Software Functional Testing from the Perspective of Business Practice

Mingtao Shi
FOM Fachhochschule für Oekonomie & Management
University of Applied Science
Bismarckstr. 107, 10625 Berlin, Germany
Tel: 49-171-2881-169 E-mail: Consulting_Shi@yahoo.de

Abstract

Industrial firms build their respective value chains by purchasing and tailoring core software to their specific business scenarios. Software functional testing is an indispensible element in the software implementation cycle. To achieve the two major aims of the functional testing, namely effectiveness and economics, business specialist, requirements analysts and the testing team should work together closely. To perform the functional testing, business requirements must be extracted and a number of recommended black-box testing methodologies should be used. Successful testing activities should enlist the support from test plans and test tools.

Keywords: Black-box testing, Equivalence-Classes Partitioning, Boundary-Value Analysis, Error Guessing, Process, Product, Parameter, Test cases, Testing plan, Test tool

1. Software Functional Testing: Some Important Theoretical Aspects

Software testing is an essential part of the software development process. The importance of software testing can simply not be overestimated in the business practice, if the quality of the system production is to be ensured. Effectiveness and economics are the two most important aspects for selecting testing methodologies. While effectiveness means that the designed test must be able to unveil the maximal number of errors residing in the software, economics implicates that the test itself should consume as little time and resource as possible.

Testing strategies and tactics have been discussed extensively in the literature. The authoritative work in the field published by Myers (2004) suggested the white-box and black-box testing for software functionalities. The major white-box tactics include statement coverage, decision coverage, condition coverage, decision-condition coverage and multiple-condition coverage. The most vital black-box testing for Myers consists of Equivalence-Classes Partitioning, Boundary-Value Analysis, Cause-Effect Graphing and Error Guessing. Burnstein (2002) also recommended black-box methods such as Cause-Effect Graphing and Error Guessing. Ammann and Offutt (2008) contended that the so-called Input Space Partitioning may be seen as a useful methodology, which is, by nature, a combination of Equivalence-Classes Partitioning and Boundary-Value Analysis. Although Bath & Mckay (2010), Pressman (2009) and Patton (2006) also looked at other black-box testing tactics, Equivalence-Classes Partitioning and Boundary-Value Analysis have again been recognised as fundamental and widely used black-box methodologies.

The analysis of Equivalence-Classes Partitioning examines the input conditions contained explicitly or implicitly in the Software Requirements Specification (SRS). These input conditions are partitioned into a number of valid and invalid equivalence classes. Test cases are then generated for each class. The assumption here is that the test case designed for one class is representative for all other input values within the same class. The charm and the reason of the wide use of this methodology lie in the fact that by deploying a small number of test cases, a large range of input conditions can be effectively covered. Boundary-Value Analysis, in contrast, focuses on the edges of equivalence classes. Values slightly above and beneath these edges may also be critical and should also be checked. In certain situations, not only the input boundaries but also the output boundaries should be taken into account while designing test cases. Cause-Effect Graphing considers the combination of input conditions and delivers a mathematical-logical “roadmap” of the software functionalities. The combination of inputs and outputs are carefully analysed and derived, which forms the basis for test case design. In addition to these relatively formal testing techniques, it is in many cases surprisingly effective to contact persons who are experts of either the system specification or the business applications. These experts are normally capable of devising test scenarios that have not been thought of or covered by the rather mathematical-logical testing techniques. The added value of these experts resides in their profound experience, tacit knowledge and intuition.

2. Software Functional Testing in Business Practice

The business reality today is that software systems have become pervasive in virtually most industries. Most industrial firms are primarily interested in applying software and only few software houses are “making” the software. Under such circumstances, industrial applications based upon core systems that can be individualised by parameterisation and customisation have emerged over the past decades. Industrial firms are not producing core, the core system is rather delivered by specialised software houses such as SAP. Industrial firms are more concerned with using the functionalities made available by core to build their respective value chains. Theses firms are, in terms of technical competencies, usually not capable of comprehending the source codes in the core system.

The implication for the practice is fairly clear. Since it is hardly possible for firms to touch the source codes of the purchased core system, the white-box testing is much less useful than its black-box companion. Black-box testing has become much more common in the business practice. The most widely used black-box testing methodologies are *Equivalence-Classes Partitioning, Boundary-Value Analysis and Error Guessing*. Although Cause-Effect Graphing may incorporate Boundary-Value Analysis, to use this technique to design test cases has turned out to be extremely time-consuming in business practice.

Certainly, systemic software testing should take place at different levels, including module testing, integration testing, functional testing, system testing and user acceptance testing and encompass tests against non-functional business requirements, such as availability, compatibility and usability. This paper concentrates solely on the realisation of functional testing in industrial context.

Notably, the general implementation cycle of software systems in industrial firms follows, at least to a significant degree, a general observable pattern: 1) familiarisation of the core software and its functionalities; 2) investigation, repeated examination and determination of business requirements; 3) documentation of business requirements; 4) feasibility studies based upon business requirements specification and design of possible workarounds for non-core business requirements; 5) parameterisation and customisation at the application level of the core system; 6) functional testing against specified and documented business requirements; 7) debugging and regression tests; 7) delivery, maintenance, continual updates and releases. Obviously, functional testing is a crucial phase, reconciling the work results stemming from requirements and development phase of the software implementation cycle.

The specialist team performing the software functional testing is usually faced with two major constraints existing in the business practice. First, time and budget are always in shortage. Second, for firms not specialised in the software development, fine-grained specification documentation is most probably non-existent. It is virtually impossible, especially for small and medium-sized firms, the main contributors of the economy in Germany, to allocate significant funds and resources to external or internal software specification engineers as new businesses and products are added to the firm’s profit portfolio. Lack of information and documentation is a general difficulty for software implementation projects which will continue to exist.

3. Software Functional Testing: A Practical Approach

This paper suggests a practical approach, which has proven to be valuable in the business practice for functional testing. Although comprehensive documented business information is rarely available in most small and medium-sized firms, a minimal set of essential business information must exist or be built for comparably smother software implementation tailored to business reality. This set of information must be made available to both developers (development at the application level) and testers in a shared network environment and changes to the information must be managed in such a way that the changes are approved by the management and the implementation team is informed about the changes.

This minimal set of business information should at least cover a number of informational areas. First, business *processes* are to be mapped, including the description of functional roles, process steps and detailed activities of the respective process steps. System-related steps and activities must be looked at in more detail, including system inputs, outputs and printouts. Process mapping should be conducted from less to more detailed informational levels. Second, *Products* must also be documented. The definition of the products becomes unambiguous when their functionalities and features, internal and external interfaces, related ancillary products and reports are rigorously clarified. Third, *user access rights* in the system, process-related and product-related *parameters* to be applied in the system should be reviewed and determined precisely. Furthermore, System implementation tailored to firm-specific business requirements consists of complex activities that can only be effectively and efficiently (in terms of budge & time) managed if a proper *project management* environment is in place.

Although this minimal set of business information may provide considerable aid, the richness of information is, in most cases, not sufficient for application developers to implement the system straightaway. The requirements analyst needs to play a particularly proactive role, translating between business and IT staff, and be involved in both application development and functional testing. Based upon the requirements documents made available, the analyst and the business specialist may begin with the delineation of the *system screens* and *data fields* needed in the respective business applications. Data fields are normally categorised in mandatory and optional fields. The categorisation, certainly, must be in line with the documented business needs. The content of the data field can be presented in multiple ways, such as read-only field, input field, dropdown field, combination of dropdown and input field, and calculated field on the basis of the content of anther data field.

Once the screens and data fields in the system are considered and designed, the testing activities can begin. Drawing upon the business information stated above (process, product and parameter), the testing specialists can compare the business requirements with the screens and data fields to scrutinise the compliance and realisation degree of the business scenarios. The discussions between the analyst and the testing team may sometimes also result in a more detailed *fine specification*. In business practice, description of business information can contain intriguing implication for specification and thus, for the derivation of test scenarios. Identifying a customer, for example, is a piece of business information, from which at least two system implications can be deciphered: the system must perform a customer search function at the corresponding business step and the search criteria (e.g. according to family name, date of birth or other unique characteristics of the customer) must be defined. Either the documented business information can be used directly to devise test cases or the fine specification needs to be formulated on the basis of documented business information for the derivation of the relevant test cases.

In order to consume as little time as possible but still be highly effective, it is suggested to emphasise and apply *Equivalence-Classes Partitioning, Boundary-Value Analysis and Error Guessing* as the main testing tactics while constructing test cases. Test cases need to be categorised in a structural way, representing major business scenarios. In a University Intranet System, for example, possible major test categories may encompass "communication among lecturers", including test cases in sub-categories such as email platform, appointment calendar etc; "communication between students and lecturers", including test cases in sub-categories such as email platform, download & upload mechanism for teaching materials and assignments etc; "examination area", including test cases in sub-categories such as submission of written examinations by the lecturers and communication of grades etc.; "class schedule for students and lecturers"; and, "room occupation plan". Such *Functional clustering* is important for testing design in business practice. Wide functional test areas are split into less wide system functionalities, which, in turn, are further broken down into narrower test cases that usually focus on individual functionalities. For each identified system function testing specialists can apply the above recommended test tactics to obtain meaningful test cases. These testing methodologies are much discussed and widely acknowledged in the literature as highly effective for discovering errors and disorders in software systems. Furthermore, from the perspective of business practice the design of test cases based upon these methodologies is much less time-consuming than almost all other testing philosophies. Once the test cases are in place, *test steps* must be defined, describing how the test cases are to be carried out step by step. Notably, the testing team should also consider and document the associated test cases while looking at a certain test case. Test cases or system functionalities are interlinked with each other and mapping the *interrelatedness* of test cases can be beneficial for systematic tests in most cases.

Although business requirements and test case design are salient, the software functional testing however, is composed of a set of other surrounding activities, ensuring the smooth testing operation. Myers (2004) has explained these activities in great detail, stating that an elaborate *testing plan* should also consist of the definition of completion criteria, scheduling of test execution, definition of responsibilities during the functional testing, test case storage and administration, clarification of hardware availability and configuration necessary for testing, error documentation and debugging procedures, and planning of regression tests. Furthermore, it is worth mentioning that the *test automation* is highly coveted in the business practice. However, except the simplest system features, trying to automate the functional testing can become an extremely time-consuming and resource-stretching endeavour for small and medium-sized firms. A sound mixture of manual and automated test cases is a better choice in most cases than striving for a high degree of test automation. A non-IT firm may need an extensively long time period to assimilate knowhow and accumulate experience in the area of test automation can take place.

Firms performing functional testing at the application level should also make use of *test tools* for efficiency and administrative purposes. There are numerous open-source freeware or license-based tools for functional testing available in the marketplace. No matter which tool is chosen, a number of vital but often overlooked issues must be dealt with effectively:

- The tool should be able to store and change business requirements (fine specification);

- The tool should able to categorise test cases according to business functions;
- The tool should be able to document the interrelationship among business requirements and test cases;
- The tool should be able to protocol test results and optimally contain an automatic interface to the tool used by the development team (communication, debugging and regression tests).

4. Conclusion

Industrial firms build their respective value chains by purchasing and tailoring core software to their specific business scenarios. Functional testing is indispensible for virtually all software migration and implementation projects. Such projects in business practice permanently experience the lack of time, budget, and detailed documentation. Under such circumstances, as this paper suggests, a practical approach can be adopted to relieve or even partially overcome these constraints. The requirements analyst should make available a minimal set of documentation, such as descriptions of related processes, products and their parameters. Subsequently, system screens and data fields are to be specified. As the testing activities unfold the discussions between the analyst and the testing team sometimes also results in a more detailed fine specification. Based upon the documented business information or the fine specification test cases can be formulated. Black-box methodologies such as Equivalence-Classes Partitioning, Boundary-Value Analysis and Error Guessing have become proven testing methodologies for devising effective and economic test cases in the business practice. Functional clustering, mapping of interrelatedness of test cases, an elaborate testing plan, a sound mixture of manual and automated test cases and the use of test tools can add more value to the software functional testing.

References

Ammann, P., & Offutt, J. (2008). *Introduction to Software Testing*. New York, NY: Cambridge University Press.

Bath, G. & Mckay, J. (2010) *Praxiswissen Softwaretest - Test Analyst und Technical Test Analyst: Aus- und Weiterbildung zum Certified Tester - Advanced Level nach ISTQB-Standard.* Heidelberg: dpunkt Verlag.

Burnstein, I. (2002) *Practical Software Testing*. New York, NY: Berlin/Heidelberg: Springer Verlag.

Myers, G. J. (2004). *The Art of Software Testing*. (2nd ed.). Hoboken, NJ: John Wiley & Sons, Inc.

Pressman, R. S. (2009). *Software engineering: A practitioner's approach*. (7th ed.). New York, NY: McGraw Hill.

Patton, R. (2006). *Software Testing*. (2nd ed.). Indianapolis, Indiana: Sams Publishing.