



Risk Based Testing: Technique for Risk-based Test Case Generation and Prioritization

Pooja Sharma

Dept. of CSE

CBS Group of Institutions

Fatehpuri, Jhajjar, Haryana, India

Abstract: Nowadays, there are more projects, more competitive pressures and greater failure risk which needs to be managed with fewer resources with tighter timelines. Instead of doing more with less and risking late projects, increased costs or low quality, we need to find ways to achieve better with less. The focus of testing has to be placed on aspects of software that matter most with an aim of reducing the risk of failure as well as ensuring the quality and stability of the business applications. This can be achieved by applying the principle of Risk Based Prioritization of tests, known as Risk-based testing (RBT). The aim of Risk Based testing approach is to ensure that appropriate testing activities are identified and prioritized based on risk. The primary role of risk-based testing is to optimize available resources and time without affecting the quality of the product. In this light, this paper presents the progress different risk-based testing metrics to measure and control test cases and test activities progress, efforts and costs. In risk-based testing, the probability of a fault and the damage that this fault can cause when leading to a failure is considered for test case prioritization. Existing approaches for risk based testing provide guidelines for deriving test cases. However, those guidelines lack the level of detail and precision needed for automation. In this contribution, we introduce the risk-based testing technique RiteDAP, which automatically generates system test cases from activity diagrams and prioritizes those test cases based on risk.

Keywords: Model-based Testing, Risk-based Testing, Test Case Generation

I. Introduction

The main objective of software testing is to detect defects in a code or in a module or in a program, in other words Process of giving assurance to the client that, the software under test is performing as intended (meeting the requirements). In general, tests are performed in order to show a lack of quality as discovered by defects, testing always involves comparing between the product and the requirements. Testing is of different types and risk-based testing (RBT) is considered as one testing which is used to optimize available resources and time without affecting the quality of your product. In RBT each test is intended to probe a specific risk that was previously identified through risk analysis. A simple example is that in Database applications, there is a risk of injection attacks, where an attacker fools the server into displaying results of arbitrary SQL queries. A risk-based test might actually try to carry out an injection attack, or at least provide evidence that such an attack is possible [1], Risk-based testing helps address the rise in business and technological complexity and the growing size of applications by prioritizing test cases based on the defined criticality of a function, encouraging impact assessment of an application functionality failure and increasing testing effectiveness.

Any uncertainty or possibility of loss may result in non conformance of any of these key factors, leading to overtime / over-budget or poor quality project. Software risks, which impact above 3 key factors, can be broadly categorized as [2]:

A. Requirement Risks:

Requirements is unclear or poorly defined requirements, missing requirement analysis, requirements that are not in-line with customers' needs, inconsistent ambiguous requirements, inadequate requirements, invalid or impossible requirements, unable to measurable the requirements in terms of specific values etc.

B. Technical Risks:

Continuous changing requirements, Complexity of architecture or Product is complex to implement, technology change, configuration change, inadequate technical support/ knowledge or No advanced technology available, lack of domain knowledge etc.

C. Schedule Risks:

Cost overruns, Wrong budget estimation unrealistic time line, Failure to address priority conflicts, No communication in team, inadequate skilled resources, improper resource planning etc.

Exhaustive testing is infeasible except for trivial cases. Thus, a subset of all possible test cases is typically determined based on one or more coverage criteria. Examples for such criteria are statement coverage, branch coverage (e.g., see [3]) or transition coverage (e.g., see [4]). Although applying those criteria leads to a tractable subset of test cases, in practice, limited testing budgets can prevent some of those test cases from being executed. With risk-based testing, testers can face the challenge of reducing the chances for the occurrence of faults that lead to high damage. When determining the priority of test cases – and thus the order in which to execute test cases – risk-based testing considers both the damage that would be caused by faults as well as the probability that those faults are contained in the test object. In general, the goal of a risk-based testing strategy is “to find the most important defects as early as possible against the lowest cost” [5]. Thus, even when testing budgets run out, risk-based testing will have helped testers to spend these budgets in an efficient way. Existing approaches for risk-based testing suggest strategies for prioritizing test cases, which either provide only rough guidelines for actually deriving test cases, or assume that test cases already exist (regression testing). In contrast to that, our RiteDAP technique, which is presented in this contribution, allows for the automatic derivation of system test cases from activity diagrams as well as their prioritization based on risk.

II. Related Work

Over the last decade, risk-based testing has received considerable attention in both academics and industry. Several researchers have proposed their approaches for the implementation of risk-based testing in the software projects. Bach [6] proposes different heuristics to assess risks and suggests taking the identified risks into account during the testing process. However, no indication is given on how to actually derive test cases. Van der Aalst [5] resp. Amland [7] propose calculating a risk score resp. risk exposure for each module based on the chance of failure and damage. Based on the result, tests are derived and executed. Yet, how to derive test cases is not covered by the approaches. Pinkster et al. [8] associate risks with a priority regarding testing. Then, test cases are derived and executed for modules which achieve the highest priority. However, no detailed technique for how to derive test cases according to prioritization of risks is presented. Chen & Probert [9] and Srikanth et al. [10, 11] suggest risk-based regression test case prioritization approaches. Furthermore, Elbaum et al. [12] present a prioritization technique for regression testing that has similarities to risk-based testing, because fault severity is considered. Obviously, the initial derivation of test cases is not covered by these regression testing approaches.

III. Risk Based Software Testing

Risk Based Testing is a method for prioritizing the tests based on the risk of their failure because tester cannot test everything within available resource, usually it starts early in the project cycle. Idea is to organize testing efforts in such an approach that it reduces the level of product risk at the time of delivery.

In other words in this approach, testing is prioritized in terms of the probability that some feature of the program will fail and the estimated cost of failure. The greater the probability of an expensive failure, the more important it is to test that feature as early as possible and as carefully as possible. [13] The objective of Risk-based testing are: most feasible coverage and effective usage of limited resources. Thus, in RBT, we organize the testing processes in order to maximize business value and resources. RBT perform the right level and type of coverage on the right parts and at the right time. The main reason for adopting RBT can be the aggregated impact of limited resources – time, budget and human. Since testing generally comes in the last stage of the SDLC, the project's calendar doesn't allow sufficient time for a thorough testing of all functions. Furthermore, the project's budget limits the number of skilled human / software resources. In such a situation, test coverage of all the minute detail of the application would not be possible. In order to balance such situations, more focus should be given on testing those areas that represent the largest risk, if a fault occurred.

A) Risk Based Testing General Process

The RBT process can be carried out through following important steps [2]:

Step1 - Describe all requirements in terms of risk associated to them also, looks at ways of establishing what the risks are and where they are.

Step2 - Prioritize the requirements, based on risk assessment, looks into the critical, complex and potential error prone areas.

Step3 - According to requirement prioritization define and plan tests also, look for risk Mitigation where tests are built to mitigate the risk.

Step4 - Execute test according to prioritization and acceptance criteria or Monitor / Report regarding the risks.

When deciding on what parts of testing to outsource, you will have to look at it from different angles, mainly at the dimensions of test levels, test types and test activities:

Test Levels: Low-level testing (Unit & Integration test) is generally carried out by the developers themselves. If development is outsourced, these activities are also outsourced with it. System testing on the other hand should

be performed by an independent test team and is therefore an excellent candidate for test outsourcing. Finally, Acceptance testing requires business know how (for user acceptance) and a production-like test environment. It is therefore difficult to outsource.

Test Types: Generally speaking, all types of testing, both functional and non-functional, can be outsourced. By its nature, regression test is a good candidate for cost saving, because it involves regular repetition and test automation. Know-how intense test types like load & performance, usability and security are best outsourced to a specialist organization, on a case-by-case basis.

Test Activities: defining what activities within the test process should be outsourced (e.g., test planning, specification, execution and reporting) requires a strategic decision on how much control and knowledge is given away. It can range from test execution only, to performance of the whole process.

IV. Approach

We propose RiteDAP (Risk-based test case Derivation and Prioritization) as a model-based approach to risk-based system testing. RiteDAP uses test models, which are augmented with risk information, for test case generation and prioritization.

A) Test Model Used For RiteDAP

RiteDAP uses activity diagrams (ADs) as test models. Such Ads already exist as a result from requirements engineering, or they can be specifically created for testing purposes. In this respect we follow other authors who have successfully employed ADs for model based system testing (see [14,15]) and build on our own positive experience in applying ADs as test models (see [16,17]). In RiteDAP risks are determined by the function $R(P, D) = P \cdot D$, where P is the probability that an entity contains a fault and D is the total damage caused by this fault. Many researchers have addressed the problem of risk assessment using guidelines, checklists, heuristics and risk criteria (e.g., see [5, 6, and 8]). All these risk assessment approaches rely on experts that perform the assessment. Other approaches enable automation in risk assessment by employing metrics based on code artifacts (e.g., see [18, 19]). In previous work, we have shown how to automatically determine the probability of risks in the early stages of software development by employing requirements metrics [20]. To allow for risk-based prioritization, the test model has to be augmented with risk information resulting from risk assessment.

B) Activities of RiteDAP

RiteDAP consists of two main activities. First, a set of unordered test case scenarios is derived from the test model. After that, the test case scenarios are ordered based on the risk information in the test model. This separation of concerns between the two activities enables us to use existing non-risk-based techniques for generating a potential set of test cases and then choosing different risk-based prioritization strategies for ordering the test cases.

Deriving potential test case scenarios: RiteDAP does not directly generate test cases but generates test case scenarios (TCSs), which abstract from concrete test data and represent a path through the test model. TCSs provide the starting point for defining concrete test cases by augmenting the TCSs with concrete test data. Test data can be either determined manually by the testers or automatically, if the test model is detailed enough (e.g., see [21, 22]).

Prioritizing and ordering test case scenarios: To derive the execution order for the TCSs, their priority is determined. For each TCS the sum of the risks of all actions that are covered by the TCS is calculated. For this step of the approach we follow the solution that has been chosen in [9, 10, and 11]. Table 1 shows the derived TCSs together with the sum of the risks.

TCS	Path	$\sum$ Risk
S1	abghklmnopef	17
S2	abghklmmtlmnopqrsf	41
S3	abghijpef	27
S4	abghijpqrsf	45
S5	abcdef	19
S6	abcdqrsf	38
S7	abghklmmtlmmtlmnopef	29

Table 1. Possible test case scenarios and associated risks

To derive an ordering of TCSs, two different risk-based prioritization strategies have been implemented in RiteDAP:

- Total Risk Score Prioritization (TRSP)
- Additional Risk Score Prioritization (ARSP)

V. Validation

The basis for the validation of RiteDAP is the hypothesis that prioritizing test case scenarios according to a risk-based strategy can uncover critical faults earlier than existing prioritization strategies that do not explicitly

consider risk; especially, when the testing resources are limited. To support this hypothesis, we have carried out a case study, based on a practical example.

A) Validation Approach

To evaluate the effectiveness of RiteDAP, we use the following non-risk-based prioritization strategies as a baseline:

Random Prioritization (RP): RP is achieved when TCSs are chosen randomly from the set of generated TCSs.

Optimal Prioritization (OP): OP can only be determined in retrospective when all faults that are uncovered using the initial TCS set and possibly additional (manual) inspections of the test object have been performed. OP can be seen as an upper bound for prioritization strategies.

Total Action Coverage Prioritization (TACP): We use total coverage prioritization based on the achieved coverage of actions in the activity diagram. It can be compared to the functional coverage described in [23]. TCSs are ordered with respect to the number of actions they cover. A possible TCS order according to TACP is S7, S2, S4, S1, S6, S3, S5.

Additional Action Coverage Prioritization (AACP): This strategy prioritizes TCSs with respect to the number of previously uncovered actions.

Thus, S7 or S2 is the first TCS to be used. After selection of e.g. S7, each action covered is marked and not further taken into account. Therefore, S4 or S6 is chosen next (they both cover 3 new actions). A possible TCS order according to AACP is S7, S6, S4, S5, S3, S2, S1. To measure the effectiveness of the different prioritization strategies we introduce the metric Average Percentage of Damage Prevented (APDP) which corresponds to the APFDC metric presented in [13].

In contrast to APFDC the APDP is adapted for a risk-based approach and we abstract from varying test costs. Damage takes the role that fault severities have in APFDC. To evaluate prioritization strategies, the damage associated with a detected fault must be estimated. We do this based on the damage estimation assigned to an action during risk assessment (i.e., we use the value D which is a parameter of the risk function). Whenever a fault in an action (more precisely, in the implementation of that action) is discovered, we assume a prevented damage that is the same as the damage that has been assigned to the action during risk assessment.

B) Case Study

Augmenting the activities of the test model with risk information is a first step in achieving our validation example. Since in RiteDAP risk is quantified by the function $R(P, D) = P \cdot D$, values for P and D have to be determined for each activity. We draw on income tax statistics to collect these values. The probability P that an action will lead to a failure is, among others, determined by the usage frequency of that action (cf. e.g. [5, 6, 7, and 24]). The usage frequency of an activity can be calculated with respect to the number of taxpayers that are affected by that action. As an example, two million of the overall 25.7 million taxpayers in Germany have profited by a specific tax exemption in 2001. The total damage D caused by a fault in such an activity thus strongly depends on the resulting financial losses for the tax payers. The average financial loss for a tax payer can be estimated by relating each activity to the number of tax payers affected by that activity and the amount of taxes calculated by that activity. The resulting activity diagram for the fiscal year 2002 contains 17 activities and 5 decisions. The final step to achieve a complete validation example is to identify actual faults and to determine which activities of the test model are affected by those faults. Software products that implemented the program flow chart for the income tax calculation provide a crucial source of realistic faults. Based on the fault data of those software products, four of the reported faults could be related to specific activities in the test model.

C) Results

The ADAP values of the non-risk-based strategies are listed in Table 2. For random prioritization and in cases where a choice between several test case scenarios was necessary, the results of the best and the worst choice are shown.

Non-risk-based prioritization strategy	ADAP value		
	worst choice	best choice	random choice
Random (RP)	0.670	0.994	0.954
Optimal (OP)	-	-	0.994
Total Action Coverage (TACP)	0.957	0.986	0.979
Additional Action Coverage (AACP)	0.986	0.992	0.988

Table 2. ADAP values with non-risk-based prioritization

For the validation example, 80 TCSs have been generated. With total risk score prioritization (TRSP) all faults have been identified after the first 8 TCSs. The corresponding APDP value is 0.981. Additional risk score prioritization (ARSP) achieves a better ADAP value of 0.988, because all faults are already detected after 4 TCSs. The ADAP values of the prioritization strategies investigated in this case study indicate that risk-based approaches provide early fault detection and thus effective damage prevention. Whereas optimal prioritization

(which is not applicable in practice) has reached a higher ADAP value, the ADAP value of random prioritization was considerably lower (in the worst case, the ADAP value for RP could be 0.67). The ADAP value of total action coverage prioritization (TACP) is lower than the risk-based alternative of total risk score prioritization (TRSP). For the random choice, TACP and TRSP are exceeded by additional action coverage prioritization (AACP) and additional risk score prioritization (ARSP), which both have an ADAP value of 0.988. However, where, in our validation example, AACP relies on choices to be taken, the ARSP was independent of such a choice. This means that when the wrong choices are taken in AACP, ARSP (the risk-based strategy) will outperform AACP (the non-risk-based strategy), because ARSP always has an ADAP value of 0.988.

VI. Conclusion

Risk based testing is a powerful testing technique that helps the testing teams to streamline their testing efforts, which in turn helps in mitigating the risk and minimizing the testing efforts, thus, bringing an objectivity to test designing and test management activities. The goal of risk-based testing cannot practically guarantee a risk-free project. What we can expect from risk-based testing is to carry out the testing with best practices in risk management to accomplish a project outcome that balances risks with quality, features, budget and schedule. Based on our analysis of different approaches of RBT, we will propose a novel risk-based testing model in future. Furthermore, we will apply that model to do some case studies in order to get empirical results for our methodology. we have presented RiteDAP, a model-based technique for risk-based system testing. RiteDAP automatically generates and prioritizes system test cases by employing test models that have been augmented with information about risks. RiteDAP has been implemented in a prototype tool, which has been applied to a practical example. The results of the validation of RiteDAP have shown that generating and prioritizing test case scenarios based on augmented test models enable the early detection of critical faults during the development process. We are aware that the presented results are only a first evidence of the efficiency of our approach. Therefore, based on the presented validation approach, we plan to perform additional case studies and experiments within the German research project ran TEST, which involves industrial partners from rail automation and financial information and portfolio management. Our future work also includes investigating additional methods of calculating risk values for the test case scenarios that go beyond summing the risks of all actions that are covered by a scenario.

VII. REFERENCES

- [1] C. C. Michael, Will Radosevich, Cigital, 2007, [Online], available, <https://buildsecurityin.us/cert.gov/bsi/articles/best-practices/testing/255-BSI.html>
- [2] Unnati Bajpai, 2012 Risk Based Testing 102, [Online], available, <http://help.utest.com/testers/crash-courses/general/risk-based-testing-102>
- [3] Beizer, B. 1990. Software Testing Techniques. Van Nostrand Reinhold, New York, NY, 1990.
- [4] Offutt, A.J.; Xiong, Y.; Liu, S. 1999. Criteria for generating specification-based tests. 5th ICECCS, Las Vegas, NV, USA, Oct 1999, pp. 119-129.
- [5] van der Aalst, L. 2006. Risk Based Test Strategy Judged. 7th ICSTEST, Düsseldorf, Germany, May 2006.
- [6] Bach, J. 1999. Heuristic Risk-Based Testing. STQE Magazine 1(6), Nov 1999.
- [7] Amland, S. 2000. Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. JSS 53(3), Sept 2000.
- [8] Pinkster, I.; van de Burgt, B.; Janssen, D.; van Veenendaal, E. 2004. Successful Test Management – An Integral Approach. Springer, Berlin, 2004.
- [9] Chen, Y.; Probert, R. L. 2003. A Risk-based Regression Test Selection Strategy. 14th ISSRE, Denver, USA, Nov 2003.
- [10] Srikanth, H.; Williams, L. 2005. On the economics of requirements-based test case prioritization. 7th EDSER, Shanghai, China, May 2005.
- [11] Srikanth, H. 2005. Value-Driven System Level Test Case Prioritization. North Carolina State University, 2005.
- [12] Elbaum, S.; Malishevsky, A.; Rothermel, G. 2001. Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization. 23rd ICSE, Toronto, Canada, May 2001.
- [13] Pramod Lumb, 2012 Risk Based Testing 101, [online], <http://help.utest.com/testers/crash-courses/general/risk-based-testing-101>
- [14] Briand, L.C. Labiche, Y 2001. A UML-Based Approach to System Testing. 4th UML, Toronto, Canada, Oct 2001.
- [15] Chen, M.; Qiu, X.; Li, X. 2007. Automatic Test Case Generation for UML Activity Diagrams. 1st AST, Shanghai, China, May 2006.
- [16] Reis, S.; Metzger, A.; Pohl, K. 2007. Integration Testing in Software Product Line Engineering: A Model-Based Technique. 10th FASE, Braga, Portugal, Mar/Apr 2007.
- [17] Reuys, A.; Kamsties, E.; Pohl, K.; Reis, S. 2005. Model-Based System Testing of Software Product Families. 17th CAiSE, Porto, Portugal, June 2005.
- [18] Benlarbi, S.; El Emam, K.; Goel, N. 1999. Issues in Validating Object-Oriented Metrics for Early Risk Prediction. 10th ISSRE, Boca Raton, FL, USA, Nov 1999.
- [19] Rosenberg, L.H.; Stapko, R.; Gallo, A. 1999. Risk-based Object Oriented Testing. 24th SWE, NASA, Greenbelt, MD, USA, Dec 1999.
- [20] Stallbaum, H.; Metzger, A. 2007. Employing Requirements Metrics for Automating Early Risk Assessment. 1st MeReP, Palma, Spain, Nov 2007.
- [21] Edvardsson, J. 1999. A survey on automatic test data generation. 2nd ECSEL, Linköping, Sweden, Oct 1999.
- [22] Ferguson, R.; Korel, B. 1996. The chaining approach for software test data generation. ACM TOSEM 5(1), Jan 1996.
- [23] Elbaum, S.; Malishevsky, A.; Rothermel, G. 2000. Prioritizing Test Cases for Regression Testing. Technical Report 00-60-03, Oregon State University, Feb 2000.
- [24] Ottevanger, I. 1999. A Risk-Based Test Strategy. STARWest, San Jose, CA, USA, Oct 1999.