

# Requirements and Test Plans – a Chicken and Egg Situation

Jerrold Landau, M. A. Sc. – Function Verification Test and Globalization Leader –  
IBM® WebSphere® Business Integration Tooling of the IBM Toronto Lab

This paper is the summary of a presentation prepared as a standby for the PSQT & PSTT East Conference, held at Washington Dulles Airport on March 23-24, 2004.

We have all heard of the rhetorical question: “Which came first, the chicken or the egg?” In reality, there is no answer to this question. The intrigue behind the question is that it forces us to realize that some questions in life have no real solution, and can be looked at from multiple perspectives.

In the field of software testing, it is taken for granted that test plans are developed based on work items such as product requirements, use cases, and detailed designs. Yet all software testers with any experience in the field know that it does not always work out that way. Often, the requirements are not of sufficient quality or detail upon which to base a test plan. This presentation will examine the causes of such a situation, as well as the ramifications and ways of coping. We will discover that there are even scenarios where the situation of the test plan preceding requirements may be appropriate.

The situation is described as a 'chicken and egg' situation rather than a 'cart before the horse' situation -- one can never put the cart before the horse if one wants to proceed, but the test plan can indeed come before the requirements..

Looking at the theory behind the software development process from a high level, there is a unidirectional flow from requirements, to the test plan, to the test cases, to the test execution. In an ideal situation, there should be no backwards feedback between these phases. Everything should be in place at the preceding stage before proceeding to the next stage.

However, in almost all real life testing situations, there is some amount of backward feedback between the phases. In general, this should be minimal, but on occasion, it can be significant. For example, the act of writing a test plan often helps to solidify and verify the requirements. Consequently, as a result of test planning, the requirements may change. In a sense, the writing of the test plan 'tests' the requirements. Of course, whatever changes are made to the requirements should feed right into a further iteration of the test plan. Similarly, when writing the test cases, one often noticed gaps in the test plan,

In the same context, when writing the test cases, the tester may be inspired to think of additional cases not covered by the test plan, or may realize that certain parts of the test plan are not feasible in a practical sense. Thus, the test case writing stage can certainly feed back to the test plan. Likewise, the actual execution of the tests can certainly point to the need for more test cases, or the unfeasibility of some test cases, and can thereby feed back to test case writing. By induction, any of these backward feedbacks can go back further, such that the running of test cases can have ramifications that feed back all the way to the product requirements.

If formal documents are kept for all of these phases (which they should be), it would be appropriate if the feedback is incorporated, such that test plans and even requirements are updated due to feedback from test execution. In practice, this rarely happens to the extent that it should, resulting in the documentation at the various stages getting out of synchronization.

## **Causes of Lack of Requirements Situation**

Let us now focus on the causes of the "Lack of Requirements" situation. There are cases where the product requirements are very scanty, or in the worst case, nonexistent. In such cases, if testing is planned for the product, the test plan will have to be built in the absence of requirements. A careful construction of the test plan may actually result in the test plan doubling as a formalization of the requirements. This poses some challenges, though, which must be anticipated and accounted for to ensure the success of the product.

There are several potential causes for what I will term, the "Lack of Requirements" problem. Of course, the most obvious cause is a sloppy development process. A product may be produced without proper process, which would dictate that full requirements be documented. This product would then have to be tested, without the benefit of knowing exactly what is to be tested. This is a particularly dangerous situation in a mature product, which is beyond the first release -- as tampering with an existing product without thinking through the ramifications can have disastrous results.

In other cases, or perhaps overlapping with the above, a working prototype may be demanded very quickly, leaving little if any time to document formal requirements. This is fraught with dangers, and the best recommendation in such a case would be to go back and formalize the requirements after a preliminary prototype is produced. However, real life contingencies often dictate that a planning stage will not take place after the fact.

A third situation, very common as well, is that the product design continues to evolve during the development cycle. Thus, the requirements represent a first pass of the product design, but become increasingly less valid as the release progresses. New features may be added, without going through the formal product development cycle. As seasoned testers, we have all surely seen examples of this during our careers. Of course, in such situations, the requirements should be updated after the fact.

Another situation where there might be a lack of formalized requirements occurs when an iterative prototype development cycle might be initiated. This would occur most often with a first-release product, which might be experimental in nature. It might be possible that the ultimate intended consumers do not know exactly what they want. As bad as this sounds, in a trend setting-company such as IBM, this certainly does occur with some frequency. It is not for us to comment on the legitimacy -- there are legitimate market factors which make this paradigm the reality. In such a case, the early prototypes really are the requirements. The test plans will then be written based on the early prototypes.

There are cases where the product requirements are defined in terms of an intended low-level result, leaving the usability (which is what customers ultimately care about) up to the developers. I am currently working on such a product. The WebSphere Business Integration area is developing a business view editor and a simulator. Standards exist as to how the data should be stored at the XML level -- however the entire usability layer is left up to the discretion of the development team, with input from usability experts and use of various precedent products as models. The developers write unit tests to ensure that their code produces the required XML. The customer layer, which dictates how a process is represented in the editor and then sent to a simulator for simulation, is what the function test focuses on -- and it does so without the benefit of formal requirements. The customer views are put together using iterative prototypes..

Is it legitimate to write test plans without the benefit of formal requirements? The question is moot in practical situations, as there may not be any choice in the matter. Reality dictates how to proceed. However, the question is worth pondering in theory. In most cases, this is not the correct way of going about things. Surely in cases where the development process is lacking, and where time is so crunched that the planning is skimmed upon, the situation is far from optimal. The tester will simply have to do the best they can, making the best of a non-ideal situation -- and hopefully making efforts to prevent such from occurring in the future.

However, in the situation of development by iterative prototype, this may be legitimate. The product might be 'planned' by iterating the prototypes, and the test plans would be developed in an iterative fashion. In fact, the test plan would then become the working version of a formalization of the requirements.

In my own case, where there is a definition of the low-level standards without the usability -- since the situation is also one of iterative prototypes, it is legitimate to go ahead and produce test plans based on the prototypes.

## **Risks of Writing Test Plans without Proper Requirements**

Writing test plans without the benefit of formal requirements obviously comes with risks. There is the risk of subjectivity, in that the testers use their own sense of what is correct and incorrect (in such a case, the testers assume too much power). On the other hand, there is the risk of testing by simply documenting the results (assuming too little power), which is merely a confirmation of behavior and not really testing at all. These two risks are at opposite ends of the spectrum. In addition, there is the real fear that the customers' desires have not been formally introduced into the process.

There are ways of coping with these risks. The tester may have to assume the role of the planner to some degree, as the test plans can be regarded as a formalized instance of the low-level requirements. The tester should also compare and contrast various parts of the tested application, in order to verify consistency. Most important, the test plan should be thoroughly reviewed by all interested parties -- including planners, product owners, developers, and usability experts -- to ensure that it represents a valid view of how the product is expected to function. A test plan review is important at all times, but it is especially important when the precursors to the test plan are inadequate.

The situation of not having proper requirements might be suboptimal in most cases (not all cases). However, the test team has no choice but to accept responsibility. Standing on ceremony and refusing to accept responsibility will only make a bad situation worse. Of course, if the test team can make constructive recommendations to improve the situation, they should do so. Even in this type of situation, a properly constructed test plan can add significantly to the ultimate success of the product. In the best case, if the test team plans its actions in a careful and appropriate manner, they can end up salvaging a release.

## **Outcomes of Testing without Proper Requirements**

There are three possible outcomes when testing without proper requirements. The first two are process based. If the situation is such that the process should be fixed, the test team might either succeed in ensuring that this is done. On the other hand, they may not succeed in convincing the larger development team to produce proper requirements documents. The third scenario is where the lack of requirements is actually valid -- the development by iterative prototype situation. In all three cases, if the test team conducts its work carefully and appropriately, they can end up making a significant contribution towards the quality of the release. This is what I term a 'win win win situation'.

If the process is broken, and the test team actually manages to convince development to fix the process and ensure that proper requirements are produced prior to the beginning of the development phase, the problem will of course disappear. It may take several releases until the problem phases out completely. Of course, the product manager might ask such an astute tester to become the product planner -- in fact, there is much overlap in the skill required to produce a requirements document and the skill required to produce a test plan. The testers will have become the heroes of the release.

If the process should be fixed, but cannot be (either because of lack of buy-in, or lack of appropriate resources), the testers must do their best to test without the benefits of formalized requirements. The test plan will have to be carefully crafted, and all the caveats mentioned earlier in this presentation will have to be dealt with. The test plan might end up becoming the formal version of the requirements. In any case, a carefully planned and executed test can salvage such a product release from complete disarray, and once again, the testers are the heroes of the release.

In the case of development by iterative prototype, as we have discussed, the absence of formal requirements might be perfectly legitimate. In such a case, the test plan should iterate with the prototypes. A carefully executed test plan, subjected to appropriate review, can formalize the requirements of the prototypes. The testers must realize the full weight of their responsibility in such a situation. Once again, the testers might be the heroes of the release, but if the product release runs smoothly, perhaps because of the "value add" of the testers, the testers might not be recognized as such.

## **Summary**

In summary, there are times when test planning occurs without the benefits of appropriate formalized requirements. This may or may not be the way things should be -- but it certainly is the reality. Testing must proceed with extreme care and caution, with the testers realizing the full extent of their responsibility. However, in all such cases, whether or not the process is appropriate, the test team can add a great deal of value to the release.

## **Copyright and Trademarks**

Copyright: International Business Machines, Corporation, 2003. All rights reserved.

IBM and WebSphere are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based logos are trademarks of Sun Microsystems in the United States, other countries, or both.