An Automated Technique for Risk-based Test Case Generation and Prioritization

Heiko Stallbaum
University of Duisburg-Essen
Software Systems Engineering
Schützenbahn 70
45117 Essen, Germany
heiko.stallbaum@sse.uni-due.de

Andreas Metzger
University of Duisburg-Essen
Software Systems Engineering
Schützenbahn 70
45117 Essen, Germany
andreas.metzger@sse.uni-due.de

Klaus Pohl
University of Duisburg-Essen
Software Systems Engineering
Schützenbahn 70
45117 Essen, Germany
klaus.pohl@sse.uni-due.de

ABSTRACT
In practice, available testing budgets limit the number of test cases that can be executed. Thus, a representative subset of all possible test cases must be chosen to guarantee adequate coverage of a test object. 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. The results of applying the technique to a practical example are presented and the ability of different prioritization strategies to uncover faults is evaluated.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms
Algorithms, Management, Measurement, Reliability, Verification

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

1. INTRODUCTION
Exhaustive testing is infeasible except for trivial cases. Thus, a subset of all possible test cases is typically determined based one or more coverage criteria. Examples for such criteria are statement coverage, branch coverage (e.g., see [6]) or transition coverage (e.g., see [15]). 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" [1]. 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.

2. RELATED WORK
Bach [3] 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 [1] resp. Amland [2] 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. [17] 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 [10] and Srikanth et al. [21, 22] suggest risk-based regression test case prioritization approaches. Furthermore, Elbaum et al. [13] 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.

In addition, the following related non-risk-based approaches exist: The CoWTeSt-approach [4] presented by Basanieri et al. assigns a relative weight to different system functionalities. Although the criteria, which are proposed for determining the weight, can have an influence on risk, risk is not explicitly addressed. The Cow_Suite approach [5] from the same authors combines CoWTeSt with Usage Interaction Testing (UIT). The test model used in Cow_Suite does not express risks which results in insufficient consideration of risks. Statistical Usage Testing [24] allows deriving test cases according to a usage profile. Although this approach employs usage probabilities, which may influence risk, it does not comprehensively address risks.

3. 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.

3.1 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 [8, 9]) and build on our own positive experience in applying ADs as test models (see [18, 19]).

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 (cf. [1, 2, 10, 20]). Many researchers have addressed the problem of risk assessment using guidelines, checklists, heuristics and risk criteria (e.g., see [1, 3, 17]). 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 [7, 20]. 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 [23]. To allow for risk-based prioritization, the test model has to be augmented with risk information resulting from risk assessment. Figure 1 shows an AD where the stereotype <<raAction>> is applied to actions that have been augmented with risks.

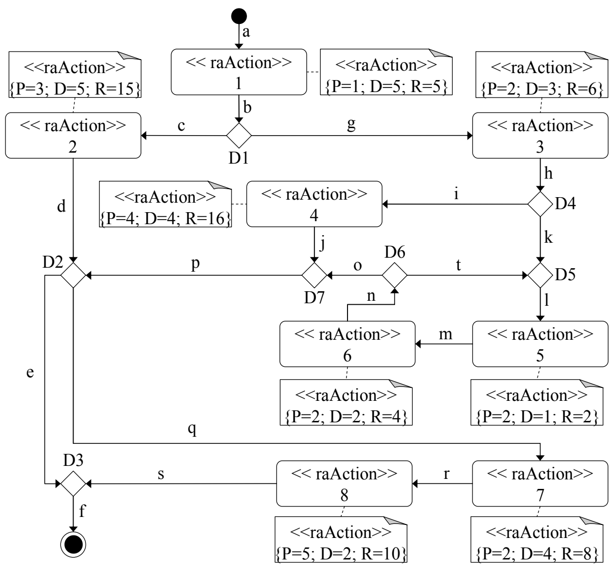

Figure 1. Sample test model augmented with risk information

3.2 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 [11, 14]). In RiteDAP possible TCSs are derived with the boundary interior criterion. The boundary interior criterion subsumes transition coverage which is a typical criterion for non-risk-based approaches [15]. The criterion produces a sufficient and manageable number of TCSs that can be used as input for the prioritization. It requires that across the set of TCSs the following three requirements are met: 1) the body of each loop is not executed, 2) the body of each loop is executed once, and 3) the body of each loop is executed more than once. The TCSs generated by RiteDAP will execute a loop twice to satisfy the third requirement. In order to derive possible TCSs, we use the node reduction algorithm [6] as a starting point. Applying this algorithm together with the boundary interior criterion to the sample test model in Figure 1 leads to the TCSs listed in Table 1.

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 [10, 21, 22]. Table 1 shows the derived TCSs together with the sum of the risks.

Table 1. Possible test case scenarios and associated risks

TCS	Path	∑ Risk
S1	abghklmnopef	17
S2	abghklmntlmnopqrsf	41
S3	abghijpef	27
S4	abghijpqrsf	45
S5	abcdef	19
S6	abcdqrsf	38
S7	abghklmntlmntlmnopef	29

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

– *Total Risk Score Prioritization (TRSP):* This prioritization has been presented in [21, 22]. TCSs are scheduled in the descending order of their associated risks. This results in a TCS order S4, S2, S6, S7, S3, S5, S1 for the example.

– *Additional Risk Score Prioritization (ARSP):* In this prioritization (see [10]), the first TCS is chosen according to the total risk score. Then, only risks not already covered by a selected TCS are taken into account for further prioritization. In the example, S4 is chosen first. After that, S7 with an additional risk score of 18 has the highest score and thus S7 is chosen next. Now, S5 and S6 both have an additional risk score of 15, whereas all others are tied with a score of 0. Whenever two or more TCSs have the same score, one of those TCSs can be selected by another strategy, e.g. randomly. After selecting e.g. S5, no TCS provides additional risk coverage.

4. 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.

4.1 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 (cf. [12]).

– *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 [12]. 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.

In the style of [13], the APDP metric can quantitatively be described as follows. Let T be a test suite containing n test case scenarios. Let F be a set of m faults revealed by T, and let d_1, d_2, …, d_m be the damages caused by those faults. Let TF_i be the number of the first test case scenario in an ordering T' of T that reveals fault i. The weighted average percentage of damage prevented during the execution of T' is defined as follows:

$$APDP = \sum_{i=i}^{m} d_i \cdot \left(n - TF_i + \frac{1}{2} \right) \Big/ n \cdot \sum_{i=1}^{m} d_i$$

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, cf. Section 3.1). 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. If there is a more complex relationship between faults and their severity and this relationship is essential to be considered during testing, the actions in the activity diagram should be refined and annotated with more precise risk (including damage) information.

4.2 Case Study

To validate our approach, we have implemented the RiteDAP technique in a prototype tool and applied it to a practical example. The German Federal Ministry of Finance annually publishes a program flow chart defining how to calculate the income tax for the upcoming fiscal year. Software companies use this flow chart to implement income tax calculation software. Thus, the program flow chart of the income tax calculation is widely used and provides a realistic basis for our validation.

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. [1, 2, 3, 16]). The usage frequency of an activity can be calculated with respect to the number of tax-payers 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 (cf. e.g. [1, 16]). 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.

4.3 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.

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

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

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.

5. CONCLUSION AND PERSPECTIVES

In this paper, 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 ranTEST, 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.

6. ACKNOWLEDGMENTS

The authors thank Thomas Rinke and Dennis Chotsko for their valuable contribution to the RiteDAP technique. We thank the anonymous reviewers for their helpful comments. The research presented was partially funded by the German Federal Ministry of Education and Research under grant 01 IS E09 B (ranTEST).

7. REFERENCES

[1] van der Aalst, L. 2006. Risk Based Test Strategy Judged. 7th ICSTEST, Düsseldorf, Germany, May 2006.

[2] 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.

[3] Bach, J. 1999. Heuristic Risk-Based Testing. STQE Magazine 1(6), Nov 1999.

[4] Basanieri, F.; Bertolino, A.; Machetti, E. 2001. CoWTeSt: A Cost Weighted Test Strategy. Escom-Scope 2001, London, England, Apr 2001, pp. 387-396.

[5] Basanieri, F.; Bertolino, A.; Machetti, E. 2002. The Cow_ Suite Approach to Planning and Deriving Test Suites in UML Projects. 5th UML, Dresden, Germany, Sept/Oct 2002.

[6] Beizer, B. 1990. Software Testing Techniques. Van Nostrand Reinhold, New York, NY, 1990.

[7] 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.

[8] Briand, L.C. Labiche, Y 2001. A UML-Based Approach to System Testing. 4th UML, Toronto, Canada, Oct 2001.

[9] Chen, M.; Qiu, X.; Li, X. 2007. Automatic Test Case Generation for UML Activity Diagrams. 1st AST, Shanghai, China, May 2006.

[10] Chen, Y.; Probert, R. L. 2003. A Risk-based Regression Test Selection Strategy. 14th ISSRE, Denver, USA, Nov 2003.

[11] Edvardsson, J. 1999. A survey on automatic test data generation. 2nd ECSEL, Linköping, Sweden, Oct 1999.

[12] Elbaum, S.; Malishevsky, A.; Rothermel, G. 2000. Prioritizing Test Cases for Regression Testing. Technical Report 00-60-03, Oregon State University, Feb 2000.

[13] Elbaum, S.; Malishevsky, A.; Rothermel, G. 2001. Incorp-orating Varying Test Costs and Fault Severities into Test Case Prioritization. 23rd ICSE, Toronto, Canada, May 2001.

[14] Ferguson, R.; Korel, B. 1996. The chaining approach for software test data generation. ACM TOSEM 5(1), Jan 1996.

[15] 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.

[16] Ottevanger, I. 1999. A Risk-Based Test Strategy. STARWest, San Jose, CA, USA, Oct 1999.

[17] Pinkster, I.; van de Burgt, B.; Janssen, D.; van Veenendaal, E. 2004. Successful Test Management – An Integral Approach. Springer, Berlin, 2004.

[18] 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.

[19] Reuys, A.; Kamsties, E.; Pohl, K.; Reis, S. 2005. Model-Based System Testing of Software Product Families. 17th CAiSE, Porto, Portugal, June 2005.

[20] Rosenberg, L.H.; Stapko, R.; Gallo, A. 1999. Risk-based Object Oriented Testing. 24th SWE, NASA, Greenbelt, MD, USA, Dec 1999.

[21] Srikanth, H.; Williams, L. 2005. On the economics of requirements-based test case prioritization. 7th EDSER, Shanghai, China, May 2005.

[22] Srikanth, H. 2005. Value-Driven System Level Test Case Prioritization. North Carolina State University, 2005.

[23] Stallbaum, H.; Metzger, A. 2007. Employing Requirements Metrics for Automating Early Risk Assessment. 1st MeReP, Palma, Spain, Nov 2007.

[24] Walton, G. H.; Poore, J. H.; Trammell, C. J. 1995. Statistical Testing of Software Based on a Usage Model. SPE 25(1), Jan 1995, pp. 97-108.