Testers Learning Requirements

Hans Hartmann

OBJENTIS Software Integration GmbH
Vienna, Austria
hans.hartmann@objentis.com

***Abstract* — When software testers talk about their job, they generally state that their task is to find defects as well as to ensure that no (major) defects exist in the main paths of business processes. They often add the comment they are doing this by checking the software (application under test) against existing requirements. Its common sense that testers should be familiar with understanding and interpreting requirements. But when it comes to creation and review of requirements testers are usually not involved. Now, taking into consideration that many errors in software development are due to improperly defined requirements, one might wonder if integrating testers directly into the phase of requirements creation would be a promising option.**

This lecture reports about a tester-specific training-program that directly touches that consideration. The training was held in practice for testers in the field of financial services in the year 2013 and 2014. It highlights the structure of the program as well as its benefits and practical value in the software development cycle. In addition commonly made shortcomings and their solutions are also covered.

***Index Terms* — requirements, software testing, requirements-based testing, training**

I. Introduction

According to Glenford Myers [6] software testing is concerned with finding errors. While this certainly is a true objective, it begs the question: How does one find errors? (Or defects regarding to today's etymology) Testers who have had some training, including certification training, will answer that one main method they apply is to check the requirements.

If we agree with this answer, we must decide whether the requirements provide a fully comprehensive source for deciding whether errors exist in a software product.

This paper is a practitioner's report from within the industry. From our consulting experience in more than 100 companies, (mainly in the field of financial services and public services) at least 50% of the requirements we have seen were insufficient for proper assessment when testers examined the product. (Typical examples would be the requests that the program should respond faster than its predecessor without supplying information about the expected count of users.)

In this paper we point out why it is difficult to write correct requirements. We then present a case in which special consideration of requirements handling was included in a training program for testers, with quite favorable results. Subsequently, we argue that it is necessary for testers to deal with erroneous requirements when using them to define test cases and general testing procedures. We also suggest that, to cope with deficient requirements, testers need a basic understanding of how to assess the correctness of requirements.

II. The difficulty of writing good Requirements

Before we can agree on what other people want, we must know what we ourselves want and what we want to communicate. A good place to start is to read the book *Writing Better Requirements* [1], whose key message is to "get agreement on what people want before attempting to create solutions."

This sentence is useful and can also be related to *Allgemeine Psychologie der Kommunikation* [9], whose author suggests that anyone desiring to write good requirements must be clear about what they wish to communicate – "Klare Kommunikation setzt Selbstklärung voraus" (Clear communication requires "self-clarification") [9].

Knowing what someone wants is not as easy as it seems. The reason for the difficulty may be illuminated with reference to:

- Fairy tales and legends
- The necessity for abstraction.

A. Fairy Tales and Legends

The famous King Midas is popularly remembered in Greek mythology for his ability to turn everything he touched into gold (chrysopoeia). This came to be called the golden touch, or Midas touch. According to Aristotle, legend held that Midas died of starvation as a result of his "vain prayer" for the golden touch.

In "The Poor Man and the Rich Man" [5] by the Brothers Grimm, we find the common fairy-tale situation of the rich man using three wishes to no avail, with the last wish needed to counteract the ill-advised wishes that preceded it. There are several tales that follow this same convention, so what to wish – and how to wish it – seems to be a long-running bestseller.

Today, we find many books about improving one's life through proper planning for the future, and by setting the right goals in life. Indeed, it should be easy to formulate a wish for one's own future, but unfortunately a wish is not complete without considering how to avoid potentially adverse consequences. For example, one may wish for a successful career but, in doing so, may be jeopardizing one's marriage with the many hours of overtime and extensive traveling.

B. The Necessity for Abstraction

Writing requirements for software products confronts us with an additional challenge. Most people have not acquired the skill necessary for dealing with high-level abstraction, to mean describing something that does not yet exist requires a certain amount of imagination. With hardware it seems relatively easy to define specifications. Physical dimensions can be supplied as absolute values and the definition of tolerances allows for deviations from nominal values. Thus, when writing requirements for a car, we can clearly state that the car must have four doors. But if the car were a piece software, we would likely write that the number of doors will be decided later, at the time when the car is designed.

A historical example might help for further clarification. In 1976, the author was developing electronic circuits. If the circuits were composed of discrete elements, every element was defined with its electrical value, its mechanical dimensions and sometimes even with the name of the manufacturer. A few months later the electronic measuring device was re-designed to incorporate a microprocessor. Suddenly, certain decisions were postponed: "The final decision will be handled by software later on."

Today we find a similar behavior when the requirements engineer – who, in many financial services companies, is often a business domain specialist – addresses the developer. The specialist writes an approximate description, expecting the developer to program what is feasible, and the tester is typically not consulted in the design of the requirements. It is therefore no surprise if the message becomes distorted, for the sender is not expressing the wish in a manner that will be clearly understood by the receiver.

C. The Difficulty in Writing Good Requirements

Writing good requirements requires training. In many companies whose testing effort we have supported, the requirements were written by the best specialists in the business domain. Certainly, a clear understanding of the business is necessary to write good requirements. However, the need for communication and technical writing skills is often neglected. Although courses for technical writing and even requirements writing exist, business domain specialists usually do not receive training in this key area. Moreover, a number of companies are cutting training programs these days – companies in Austria and Serbia, for example, although they are probably not alone.

III. Requirements Training for Testers

A. Tasking and Conditions

We were tasked with training recently hired employees of a big insurance company to prepare them for implementing a new test center. A group of eight people were trained in practical testing and the language used during training, and in the work environment, was English. All participants were non-native English speakers, a fact that is important when considering that language is one of the keys to understanding requirements. Seven trainees had an academic degree, while one was still studying. The participants had previously undergone a formal International Software Testing Qualifications Board (ISTQB®) training, including certification at the foundation level.

The later training described here contained among other topics the interpretation of requirements. One week was devoted to this topic. The topic was divided into six instructional steps:

Step 1: Writing requirements from scratch
Step 2: Completing requirements from a given example
Step 3: Reviewing the requirements
Step 4: Reviewing "real" requirements
Step 5: Re-writing the requirements
Step 6: Deriving a test case list

B. Step 1: Writing Requirements

A short discussion led to a sample application that everybody in the group was familiar with. The requirements had to describe a program that could calculate the working hours of employees using the supplied data, such as the time present, breaks, project assignments, etc. All participants confirmed that they understood the objective of the program.

As a group, the participants worked together for four hours spread over two days. They were able to consult the Internet to complete the task. Nevertheless, their efforts produced no results. Despite lengthy discussion, the group was not able to agree, although the discussion did help build the group into a team. At the time of the assignment, the participants had no formal background on how to phrase and express requirements.

The challenge: “sit in front of a blank paper and start writing”

C. Step 2: Completing the Requirements

The instructor gave a background explanation of use cases and mental models for the operation of the software. He presented a typical requirement and asked the participants to complete the written description, for which the trainees were separated into two groups. The teacher answered questions about what the program should do, but supplied no formal support. Completion of the task took one day.

The documents produced by the sub-groups contained a partial description of the program - with some omissions and inconsistencies in the various requirements. The participants conceded that writing proper requirements was quite difficult.

The challenge: “develop a formalism to write testable requirements”

D. Step 3: Reviewing the Requirements

Following instruction on the formalities of structuring and enumerating requirements, each group was asked to review the requirements of the other group. A list of formal inspection was supplied (based on Software inspection [4]). This task was done in half a day and resulted in two documents. One of the documents was a set of requirements that would suffice to design and program a prototype of the application.

The challenge: “apply formal rules to a requirements document”

E. Step 4: Reviewing the "Real" Requirements

Two different groups were formed and a "real" requirements document from the current project's development cycle was reviewed. The document contained two change requests that were to be implemented in a car insurance administration program. The first sentences were analyzed by the teacher and some errors were identified. Typical forms of errors according to Hartmann [6] were provided to the participants, who found some missing descriptions as well as ambiguities within two hours. The results led to Step 5.

The challenge: "scrutinize a paper of requirements to decide upon formal compliance"

F. Step 5: Re-Writing the Requirements

The participants – again working in two groups – rewrote the specification of one of the change requests. Their task was to precisely write the structure and enumerate the requirements, including their additional contributions, thus enabling the requirements to be tested. Ambiguities that would require interaction with the original authors were to be marked. If possible, a favored interpretation of each ambiguity was to be inserted as a proposal.

Half a day was needed for this exercise. The resulting document was 20% longer than the original requirements. It had a clear structure and was a suitable source for Step 6. It contained two requirements (instead of the original one) each containing around 3 paragraphs to be checked independently. Each paragraph had a unique ID.

G. Step 6: Deriving a Test Case List

Using the documents of Step 5, each group was asked to write a list of test cases for the document created by the other group. One group found 41 significant (non-redundant) test cases; the other group listed 39 different test cases.

This result (the count of necessary test cases) was presented to the head of test management for review. It was confirmed that all test cases were relevant for the actual business application. It was also conceded that the current test portfolio might have a maximum of only 10 test cases. Available time for testing and the necessary ranking of test cases could also result in the actual testing of not more than 10 test cases from the list of 40 that had been identified. But the depth of investigation did impress the head of test management. While this was the result of a training session, it has been shown that the execution of Steps 4 to 6 can be managed within reasonable time as part of the daily work routine, by no more than two testers.

H. Additional Considerations

Using the method of re-writing the requirements seems feasible for typical change request descriptions. However, for a long list of requirements that describe the first version of a product, testers could not re-write all requirements – nor are they expected to do that.

The test management team must consider the number of expected errors as relevant for planning test resources and time slots. Various studies have shown that, in the year 2000, errors in the requirements and design phase added up to 43% of all errors in a project [3]. Current statistics from 2012 show values around 45% (IBM), 50% (SPR), 60% (TRW), 64% (MITRE) and 60% (NEC), based on industry data on defect origins [7]). If the requirements have been properly reviewed, then testers can expect the found errors to have been repaired. In this case, an independent test following the coding phase would find between 20% and 25% of all errors in the project.

If the requirements have not been properly reviewed, independent testing will typically find three times as many errors. This will adversely affect the time needed for testing, as well as the time required by developers to fix the errors.

Given the above, the question arises: Why are requirements generally not reviewed properly? In the original waterfall model and in the V-model, the development process relies on correctly reviewed requirements. In practice, it seems that the prevailing opinion in many companies is that only business domain specialists can review requirements properly. However, whereas domain specialists tend to detect specific errors connected with the business per se, checking for omissions and contradictions, as well as comprehensibility, are activities that demand the destructive approach used by a professional tester.

We therefore argue that testers should be employed in the process of requirements engineering and review, as this also makes economic sense ("Table Normalized Cost-to-Fix Estimates" [10], COCOMO [2]).

IV. Summary

We started by mentioning the difficulties in writing good requirements. Therefore, testers who deal with requirements need to consider their own possible flaws in this respect. In turn, the impact of training in writing and reviewing of requirements was shown to be favorable. Not only did it prepare the participants for subsequent testing tasks and projects, but project leaders later commented on the trained testers' insight in business domain-related topics, in which the latter were not specialized. This shows that formal assessment and increased knowledge regarding the potential errors in requirements had helped further the participants' understanding of the essential goals of the software products in question.

These results imply that it is highly recommended to employ appropriately trained testers in the requirements engineering and review phase. They will surely find more errors than business domain specialists are able to find – earlier and at a lower cost of repair.

References

[1] Ian Alexander, Richard Stevens and Ralph R. Young; *Writing Better Requirements*; Edinburgh Gate: Pearson Education Ltd, 2002.

[2] Barry Boehm et al.; *Software Cost Estimation with COCOMO II*; Upper Saddle River NJ: Prentice Hall, 2000.

[3] Frederick P. Brooks, Jr.; *The Mythical Man-Month: Essays on Software Engineering*; New York: Addison-Wesley, 1975.

[4] Tom Gilb and Dorothy Graham, *Software Inspection*, Edinburgh Gate: Pearson Education Ltd., 1993.

[5] Jacob Grimm and Wilhelm Grimm; *The Fairy Tales of the Brothers Grimm*; San Diego: Canterbury Classics, 2011.

[6] Hans Hartmann; “The even darker side of creativity;” research report for the 19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2013). Proceedings of the REFSQ 2013, Berntsson Svensson, Richard (Ed.) et al., Essen: University Duisburg-Essen, p. 41, 2013.

[7] Capers T. Jones; “Software quality in 2012 – a survey of the state of the art;” presentation at sqgne.org: p. 20, 2012.

[8] Glenford Myers; *The Art of Software Testing*; New York: Wiley, 1979.

[9] Friedemann Schulz von Thun; *Miteinander reden 1: Störungen und Klärungen*; Reinbek: Rowohlt Taschenbuch Verlag, GmbH, p. 113, 2005.

[10] J. M. Stecklein; “Error Cost escalation through the project life cycle;” NASA Johnson Space Center: p. 2, 2004.