

Tests without specs

Good testing requires system documentation - but this is very often poor or non-existent. It's no use complaining, so

George Wilkinson explains what to do about it



It usually goes without saying that to derive good scripted test cases a tester or developer needs a good 'frame of reference' from which to work. This will normally come in the form of a requirement, functional or design specification. Essentially this is something to measure the quality of test object, the software product under test, against.

Well-written and tested requirements make a tester's job easier by a good margin. It saves the contractor and the customer both time and money and helps build a better product. Today, sadly, testability is not injected very well into documentation when requirements capture is undertaken, if at all. There are of course many reasons for this, but the main one is probably that organisations see testing as a discrete, separate part of the development process, which clearly it is not.

So when untested, or no, requirements come your way, at whom do you point the finger? Everyone's problem is no-one's problem. So you have to start somewhere. Your development process is obviously one of the failings, but what is your company or departmental Test Strategy? The Test Strategy is the business case for testing and will include things such as test level, test coverage and recommendations for test design techniques suitable for the business. If your organisation is not recommending your strategy, are you testing the right things in the right way? A good Test Strategy would help prevent poor requirements by introducing a requirements testability analysis activity or similar; allowing you to enter the test generation phase with a better 'frame of reference' and more at peace with the world.

So what if we as testers find ourselves in that situation, where we get delivery or notification of a product heading our way with little, badly written, or no requirements. What can we do apart from panic or play dumb? This article will describe some techniques that can be used to derive test cases in this situation. The techniques explained are relatively generic and can be applied to most software products. For simplicity, the examples used are written at a high level.

Identification & definition of Test Requirements

Tests, as stated above, are developed from the sources of available information to describe the product being developed at a specific level of the development life cycle. Identification of Test requirements (sometimes called test conditions) is used as part of the standard test design process to assist in generating a satisfactory suite of test cases.

Test requirements can be generated using past experience, brainstorming or test specification techniques.

Use past experience

Draw from past experience; think about how you may have approached a similar project before. Even if the project in hand is not the same, think about how things went last time, what did you test, how did you test it, who did you contact – was there anyone on whom you could safely sound out ideas? Were there any particular buggy areas in past or similar projects? Maybe searching through a defect database may help get you going and focus on where you could help make a contribution to the project.

Brainstorming

Brainstorming can be an effective way to generate lots of test ideas and then determine which of them best solve the problem. Brainstorming works best when you have a larger group of varied people – this will result in the widest and most creative range of ideas and hopefully good test requirements. So it may be worth inviting developers, other testers and users if possible.

Test specification techniques

There are various ways to generate test cases using test specification and design techniques. The reasons for choosing specific techniques will come from the type of software you are testing and from your experience. It may also, in some instances, be enforced by the customer: eg in a safety critical environment. This however should ideally be stated in the test strategy or test plan.

Step 1: read the project documentation

Test requirements are the ideas of *what* should be tested, not how to test it: that will come later. They will normally be gathered from a number of sources. Useful documents include:

- User guides, manuals etc
- Requirements specifications
- Functional specifications
- Design specifications

Ideally all the relevant levels of specification should exist; eg acceptance test cases will rely heavily upon customer or user requirements, whereas integration testing will rely upon low-level requirements, or a low-level specification describing data flow. What specification material actually does exist of course depends upon the nature of the product, the development process and what people have what time to create it.

Essentially requirements at any test level are an important and necessary part of the software development process, not only to build the product, but also to test it. Given correct and well-written requirements you can normally just work from these using standard test design techniques. However when requirements don't exist as a basis for test design, say in an ad-hoc working environment, Test Requirements can still be generated. Generating these Test Requirements from scratch is certainly not ideal but can be necessary as the only way to create a satisfactory suite of test cases.

Even when requirement documentation does exist, some of the techniques described here can help with gaining more understanding of the product.

Step 2: preparation for interview - getting your bearings

Understand what you 'should be testing' and what is essential to you and your team. You may have to base your test design on past experience or asking others with relevant standing in the project. If you don't have any strategy it may be an idea to seek guidance. If you are acceptance testing, this will probably

Interview questions and answers

This example assumes that no requirements specification documentation at all exists, and is not complete

Q	A	Q	A
What is the product?	A building security alarm system	Who will use it?	Security guard, intruder
What are its features?	Set alarm, test alarm, detect intruder, sound alarm, etc	What is the function of the 'detect intruder' feature?	To alert security staff to unauthorised entry into the building
How should the system work?	The security guard sets the alarm by entering a key number into a control keypad. After this the system will detect any intrusion via the doors	Which doors specifically will the system monitor?	All external doors except the loading bay door
When will the alarm go off?	By default, 30 seconds after intrusion is detected (but this can be changed), unless someone prevents it	How would they prevent it going off?	By entering a different key within the 30 seconds (or configured time)

be from a 'user perspective'. This can be done using *birth to death scenarios*. For example an air traffic control system at a still but ready state receiving an inbound message, processing it and then sending an output to the air traffic controller, the system returning to a still state afterwards. Or if you are doing an integration test you may be trying to assess successful data flow, for example sending a message from component X to component Y and trying to assess a successful receipt of the message. You need to understand what is important to you.

Step 3: choose your test design techniques

Let's say that with past experience and your own knowledge you decide on a list of test design techniques. Others will come out of the analysis, but I'd prefer to go in with some intent rather than get pulled around in all directions and not test what is actually important to the business and the customer.

Test Requirements can be gathered using the following Techniques:

- user-based scenarios
- equivalence partitioning
- boundary value analysis

For information on the last two techniques see BS 7925-2, the Software Component Testing Standard.

Step 4: conduct interviews

By now you should have read any relevant documentation to hand (if there isn't any, it's not your fault) and will hopefully have built up at least some background on the system. You are now ready to interview project staff. Typical choices would be

- system analysts
- software engineers
- testers (with relevant experience)
- users (if possible)

System analysts will help you understand the product, how it is meant to function, who the users will be, and what problem the customer is trying to solve.

Developers will help give you an understanding of the inner workings of the product. They should be able to describe individual functions and what they are for. A good tip here is to create the right atmosphere for them. If you get them laughing and feeling good about themselves they'll even begin to tell you where the bugs are! Just be aware of the danger of getting dragged down too deep; keep thinking from the perspective of your test level (what you are intending to test) and what test design techniques could be relevant to you.

Other testers may have some experience with the product or related products, or an insight into what has gone right and wrong before in development and testing within the organisation.

Users will provide you with a wealth of information about how they intend to use the product, especially if it is an enhancement or addition to a product they are already using.

To summarise, the objectives of the interviews are to:

- identify what the System or product does. Get the interviewee to tell you about the product as a whole, from a high-level perspective. Ascertain the reasons the product is required and for what it will be used. Acquire schematics: ask them to draw the system, explaining as they go (even if you already have a drawing). Ask the same question of different people; a different viewpoint or explanation can work wonders for your understanding.
- establish the product's function or feature set - ie find out what are the items to be tested. You might start with a list of what the product comprises in its entirety and reduce this to obtain the test items. It is important here to keep asking yourself 'what is important to my team?'

- establish how the functionality should operate. Take the first test item and establish how it is meant to work, expressed according to the test design techniques you intend to use. For example, if the technique is user scenarios, you will try to ascertain what the users will be doing and how they would interact with the item.
- establish the risk associated with each function/feature set, to help you prioritize the test cases.

You have to play it as though you have just come down from Mars and you know nothing about the product. Let the information flow and allow them to talk, but remember to stop them at any point where you feel you have missed or wish to expand on something. Remember that you are only human and cannot obtain information using telepathy

The point is to start to generate as many test ideas as you can, with as much diversity as possible. Keep thinking until you feel you have done what you can in the time available. The tests will be rationalized later. Focus on the test idea, not the actual test. Test inputs or outputs are not required. Do not worry about possible overlap. Just keep the ideas coming.

Ask the interviewees questions such as:

- what is this feature supposed to do?
- how would I know if it did that?
- what would you expect the result to be if this happened?
- is there anything else I should know? about this?
- what could be the consequences if this feature failed?

Test requirements can be captured using lists, tables, etc.

Applying test design techniques *Again, this list is incomplete*

ID	Test design technique	Test requirement
TR1	User scenario	The system is enabled by the security guard entering the key number
TR2	Equivalence partitioning	Valid partition: system is enabled using correct key number
TR3	Equivalence partitioning	Invalid partition: attempt made to enable system using incorrect key number
TR4	User scenario	Entry is gained through an external door
TR5	Equivalence partitioning	Valid partition: entry is gained through an external door other than loading bay door
TR6	Equivalence partitioning	Invalid partition: entry is gained though the loading bay door
TR7	User scenario	Alarm is disabled using correct key number
TR8	Equivalence partitioning	Valid partition: alarm is disabled less than 30 seconds after external door is opened
TR9	Equivalence partitioning	Invalid partition: alarm is disabled more than 30 seconds after external door is opened
TR10	Boundary value analysis	Valid boundary: alarm is disabled at 0 seconds after external door is opened
TR11	Boundary value analysis	Valid boundary: alarm is disabled before it sounds at 30 seconds after external door is opened
TR12	Boundary value analysis	Invalid boundary: alarm is disabled after it sounds at 30+ seconds after external door is opened
TR13	Equivalence partitioning	Valid partition: correct key number
TR14	Equivalence partitioning	Invalid partition: incorrect key number

Conclusion

Test generation is a demanding task that requires skill. Sadly though, the process is widely misunderstood in the world of software development.


It should be emphasised again and again that good review and testability techniques injected into requirements at an early stage (subsequent to requirements capture) will help prevent a good number of defects from reaching the design, coding and testing stages. Freedman and Weinburg's *Handbook of Walkthroughs, Inspections & Technical Reviews* states that "...projects with a full

system of reviews report a *ten times* reduction in the number of errors reaching each stage of testing...(resulting in) cost reduction (of) between 50 and 80 percent, even when the cost of reviewing is added to testing costs."

The result of bad or no *product* requirements is that testers have to come up with ways to work and operate reasonably effectively until things can be improved, and attempt to stay sane at the same time. This article has explained one such way - generating *test* requirements - but 'working around' like this is not good enough and does not make business sense in the long term. That is why

we should request or even demand that our business creates a good test strategy and works in an agreed fashion. This will make everyone's life easier and, more importantly, help to ensure the customer gets more exactly what they need.

As a fellow tester I wish you luck in your endeavours to try and do an honest day's work in an industry that still has some way to go before being fully established as a quality conscious one.

The author would like to give special thanks to Don Ventress, Torbjörn Ryber and Simon Kiteley for their help with this article 

Deriving test cases

Test requirement	Input	Expected output	Test case
TR1 The system is enabled by the security guard entering the key number	Alarm enabled using correct key	Alarm is set	TC1
TR2 System is enabled using correct key number			
TR4 Entry is gained through an external door	Entry is gained through an external door other than loading bay door	Alarm starts 30 second countdown then sounds	TC2
TR5 Entry is gained through an external door other than loading bay door			
TR7 Alarm is disabled using correct key			
TR8 Alarm is disabled less than 30 seconds after external door is opened	Correct key is entered at 30 seconds after door was opened	Countdown is stopped, alarm does not sound	TC3
TR11 Alarm is disabled before it sounds at 30 seconds after external door is opened			
TR13 Correct key number			