off requirements are also a prerequisite for categorizing failures reported during testing as defects or change requests. Without a recognized base level Problem and Change Management become an impossible task. Each failed test case will become an object of discussion and be resolved on its own merits, probably in isolation from others. As a result of the inability to define what is within the scope of the requirements for the system and what is out of scope, scope creep becomes almost inevitable.

In his book "Facts and Fallacies of Software Engineering" Robert Glass [2003] defines Fact 23 as being "one of the two most common causes of runaway projects is unstable requirements". Runaway projects are the testers nightmare as they invariably lead to never-ending development. Since those projects often seem to have immovable deadlines the testing effort gets squeezed between the rock of incessant development and the hard place of the project deadline.

Prevention

The lack of an adequate requirements management process is admittedly not a problem that can be solved by the test organization alone. The cause of the problem is rooted in the software development process and as such needs to be addressed there. The business problems to be solved by developing a new system or enhancing an existing one are unique and therefore new each time. This fact should be accepted. As a consequence it should be accepted that it will be impossible to "get it right first time". The software development process should be prepared for change and flexible enough to deal with it. We should also have the courage to accept "good enough" requirements and get them signed off. If it is agreed from the start that the development process is geared to change, then the reluctance to commit to the first attempt at requirements should also diminish.

Getting requirements signed off implies that they are documented and have reached a predefined level of quality. Part of the process to achieve this goal should include reviews of the Requirements Specification. Although it is unrealistic to expect all documents produced during the project to be subject to review, as a minimum, it is important that all major documents, which have a critical influence on the development of the system, should be reviewed. Just as I recommend that business stakeholders are involved in the review of the Requirements Specification, I also strongly recommend that as testers we participate in the review of both the Requirements and Functional Specifications. Testers bring a new perspective to the review process, as they are primarily interested in whether the requirements are testable. Asking the question "how can I test whether this is fulfilled?" highlights many an ambiguous requirement.

The Cure

If your project is already past the stage where any of the above can be applied, what options do you have to test without requirements? There are probably other sources of information on which to base your test cases. Some likely places to look are:

- Prototypes / screen layouts
  These are a possible source of information as to the type of inputs the system expects.
- Problem Management databases
  If the product is already in use in an earlier version, then existing problem reports can be used to write test cases to verify that the problem has been resolved.
- Similar systems
  Identify similar systems on the market or in use in the company and try to obtain handbooks or evaluation software.
- Request for Proposal/Request for Tender documents
  If the software is not to be built in-house these documents should be available and contain high level requirements. Although probably not at level of detail to produce test cases they are at least a starting point.

- Minutes of meetings, emails
  As a last resort when you're getting really desperate!

I have tried all of these options at one time or another, usually a combination of a number of them. Although not the perfect solution it is certainly a feasible one.

If your project isn't already a death march [Yourdon 1997] and you can persuade the key team members and project stakeholders to give up a good chunk of time to participate in a JAR session, then this is a good way to collect requirements. The Joint Application Requirements session brings the knowledge bearers together to interactively define the requirements. One way of doing this is described in the book Rapid Testing [Culbertson 2002]. I have personally never used this method for obtaining a complete set of requirements, but have participated in such sessions to quickly assess and analyse change requests. After an initial scepticism among the participants it was soon accepted as being an efficient way of dealing with change requests in the organisation.

Whether you decide to use the document mining technique to define test cases or to hold JAR sessions to elicit a set of requirements, having invested a lot of time and effort to produce these documents it is essential to get them reviewed and signed off. This gives the assurance that you are assigning your efforts to testing the *right* system.

If despite your best efforts you still haven't been able to dig out anything other than a set of high-level requirements, now is the time to consider using exploratory testing techniques. There has been much written on this topic, so I refer the reader to the literature [Bach 2002]. However, my experience has often been that it is those projects, which live with uncertainty during the entire development process where the resistance is greatest to exploratory techniques. Testing is suddenly seen as the last chance to head off disaster and the willingness to try anything new is zero.

3. No plan for dealing with bugs: No sleep at night

I could have called this one "No plan for dealing with bugs: No test" too. It often seems to me that development is convinced they will produce the perfect software first time round. In which case we don't need to test it, do we? Why do I think this? I have seen many project plans, where testing is only including once as one block at the end of development and there is no time at all scheduled for bug-fixing. Another strange thing is that whatever the size of the project, testing always seems to be scheduled to last just 4 weeks.

I'm afraid I'm not one of those who believe that the test plan should be based on an approved project plan. Testing is an integral part of the project. Test planning must be embedded in the project plan. The planning of the entire project must be a joint effort involving development and testing. The goal of the testing subproject must be to get all test related activities, with their dependency on development tasks and with realistic schedules in the project plan.

Common Mistakes

Some of the common mistakes made are:

- assuming that each phase of testing is only performed once;
- omitting bug-fixing from the development task list;
- assuming a constant “drip-feed” of bug-fixes into the testing environment.

There is no allowance made in the plan for retesting. Retesting occurs at all levels of testing. It becomes a significant planning problem as soon as the developer is no longer able to test and debug his software independently of others. This situation occurs at the first level where the testing is performed by an independent testing group or where multiple components are integrated. As soon as such dependencies exist then separate tasks must be defined and scheduling becomes important. It may not always be possible to accept each new release into the test environment as soon as the development is finished.

Another mistake made is that even if retesting is included in the plan each test cycle is planned with decreasing effort. This may be appropriate but may also indicate that regression testing has been ignored. The number of bugs to be retested should be diminishing with each test cycle so that the retest effort is also decreasing. However regression testing effort may not necessarily decrease. This is dependant on how wide reaching the effects of the bug-fixing may be.

And how to avoid them

The answer to the problem is a simple one. It involves normal project planning skills.

1. Draw up a detailed Work Breakdown Structure for all test related activities, not forgetting all the preparation activities, retesting and regression testing.
2. Indicate for each task which deliverables are required as input to the task. It’s here that the dependencies on the output from other tasks in the project become apparent.
3. Allocate only resources, which are physically available, i.e. don’t plan with A.N. Other who may never get hired or freed up from existing activities.
4. Combine all development and testing activities in one Gantt chart. This is another of those pictures worth a thousand words.

The symptoms of failure to achieve an adequate strategy for dealing with bugs are unpleasant and worth investing the effort up front to avoid them. As dynamic testing is the final task before the product release and subject to time-squeeze, if you fail to get testing adequately represented in the project plan be prepared for long days, no weekends and no sleep at night. Although such test team heroics may be needed in exceptional circumstances, no tester wants to have such demands made of them permanently. They will almost certainly vote with their feet eventually.

4. No stable test environment: No release without risk

Do any of the following situations sound familiar?

- A bug that was corrected in the last release suddenly reappears.
- The release you receive from development requires you to run nearly every test suite, although only a small number of bugs are fixed.
- A release that was supposedly only to support a new platform suddenly contains new features too.
- The software that ran perfectly on the developer's PC refuses to do likewise in the test lab.

If you answered yes to any or all of the above, then your organization is probably missing Configuration Management and/or Release Management. Both can be considered to be subdisciplines to Change Management, because both address different aspects of managing changes to software systems.

Software Configuration Management

What is Software Configuration Management (SCM)? A simple definition is: "Software Configuration Management is the discipline of controlling the evolution of software systems" [Tichy 1988]. The software configuration is the set of documents, programs and data that fully describe the system. The job of SCM is to control the software configuration. The important issue is that change is accepted as a normal side effect of developing software. SCM helps to control *how* the changes are made to the system.

Without SCM there is often confusion. Simultaneous changes are made to the same source code by different developers and some of the changes are lost. Changes are made to interfaces and not all users are notified. It isn't possible to identify the changes made in a particular version of the software. These problems can waste considerable amounts of time; often happen at the worst possible moment; can be extremely costly and almost always raise everyone's level of frustration.

Ed Kit explains testing's interest in Configuration Management in his book "Software Testing in the Real World" [1995] in the following points:

- to manage its own validation tests and their revision levels efficiently;
- to associate a given version of a test with the appropriate version of the software to be tested;
- to ensure that problem reports can identify software and hardware configurations accurately;
- to ensure that the right thing is tested;
- to ensure that the right thing is shipped to the customer.

He also adds "to ensure that the right thing is built by development". I personally do not consider that to be influenced by SCM but rather by Requirements Management. I would replace his point with my own – to ensure that the right thing is delivered into the test environment by development. One of the most frustrating tasks in testing where no formal SCM is established is determining just what components need to be installed in the test environment to enable testing to begin. Much time is wasted and many tests have to be repeated because of spurious test results caused by incompatible software installations.

Set a good example

If there is no formal SCM within the organization one can try the bottom up approach and set a good example for development. Introduce the principles of CM into the test process.

Maintain versions of your testware. This can be as simple as setting up a directory structure to separate the different versions of all documents and software or may involve using a test management tool to help keep track of revisions to the testware. Older versions that are not so frequently used can be transferred to a suitable back-up medium.

Know and document what version of software is installed in all your test environments. I prefer to keep this information visible to all – not hidden in some file on the computer that no-one looks at or keeps up to date. I usually have a large flip chart up on the wall with a test environment / component matrix. The current version of each component is documented using post-its. The word spreads quickly and similar flip charts often spring up in the offices of the developers.

Release Management

I had to search quite hard to find a definition of Release management. I didn’t find one issued by any standards board. The following definition comes from SiteLite [2003] who are a ICT Management Services Provider: “Release Management is the process that encompasses the planning, design, build, configuration and testing of hardware and software releases to create a defined set of release components.” Release Management controls *which* changes are made to the software system and *when*.

Testing is interested in Release Management for the following reasons:

- to know what is affected by changes made;
- to know when the changes are made;
- to ensure changes made are testable
- and thus to allow for a more efficient testing process by using testing resources economically.

A badly planned release from the testing point of view may not be for development. If testing is not involved in release planning the changes chosen for a particular release may lead to unnecessarily high effort in the test organization. What appears to be a small change for development may affect large areas of the testware. Of course test cases and test suites should be designed to be robust to changes in the software under test, but sometimes major changes to the testware cannot be avoided. At these times it is advantageous to have accurate information regarding the exact nature of the change and to have sufficient time to implement the changes necessary in the testware and test environment.

Possible Substitutes

Without formal Release Management there are some things you can do to avoid nasty surprises arriving with each new release from development. Insist on release notes from development stating all change made and all bugs fixed in the release. Try to get this document as early as possible and not with the delivery. This allows you time for test planning. If you can't get development to provide the release notes then write them yourself and get them to sign them off.

Another good practice is to include a field "to be solved in" in your Bug and Change Request tracking tool. Encourage development to use it to track which release of the system will contain the bug-fix or change request. The field should be "to be solved in" and not "was solved in". In other words the field should be filled during release planning and not once the build is completed. This should hopefully provoke development into doing some release planning and gain you the time for test planning.

The best way to get release information as early as possible is to ensure that testing is represented on the change board. This offers the biggest chance to influence what goes into each release and thus to enable an efficient use of the testing resources.

Summing up

If Change Management is still fairly chaotic in the organization with no visible Configuration or Release Management, the chances are high that you will be plagued by an unstable test environment. Forgotten components will have to be installed on the fly into the environment. You may even find yourself in the situation where you are forced to accept more than you had planned for into test as a hot fix for a bug halting testing can only be provided if you take features x,y,z too.

If the organization can't manage the transition from development into testing in an orderly manner, then what grounds exist for presuming that the transition into a live environment will proceed more smoothly? There can be no release without risk in these circumstances. The risk is not in the quality of the software but in the quality of the supporting processes.

Closing Remarks

"You see things; and you say, 'Why?' But I dream things that never were; and I say, 'Why not?'" *George Bernard Shaw*

I hope this paper has provided some ideas as to how to avoid saying "why?" and proceed directly to the “why not?” stage.

References

Bach, James. 2002. Exploratory Testing Explained. Available at www.satisfice.com

Copeland, Lee. 2001. Seven Habits of Highly Effective Testing Organizations. Available at www.stickyminds.com

Culbertson, Robert, Chris Brown, Gary Cobb. 2002. Rapid Testing, Prentice-Hall

Glass, Robert. 2003. Facts and Fallacies of Software Engineering, Addison-Wesley

Hendrikson, Elisabeth. 2002. Going down with the Ship. Available at www.stickyminds.com

Kit, Edward. 1995. Software Testing in the Real World, Addison-Wesley

Lyndsay, James. 2001. The Importance of Data in Functional Testing. Available at www.workroom-productions.com

Muccini, Henry. 2002. Available at www.henrymuccini.com/Testing.htm

Pol, Martin and Tim Koomen. 1999. Test Process Improvement, Addison-Wesley

Robertson, Suzanne and James. 1999. Mastering the Requirements Process, Addison-Wesley

SiteLite. 2003. www.sitelite.com/solutions

Tichy, Walter F. 1988. Paraphrased from “Tools for Configuration Management”, Proceedings of 2nd International Workshop in Software Version & Configuration Control.

Yourdon, Edward. 1997. Death March, Prentice-Hall

Van Veenendaal, Erik. 2002. The Testing Practitioner/Reading Techniques, Uitgeverij Tutein Nolthenius