Test Optimization using Risk Based Testing Approach (RBT) in Software Engineering

Sreenivasa Pisupati, Senior Vice President, W3Softech India Limited.
sreenivasa.pisupati@w3softech.com

Abstract

This paper describes “How to Optimize Testing” in software Engineering. With the growing constraints on the budgeting of the software, there is a huge need for reducing the effort spent on various activities in a Software Development Life Cycle(SDLC). Among all the activities in the SDLC, testing seems to an activity with huge scope for reducing the effort by using optimized testing techniques such as Risk based Testing Approach(RBT). By following Risk based Testing (RBT) Approach, effort spent on testing can be reduced by more than 40%.This approach ensures adequate test coverage with optimum test effort and cost. The testing efforts are prioritized towards executing test cases that have high probability of failure and the impact associated with these failures is high as well.

Introduction

Various analysis carried out in the industry reveal that, more than 30 percent of the overall effort for software projects are spent on testing. In spite of spending 30% of the overall effort, confidence in the system being tested is not increasing eventually. However spending effort on testing is very much necessary, but the test cases designed and selected for execution make the real difference. There is no point in endlessly executing the test cases that can neither uncover the bugs in the system nor increase the confidence in the system. Risk is an important consideration when managing a project. Controls are the means used by organizations to minimize risk. Software testing is one such control that can be used to minimize or reduce the risk that information used in making business decisions will be incorrect or late. The purpose of the testing control is to provide management with the information they need to

Hence the actual essence of testing lies in designing the minimal set of test cases which can uncover the maximum number of bugs in the system and at the same time gives a comfortable feeling about the quality of the system. Risk based Testing approach enables to identify the test cases that are highly risk prone and have the high probability of uncovering maximum number of bugs..Features to be implemented in a system and the corresponding test cases used to evaluate those features should be ranked according to the estimated risk magnitude. High risk features should be tested early, while lower risk items can be deferred to later test cycles.

Guidelines for Assessing and Managing Risk in Testing

The following guidelines describe how risk can be used to guide the testing process.

Testing should minimize risk with a minimal amount of effort. Risk analysis is used to identify the features or other factors affecting a system or project that present the most risk to the organization. When testing a complex system, emphasis should be focused on early testing of those features that have the highest risk. The highest business risks should receive the most test resources, while the lowest business risks should receive the least amount resources. High risk items that are identified early can be mitigated sooner while there is time to do something about it. Furthermore, when time is at a premium and sacrifices in testing must be considered, then only those items with the lowest risk will remain to jeopardize the project.

Risk can be characterized using the following two attributes:

- **Probability of occurrence (frequency)**
- **Impact on the project (severity)**

The two risk attributes can be combined into a single **risk magnitude indicator,** usually derived as the product of the two risk attributes. Thus, probability of an outcome (0 to 100 percent) times the impact or severity will yield a single risk magnitude indicator that can be used to compare the risk of one situation versus another. Impact may also be quantified using a dollar amount should the potential risk be realized or a number to subjectively quantify the degree of impact, such as 1 for low impact and 10 for high impact.

Features to be implemented in a system and the corresponding test cases used to evaluate those features should be ranked according to the estimated risk magnitude. High risk features should be introduced early and tested early, while lower risk items can be deferred to later test cycles.

Test Factors

The objective of testing is to reduce the risks inherent in computer systems. The test strategy must address the risks and present a process that can reduce those risks. The system concerns or risks then establish the objectives for the test process. The two components of the testing strategy are the test factors and the test phase, defined as follows:[i]

- **Test factor** – The risk or issue that needs to be addressed as part of the test strategy. The strategy will select those factors that need to be addressed in the testing of a specific application system.
- **Test phase** – The phase of the systems development life cycle in which testing will occur.

Not all test factors will be applicable to all software systems. The development team will need to select and rank the test factors for the specific software system being developed. Once selected and ranked, the strategy for testing will be partially defined.

Common Test Factors

When stated in a positive manner, the test risks become the factors that need to be considered in the development of the test strategy. The following list briefly describes the test factors to consider when formulating a test strategy.[ii]

Correctness. Assurance that the data entered, processed, and output by the application system is accurate and complete. Accuracy and completeness are achieved through controls over transactions and data elements, which should commence when a transaction is originated and conclude when the transaction data has been used for its intended purpose.

File integrity. Assurance that the data entered into the application system will be returned unaltered. The file integrity procedures ensure that the right file is used and that the data on the file and the sequence in which the data is stored and retrieved is correct.

Authorization. Assurance that data is processed in accordance with the intents of management. In an application system, there is both general and specific authorization for the processing of transactions. General authorization governs the authority to conduct different types of business, while specific authorization provides the authority to perform a specific act.

Audit trail. The capability to substantiate the processing that has occurred. The processing of data can be supported through the retention of sufficient evidential matter to substantiate the accuracy, completeness, timeliness, and authorization of data. The process of saving the supporting evidential matter is frequently called an audit trail.

Continuity of processing. The ability to sustain processing in the event problems occur. Continuity of processing ensures that the necessary procedures and backup information are available to recover operations should integrity be lost due to problems. Continuity of processing includes the timeliness of recovery operations and the ability to maintain processing periods when the computer is inoperable.

Service levels. Assurance that the desired results will be available within a time frame acceptable to the user. To achieve the desired service level, it is necessary to match user requirements with available resources. Resources include input/output capabilities, communication facilities, processing, and systems software capabilities.

Access control. Assurance that the application system resources will be protected against accidental and intentional modification, destruction, misuse, and disclosure. The security procedure is the totality of the steps taken to ensure the integrity of application data and programs from unintentional and unauthorized acts.

Compliance. Assurance that the system is designed in accordance with organizational strategy, policies, procedures, and standards. These requirements need to be identified, implemented, and maintained in conjunction with other application requirements.

Reliability. Assurance that the application will perform its intended function with the required precision over an extended period of time. The correctness of processing deals with the ability of the system to process valid transactions correctly, while reliability relates to the system's being able to perform correctly over an extended period of time when placed into production.

Ease of use. The extent of effort required to learn, operate, prepare input for, and interpret output from the system. This test factor deals with the usability of the system to the people interfacing with the application system.

Maintainability. The effort required to locate and fix an error in an operational system. Error is used in the broad context to mean both a defect in the system and a misinterpretation of user requirements.

Portability. The effort required to transfer a program from one hardware configuration and/or software system environment to another. The effort includes data conversion, program changes, operating system, and documentation changes.

Coupling. The effort required to interconnect components within an application system and with all other application systems in their processing environment.

Performance. The amount of computing resources and code required by a system to perform its stated functions. Performance includes both the manual and automated segments involved in fulfilling system functions.

Ease of operation. The amount of effort required to integrate the system into the operating environment and then to operate the application system. The procedures can be both manual and automated.

Developing a Risk-Based Test Strategy

The test strategy is illustrated in Figure 15.1 - Test Strategy Matrix. This is a generic strategy that would be customized to a specific software system. The applicable test factors would be listed and ranked, and the phases of development would be listed as the phases in which testing must occur. The risks are then shown for each phase and for each factor.[iii]

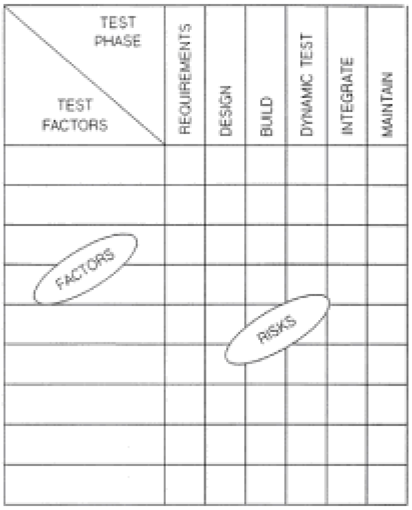


Figure 1 Test Strategy Matrix

Four steps should be followed to develop a customized test strategy. The completed customized strategy will be the test factor/test phase matrix as illustrated in Figure 15.1.

1. **Select and rank test factors.** The customers/key users of the system, in conjunction with the test team, should select and rank the test factors. In most instances, only three to seven factors will be needed. Statistically, if the key factors are selected and ranked, the other factors will normally be addressed in a manner consistent with supporting the key factors. These should be listed in the matrix in sequence from the most significant test factor to the least significant. Rank your factors in sequence from the most to least significant. Specific test risks can be substituted for factors, or you can expand the factors to describe risks in more detail.

2. **Identify the system development phases.** The project development team should identify the phases of their development process. This is normally obtained from the system development methodology. These phases should be recorded in the test phase component of the matrix.

3. **Identify the business risks associated with the system under development.** The developers, key users, customers, and test personnel should brainstorm the risks associated with the software system. Most organizations have a brainstorming technique, and it is appropriate for individuals to use the technique in which they have had training and prior use. Using this technique, the risks should be identified and agreed upon by the group. The risks should then be ranked into high, medium, and low. This is a relational severity indicator, meaning that one-third of all risks should be indicated as high, one-third as medium, and one-third as low.

4. **Place risks in the matrix.** The risk team should determine the test phase in which the risk needs to be addressed by the test team, and the test factor to which the risk is associated. Take the example of a payroll system: If there was a concern about compliance to federal and state payroll laws, the risk would be the penalties associated with noncompliance. Assuming compliance was picked as one of the significant test factors, the risk would be most prevalent during the requirements phase. Thus, in the matrix at the intersection between the compliance test factor and the requirements phase, the risk of "penalties associated with noncompliance to federal and state payroll laws" should be inserted. Note that this may be done by a reference number, cross-referencing the risk. The risk would then have associated with it an H, M, or L, for high, medium, or low risk.

Quantifying and Prioritizing Risk

In this section, a formal approach for establishing risk and prioritizing the items on the test inventory using both quantitative and qualitative analysis to establish a numeric value for risk, based on a number of specific criteria. In the early planning phases of a test effort, this analysis is used to focus test resources on the most important parts of the project and to size the test effort.[iv]

Define risk ranking attributes

In order to focus on the risks that are most important, a system for ranking test factors should be defined.

The following attributes are most frequently used in ranking test factors. Different systems and industries may have others that are important to them. Additional attributes for ranking may be added as needed.

- Requirement
- Severity
- Probability
- Cost
- Visibility
- Tolerability
- Human factors

Define ranking criteria and index values

The next step is to determine the appropriate ranking index for a given ranking criteria.

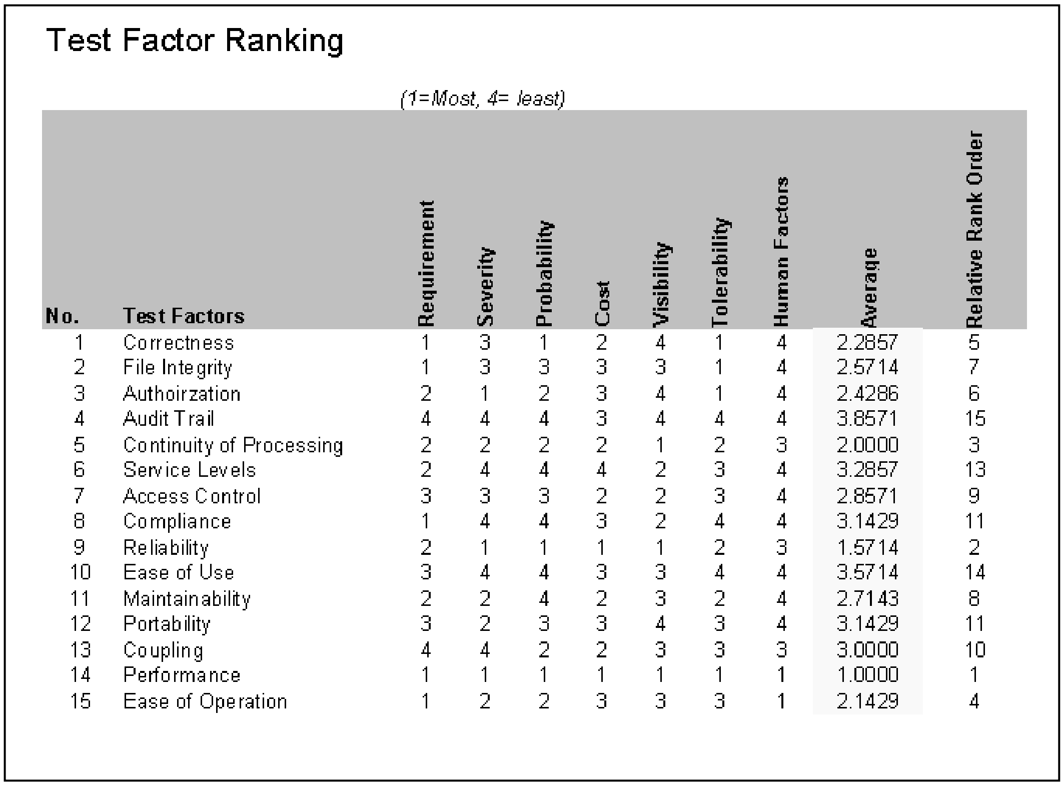
Ranking Attribute	**Ranking Criteria**	**Risk Index**
Required	Is this a contractual requirement such as service level agreement?	Must have = 1

	Mitigating factors: Is this an act of God? If the design requirements have been prioritized, then the priority can be transferred directly to rank. Has some level of validation and verification been promised?	Nice to have = 4
Severity	Is this life-threatening? System-critical? Mitigating factors: Can this be prevented?	Most serious = 1 Least serious = 4
Probability	How likely is it that a failure will occur? Mitigating factors: Has it been previously tested or integrated? Do we know anything about it?	Most probable = 1 Least probable = 4
Cost	How much would it cost the company if this broke? Mitigating factors: Is there a workaround? Can we give them a fix via the Internet?	Most expensive = 1 Least expensive = 4
Visibility	If it failed, how many people would see it? Mitigating factors: Volume and time of day intermittent or constant?	Most visible = 1 Least visible = 4
Tolerability	How tolerant will the user be of this failure? For example, a glitch shows you overdrew your checking account; you didn't really, it just looks that way. How tolerant would the user be of that? Mitigating factors: Will the user community forgive the mistake?	Most intolerable = 1 Least intolerable = 4
Human Factors	Can/will humans succeed using this interface?	Fail = 1

	Mitigating factors: Can they get instructions or help from Customer Service?	Succeed = 4

The only caution with risk ranking is if you consistently use too few criteria. If you use only two criteria for all ranking, for example, then the risk analysis may not be granular enough to provide an accurate representation of the risk. For example, if a project uses only two criteria to rank the testable items: probability and impact. These two criteria will not account for human factors, or cost, or intersystem dependencies. It may turn out that these factors are the driving qualities of the feature set. So, the risk analysis will not provide an accurate ranking.

Calculate weighted rank index.



Test Factor Ranking

(1=Most, 4= least)

No.	Test Factors	Requirement	Severity	Probability	Cost	Visibility	Tolerability	Human Factors	Average	Relative Rank Order
1	Correctness	1	3	1	2	4	1	4	2.2857	5
2	File Integrity	1	3	3	3	3	1	4	2.5714	7
3	Authoirzation	2	1	2	3	4	1	4	2.4286	6
4	Audit Trail	4	4	4	3	4	4	4	3.8571	15
5	Continuity of Processing	2	2	2	2	1	2	3	2.0000	3
6	Service Levels	2	4	4	4	2	3	4	3.2857	13
7	Access Control	3	3	3	2	2	3	4	2.8571	9
8	Compliance	1	4	4	3	2	4	4	3.1429	11
9	Reliability	2	1	1	1	1	2	3	1.5714	2
10	Ease of Use	3	4	4	3	3	4	4	3.5714	14
11	Maintainability	2	2	4	2	3	2	4	2.7143	8
12	Portability	3	2	3	3	4	3	4	3.1429	11
13	Coupling	4	4	2	2	3	3	3	3.0000	10
14	Performance	1	1	1	1	1	1	1	1.0000	1
15	Ease of Operation	1	2	2	3	3	3	1	2.1429	4

Figure 2 Test Factor Ranking.

The last step is to compile the ranking results for each test factor by applying the ranking criteria to determine the average ranking for each factor. Summing the rank index values for each factor and dividing by the number of attributes that have a value will yield the average ranking. Those factors with the **lowest average ranking index** should be sorted to the top of the matrix as these are the factors that should be considered the most important. See the example in Figure.2.

Conclusion

Risk Based Testing Approach is an highly effective method for optimizing test case execution in software test life cycle (STLC). If used meticulously, RBT can reduce the testing effort close to 40% maintain adequate test coverage with cost alleviation. The number of bugs that are found during production can be drastically reduced by implementing RBT approach. As the priority of test case execution is based on the risk proneness of the test cases, the chances of missing the high risk test cases is minimized in test case execution and hence the bugs that can yield by not executing these high risk test cases is also reduced. It is recommended to automate the entire process of Risk based Testing (design the algorithm of identification of the risk test cases, ranking them and executing them.) for faster and effective results of this approach.

References

1.Perry, William E., Effective Methods for Software Testing, Chapter 2, *Test Strategy*, *Test Factors,* John

Wiley & Sons, New York, 2000
2. Hutchenson, Marnie, Software Testing Fundamentals: Methods and Metrics, Chapter 9 – Risk Analysis, John Wiley and Sons, 2003