What Techniques Can Be Used for GUI Risk-based Testing?

Behzad Nazarbakhsh and Dietmar Pfahl
Institute of Computer Science, University of Tartu, J. Liivi 2, Tartu 50490, Estonia

Keywords: GUI Testing, GUI Risk-based Testing, Risk Analysis, Risk-based Testing, System under Test, Test-based Risk Analysis.

Abstract: Risk-based testing (RBT) is an approach that uses metrics to find critical parts of software applications under test. In order to understand to what extend RBT has been applied for GUI testing, and to capture the lessons learned, we conducted a literature review. Based on the selected literature, we discuss the advantages that RBT may bring to the various activities involved in testing. Moreover, we analyze the rationale for applying different variants of RBT presented in the selected literature. Finally, we discuss the RBT techniques which can be specifically used for GUI testing.

1 INTRODUCTION

Graphical user interface (GUI) is a fundamental software component with critical impact on how a user perceives a software. This explains why GUI testing is important. GUI testing checks whether the interface of a software application meets the specification. Since GUI testing comes on top of the standard verification and validation activities required to test software applications, there is a need to reduce test effort by making GUI testing efficient and effective.

Testers don't have much interest to repeat testing activities or perform testing more than what is necessary (Garousi, V., Mäntylä, M., 2016). To avoid unnecessary test activities, testers prefer to make a decision about what part of the software application has priority for testing. Making this kind of decision is always critical, hard and uncertain for the testers (Crispin, L., Gregory, J., 2009). Metrics may help to recognize the defect-proneness of system modules, so that extra development, maintenance, and test effort can be directed to those modules (Last, 2005). Risk-based testing (RBT) combines test process with metrics such as probability, time, and impact criteria.

RBT helps to control, manage, and assess the test process during the software development lifecycle (Sharma, 2014). However, it is still unclear to testers when and how it should be practiced and one needs to understand "How far RBT has been practiced for testing the GUI applications" (France, 2016).

The goal of this review is to establish a body of knowledge regarding approaches that combine risk-based testing and GUI testing.

The rest of this paper is organized as follows. In Sect.2, we discuss risk definitions to understand risk-based testing and its concepts. In Sect.3, we describe our methodology and how the research process and data extraction has been done in this study. In Sect.4, we provide the answers to our research questions. Moreover, we identify and discuss different approaches, techniques, methods that have been practiced for implementing RBT. In Sect.5, we discuss threats to validity before concluding in Sect.6.

2 RISK ANALYSIS AND TESTING

Many industrial test projects focus on software risks (Grood, D., Derk, J., 2008). There are different kinds of risks which can impact software negatively such as risks related to the customer business, software interoperability risks and others. For testing software applications based on risks, risk analysis should be carried out to determine the impact of a risk on certain software quality attributes. Moreover, the probability of an identified risk must be assessed.

Suman et al. (2014) provide a list of quality attributes specified in different quality models (Figure 1). These software quality attributes may serve to classify risks. For example, Felderer et al.

Nazarbakhsh, B. and Pfahl, D.
What Techniques Can Be Used for GUI Risk-based Testing?.
DOI: 10.5220/0006416102390248
In *Proceedings of the 12th International Conference on Software Technologies (ICSOFT 2017)*, pages 239-248
ISBN: 978-989-758-262-2

(2014) claim that security, correctness, and functionality are three dominant risk drivers.

S/NO	Attributes	McCall	Boehm	ISO 9126	DEQUALITE	UML	SQuaRE's
1	Accuracy			*		*	
2	Availability/Reliability	*	*	*		*	*
3	Correctness	*					
4	Efficiency	*	*	*		*	*
5	Flexibility	*					
6	Functionality			*	*	*	*
7	Human Engg.		*				
8	Integrity					*	
9	Interoperability	*		*		*	
10	Maintainability	*	*	*			*
11	Modifiability		*		*	*	
12	Portability	*	*	*		*	*
13	Reusability	*			*		
14	Robustness				*		
15	Scalability				*		
16	Security	*		*			
17	Testability	*	*		*	*	*
18	Understandability		*		*	*	
19	Expandability				*		*
20	Learnability				*	*	
21	Simplicity				*		
22	Modularity				*		
23	Usability	*		*	*	*	*
24	Ambiguity						*

Figure 1: Comparison chart of software quality attributes of software quality models.

Test-based risk analysis (TRA) and Risk-based testing (RBT) are two strategies for the combined use of risk analysis and testing (Hettiarachchi, C., Do, H., Choi, B., 2016).

TRA aims to answer questions such as “How safe is safe enough?”, and “Which of several different risks is lowest?” (Papageorgiou, 2015). In the TRA strategy, testing is used to identify, validate and analyze risk results.

RBT analyzes risks to identify and select tests. Moreover, it aids to prioritize and emphasize the appropriate tests during test execution based on the risk of failure, likelihood or impact of failure (Felderer, M., Schieferdecker, I., 2014).

RBT can identify risky software components by finding the probability of risk occurrence via threats on critical requirements or feature complexity. A software component is risky when it has a high probability to fail or its failure will have a serious consequence (Noor, T., Hemmati, H., 2015). In general, the risk of software components can be defined by the probability to fail, and the consequence of the failure, $Risk(f) = P(f) * C(f)$, where f is a software component, Risk(f) is the risk exposure, P(f) is the failure probability and C(f) is the cost of the failure (Bai, 2012).

Inside-Out and Outside-In are two heuristic approaches that have been presented in (Bach, 1999) to explore and analyze the risks in a piece of software. The inside-out approach starts with learning the software component states and identifying the risks associated with them. Outside-In starts with defining the group of risks and then react to the threat in a specific situation.

The outside-in approach collects information about the component to be tested, determines the scalability of the problem, discovers the importance level of a risk, finds the unknown risks that should be recorded and checks the risk distribution (Alam, M., Irshad Khan, A., 2013). Inside-Out and Outside-In approaches not only find the threats and vulnerabilities of software but exploit defective components and analyze failure consequences. In (Shahamiri, R., Nasir, W., 2008), risks are grouped based on three categories for the Outside-In approach. Those approaches, namely a) Quality Criteria b) Generic Risks c) Risk Catalogs.

Into the category "Quality Criteria" all risks have a specific risk driver. Examples of quality criteria are capability, reliability, usability, performance, and localizability. New and existing requirements can be elicited or classified based on quality criteria.

Generic risks are the common risks to all software systems such as complexity, novelty, change, upstream and downstream dependency, criticality, precision, popularity, strategy, third-party software, bugginess, and recent failure history (Bach, 1999).

Finally, risk catalogs are lists of risks that have been identified during past testing activities, e.g., memory clobbering, software misconfigurations, and other issues.

There are RBT techniques to communicate the risks and organize the testing around those risks. Bach (1999) distinguishes three techniques to communicate risks: a) risk watch list, b) risk matrix, c) component risk matrix.

Risk watch list is an approach which assists the test cycle by repeatedly reviewing a collection of the risks. While the risk matrix assists in sorting the risks according to the level of importance. Lastly, the component risk matrix technique is employed for testing the software components based on risks specified during the test.

RBT techniques can jointly be used to effectively control and manage the test process. For example, RBT can be implemented for producing and tracing the log files which records the test progress. Moreover, it can be integrated into the test automation tools to control the execution of test cases or compare the actual outcomes of tests with the predicted outcomes. Therefore, risks and risk-based testing are helpful to assist testers in deciding where and to what extent testing should be automated.

3 METHOD

This study has been conducted as a Systematic Literature Review (SLR) loosely following the guidelines proposed by Kitchenham (2007). We conducted the following steps: 1) Formulation of research questions; 2) Identification of relevant literature; 3) Study quality assessment and data extraction.

3.1 Research Questions

For developing our search strategy, we applied the PICOC method (Population, Intervention, Comparison, Outcome, and Context) proposed by Petticrew et al. (2005). In the following, we present the steps involved in the PICOC method.

Population: Numerous software test projects using RBT.
Intervention: There are various RBT techniques, methods and solutions that can be involved during the test life cycle of the project.
Comparison: Our research is not a comparative study. Therefore, this section has not been answered.
Outcomes: There are RBT techniques and methods that can be employed for GUI testing.
Context: RBT techniques and methods are used for the different types of testing. Testing the GUI applications can gain benefits from RBT techniques.

Following are the main research questions addressed in this study:

(RQ1) What are the benefits of RBT?
(RQ2) What are the testing types and purposes for which RBT has been used?
(RQ3) What techniques exist for performing RBT?
(RQ4) What is the main goal of each selected study, and what key strategies are used to achieve that specific goal?
(RQ5) Which RBT techniques have been specifically used for GUI testing?

3.2 Finding the Relevant Literatures

We used six online data sources in which most of the software engineering articles can be found. These data sources are IEEE Explore, Springer Link, Science Direct, ACM, BASE, and Google Scholar.
We searched for keywords such as “risk-based testing”, “risk analysis”, “risk assessment”, “GUI risked-based testing”, “GUI risk analysis, “GUI risked-based testing framework”, “risk-oriented testing”, “test-driven risks analysis”, “risked-based testing tools” and combinations of those keywords.

3.2.1 The Search Process

The first step of our search process is finding the papers based on the defined search strings. In this step, we found overall 150 papers. We selected the papers for further analysis based on three criteria: 1) Papers that mentioned risk or RBT in their title or abstracts; 2) Papers that were published in the last eight years (2008–2016); and 3) In the case of duplication, we retain only one copy of that paper.

We found 64 papers that met the above criteria. The imprecise or unclear studies were removed from the SLR. Moreover, the content of each paper was assessed to decide whether the paper should be included into SLR or not. The inclusion/ exclusion and assessment criteria are described in the next section. Finally, we looked at the references of the included papers and assessed those as well, which answer our research questions. Finally, we found 22 papers relevant to our research questions. In the following, we label these papers with P01 to P22. These labels are also added to the respective papers in Sect. 7 (Bibliography).

3.2.2 Inclusion and Exclusion Criteria

Our Inclusion Criteria are:
1) The papers are about empirical research in software engineering (SE), testing of software and communicating systems, test automation, GUI testing and risk-based testing knowledge area.
2) The papers describe the approaches combining software risk analysis and software test automation, explicitly GUI risk-based testing.
3) The papers are written in English and published in the peer-reviewed journals, conferences, workshops, and book chapters (published in journals or conference).

Our Exclusion Criteria are:
1) The papers discuss informally risk analysis and testing without presenting a concrete approach.
2) Invalid and incomplete documents such as white papers, technical report, and general web pages.

3.2.3 Classification of Studies

After studying our 22 selected papers, we understood that most papers propose and discuss different types of RBT or benefits of applying RBT for different types of testing. There was only one paper that actually focused on RBT used to test GUIs. Consequently, we classified our papers into 2 categories. This classification is presented in Table 1. “Category 1” comprises papers which propose

solutions to identify and assess risks in the context of software testing. "Category 2" comprises papers that use the RBT approach for various types of testing. Indeed, this classification can help us to find the applicable risk analysis methods and techniques for GUI testing. Finally, after classifying and analyzing all papers, we extracted data to answer our research questions.

Table 1: Papers category.

Category 1	P22, P18, P02, P03, P09, P05, P10, P13, P01, P15, P04, P07, P21, P16
Category 2	P11, P08, P19, P12, P20, P06, P14, P17

3.3 Study Quality Assessment

After classifying the papers, we assessed their quality. In our opinion, the quality of a paper is better whenever: a) It has a high degree of formality; b) It is focusing on concrete and specific contribution; c) It provides the empirical evidence; d) It describes the chosen research method adequately, or e) It describes tool support for the presented method.

We developed a checklist to measure the papers' quality. The compliance of a study with each checklist item was measured using the following scoring system: a) 0 for no compliance; b) 0.5 for partial compliance, and 1 for full compliance. Our checklist was prepared by adopting the questions from checklists that were prepared in (Sulayman, M., Mendes, E., 2009). Lastly, we extracted the relevant data that may be used to answer the SLR's research questions. Following are the questions used in the checklist and the summarized scores present in Table 2.

1. Does the methodology address the research questions?
2. Does the paper discuss any of the previous RBT literature?
3. Is the methodology specified in the paper repeatable?
4. Do the findings of the paper address the original research questions?
5. Are the aims of the research clearly stated?
6. Was the proposed concept, technology, framework used for risk-based testing?
7. Does the paper describe the research method?
8. Does the research use any tools for their study?
9. Does the paper report an empirical study?
10. Does the paper report an evaluation?

Table 2: Assessment scores of selected studies.

Assessment score	Papers
8.0	P18, P09, P12, P01, P20, P15, P07
8.5	P02, P03, P06, P04
9.0	P05, P10, P14, P17, P16
9.5	P19, P13, P22
10.0	P08, P21, P11

4 RESULTS

The following sub-sections summarize the results related to research questions RQ1-RQ5.

4.1 RQ1: What Are the Benefits of RBT?

The answer to RQ1 lists the advantages that RBT may bring to software testing. The main RBT goal is to reduce testing complexity by prioritizing different parts of the system under test. Besides this, RBT optimizes test efforts by reducing the number of test cases, minimizing time and cost of software testing (P11) to enhance the quality of a software product (P16). Moreover, it aids testers to reduce the cost of software maintenance (P21) and determine how much software testing is sufficient to attain the product quality. RBT has been successfully applied to find the right unit tests, integration tests, security vulnerabilities or end to end tests and to measure the security of software (P10). The overall benefits of RBT are summarized in Table 3.

Table 3: Risk-based testing benefits.

Benefits	Papers
To optimize test process.	P11
To reduce the overall cost of the project.	P11, P21
To reduce the number of test cases.	P19
To enhance the quality of software product.	P16
To give insight on how much to test the software.	P22
To find out which parts of the software are critical and have priority to test.	P08
To improve software maintenance.	P21
To increase test coverage.	P22
To discover software vulnerabilities.	P10, P06

4.2 RQ2: What Are the Testing Types and Purposes in Which RBT Has Been Used?

The answer to RQ2 explains different objectives of RBT's usage. Moreover, it shows what type of software testing can be supported by RBT. The risk-based analysis was employed to prioritize the risky parts of the system under test (SUT) or to identify software failures and threats (P18).

The studies such as (P02), (P03) combined the risk analysis information and assessment activities with requirements engineering activities. The sequence-based specification is the pioneer RBT methodologies that built based on requirements of the embedded system (P05). Hettiarachchi et al. (2016)

transformed the use case requirements to FTA via augmenting the behavioral tree with the risk information. *ORTS* tool is implemented for regression testing purpose. This tool selects test cases after analyzing risk which threats the software under development. The risks of a software under developments are classified based on the number of change point excised, change types, invocation counts of change points, and bug history (P12). In Lachmann et al. (2017), test cases prioritization practiced based on the sum of the risks of all actions and considering the risks that not covered by previously chosen test case scenarios. RBT was practiced in (P19), (P13), (P17), (P01) to select or prioritize test cases. The automate test case generation for GUI testing has been appeared in (P08). Entin et al. (2012) proposed the model definition for the purposes of regression and risk-based testing of GUIs. Moreover, the new algorithms for detecting the suitable test case derivation was discussed.

Behavior Engineering (BE) method was practiced to derive requirements models via using behavior trees. This technique employed UML profile, risk extension of behavior trees, tooling to capture risk matrices and testing directives to generate test cases (P20). Risk-Based Vulnerability Testing (*RBVT*) is another framework proposed to assess risks and generate test cases by practicing the risk metrics and vulnerability test patterns (P06). Convergence of risk analyzing, statistical service-oriented computing and semantic engineering can be used to automate web service integration testing. The proposed technique in (P04) ranked and selected test cases for web service testing via dynamic and online risk measurement and control. The estimation of probabilities, the specification of dependencies, dynamic updating of estimates, and sensitivity evaluation of group testing rule parameters are the activities which can be used for web service integration.

The risk-based approach was also applied to test the web services transactions quality and distinguishing situations in which transaction require to be tested in (P07). There are few studies such as (P11), (P15) that applied RBT to improve the risk management process while testing the software. The risk management process can find the most important defects earlier than the traditional approaches (P11). XRISK is the model that was developed to analyze the risk of software failure based on finding and analyzing risks in the source code (P21). The designed risk model comprises the metrics related to the static structure of the source code and the dynamic test coverage of the code (P21). Risk-based security testing and Risk-based Fuzz testing were used in (P10), (P06) to test the implementation of software security flaws. It could handle malicious input and focused on certain security aspects. Indeed, a combination of RBT and security testing assist test process to concentrate on the certain software vulnerabilities.

We classified different approaches that have been practiced based on their objectives and type of software testing in Table 4 and Table 5. Table 4 shows that most of the studies have been tried to practice RBT for risk analysis or improving test activities such as selecting, prioritizing and generating test cases. On the other hand, Table 5 presents different software testing types that may gain benefits from RBT.

Table 4: Classification based on purpose of studies.

Purpose	Studies
Risk analysis and assessment	P18, P02, P03, P22, P09, P05, P08, P04, P07, P10, P13, P06
Test case prioritization and selection	P19, P10, P13, P14, P12, P17, P01
Test case generation	P22, P08, P20, P16, P06
Risk management	P11, P15
Source code risk analysis	P21
Requirement risks identification	P02, P18, P03

Table 5: Classification based on type of software testing.

Software testing type	Studies
Security testing	P10, P06
Regression testing	P08, P12
GUI testing	P08
User acceptance test	P02, P18, P03, P11, P05
Web service integration test	P04, P07

4.3 RQ3: What Approaches Exist for Performing RBT?

RQ3 investigates the various approaches and techniques that have been practiced in the domain of risk-based testing. Risk identification and risk analyzing techniques are the most challenging part of RBT implementation. Risk analysis methods and techniques assist in managing and mitigating risks. Moreover, it can be used for learning the relations between the risks. For this reason, part of our answer to this question is finding the techniques that can be used for risk analysis.

It is important to find the risk analysis methods and techniques to manage and mitigate risks. Lund et al. (2011) introduced two risk analysis methods

(OCTAVE and CRAMM) to show that how risk analysis can be carried out. Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) is the risk-based strategic assessment and planning method which identifies the critical assets and threat profiles as a key component to mitigate risks. CCTA Risk Analysis and Management Method (CRAMM) analysis and manages risks by identifying, assessing risk and finding the appropriate treatments for hazards.

There are risk analysis techniques which can be integrated into diverse methods to addresses some aspects of the risk analysis process. Hazard and Operability (HazOp) is a risk identification technique that was practiced to analyzes the hazards and operational concerns related to that (P03). In (P09), Failure Mode Effect Analysis/Failure Mode Effect and Criticality Analysis (FMEA/FMECA) techniques were practiced to detect a system's possible failure modes and determines their consequences. In the other word, these techniques determine which failures of a system's components may lead to which system faults as well as to which consequences and by which countermeasures those consequences can be mitigated (P22).

Fault Tree Analysis (FTA) is widely used for analyzing risks. It assists in identifying the potential causes in the components which may lead to hazards. In (P09) risk analysis has been combined FTA and FME to include structural views of the system under consideration into defect analysis. Event Tree Analysis (ETA) and Attack Tree are the other event tree techniques like FTA. ETA determines the probability of consequences once a risk occurred by specifying every detail about the expected outcome of an unwanted incident. Attack Tree is a formal and methodical way of describing the security of a system based on the exposable attacks (P15).

Cause-Consequence Analysis (CCA) is a graph-based technique that can be practiced for risk analysis. This model combines the features of fault trees and event trees. Even though this technique is introduced in the literature review of (P15) but the selected papers in this study have not practiced this technique. Bayesian Network is an important acyclic graph-based technique demonstrates the relations between causes and effects. Moreover, it can be used as a mathematical model for computing the probabilities. This technique is used in (P01) to predict the number of faults in the software component. Finally, Markov analysis is a method that looks at the system as several states and determines and assigns probabilities to the changes between these states. This technique is suitable for analyses systems that may fail partially. In (P08), a set of algorithms based on Markov chains were implemented to calculate fair transition probabilities. The risk analysis techniques are mostly practiced for preventing the unnecessary rejected test cases creation, identifying the hazards and hazardous states and defect removal in test engineering. The above-discussed risk analysis techniques are summarized in Table 6.

Table 6: Risk analysis model and related techniques.

Risk analysis models	Risk analysis techniques	Papers
Table-based	HazOp, FMEA, FMECA	P03, P09, P15, P22
Tree-based	FTA, ETA, Attack Trees	P09, P05, P15, P07, P16
Graph-based	CCA, Bayesian Network, Markov Analysis	P08, P01, P20

After finding different RBT techniques for analyzing risks, we discovered that most of papers practiced different model-based testing to stimulate the testing process towards automation. Consequently, identifying and understanding the purpose of different practiced models in each study is necessary. Model-Based Testing (MBT) have been practiced in many studies to analyze the risks for test cases derivation or execution. Approaches that practiced in (P22), and (P09) assume that the precise test models are available to analysis and manage risks.

RiteDAP is an MBT approach that augmented the risk information analysis in the model to prioritize and generate test cases (P19). APSP is another MBT approach that employed the non-risk-based prioritization strategies such as Random Prioritization (RP), Optimal Prioritization (OP), Total Action Coverage Prioritization (TACP) and Additional Action Coverage Prioritization (AACP). Then, a metric algorithm was practiced for analyzing, evaluating and prioritizing the risk in the model (P09). Model-Based Statistical Testing (MBST) is the model that employed to decline the complexity of the test problem by using risks (P05).

Bayesian statistical (BST) is a model that powers the selection of test cases based on the prediction of a risk decrement (P01). This model assists in integrating test framework with the supporting decisions model. The selection and prioritization of this model had been done through calculating defect probability, using statistical model covering Bayes Risk (BR) decision criterion, cost factors, likelihood functions and operating characteristics.

In (P22), RBT and non-RBT approaches were practiced to generate critical test cases. In the first method, non-critical test cases were filtered after test case generation. In the second method, RBT was practiced to derive out the critical test cases by using Markov analysis. After studying the evaluation of this approach, it is claimed that in second method the test coverage is increased and the critical faults can be detected earlier.

The system behavior model was integrated with FTA to generate the test cases out of misuse cases in (P16). In this model, the risk information was extracted from fault tree analysis and integrated into system state charts. Then, the attack patterns and threat profiles techniques were employed to generate test cases. The risk-based regression test model combined risk analysis, test case selection and end-to-end test scenario selection with each other to enhance the accuracy of test cases selection (P17). In this model, the test cases are selected by calculating severity probability for each test case; and applying risk exposure technique that was proposed in (Amland, S.). Moreover, for calculating the risk exposure, the highest risk test scenario was selected based on integrated traceability that employed in (P12), and (P17).

There are few studies which tailored algorithms to enhance the performance of risk analysis and boost the test case generation. For instance, the greedy algorithm was implemented to analyses the group of tests by using risk metrics. In (P08), new graph-based algorithm integrated with Markov chains to enhance the critical test case generation.

There are models such as CORAS model (P15), pattern-based approach, Based Security Risk Assessment (TBRA) or Risk-Based Security Testing (RBST) that exploit the vulnerabilities, determine the data protectability and maintain the functionalities from the software security perspective (P10). For generating the test cases from security models there are models like a) Complete model, b) Partial model, and c) Missing model. The Complete model derives automatically security tests from a formal model. In the Partial model, security tests are partially generated automatically and partially added manually. Missing model is not practicing any test models and the test generation is missed in this model. Testing approaches attempted to integrate these models with the software product risks. The behavioral fuzzing is the risk-based fuzzing approach which generates tests and finds authentication bypass vulnerabilities. In this approach, the behavior model was augmented with security-related information where vulnerabilities might be relevant (P16). The model-based security testing models are classified in Table 7.

Table 7: Model-based security testing classification.

	Complete model	Partial model	Missing model
Risks integrated into models	Automated RBT security test generation	Risk enhanced scenario-based MBT	Adapted RBT
Risks not integrated into models	Adapted MBT	Scenario-based MBT	Individual knowledge

4.4 RQ4: What Is the Main Goal of Each Selected Study, and What Key Strategies Are Used to Achieve That Specific Goal?

RQ4 helps identify the key reasons of practicing RBT in each study. Table 8 explains the goal of each study and Table 9 explains the strategy which was practiced to achieve those goals.

Table 8: Goal of each study.

Paper	Goal
P18	Testing software based on the priority of its requirements.
P02	Prioritizing the test cases of software.
P03	Discovering risk information from software requirement and integrating them into the test design process.
P22	Generating high-risk test cases which can trigger the certain critical software situation.
P09	Integrating requirements engineering activities into the tasks of system test engineering and generating test cases based on the founded hazardous states in the software.
P05	Reducing the complexity of the test problem represented by a large number of possible test cases.
P08	Automatically derive test cases which cover the most critical part of a graphical user interface.
P19	Risk-based test case derivation and prioritization.
P10	Integration of risk assessment with security testing as a single process.
P13	Prioritizing what to test against the list of requirements.
P14	Identifying the software components failure and computing their impact probabilities.
P12	Generating the optimized regression test suite.
P17	Identifying, prioritizing, and selecting test procedures after identifying the software risks.
P01	Risk-based test selection and prioritization.
P20	Combining the test generation directives and risk level to generate risk-optimized test suites.

Table 8: Goal of each study (cont.).

Paper	Goal
P06	Optimizing test process by integrating risk analysis and security testing.
P11	Reducing the number of risk items that can be used for software risk estimation and simplify test case prioritization.
P15	Managing risks that may be manifested in the software by identifying, analyzing and assessing the software vulnerabilities.
P04	Ordering set of test cases to detect bugs and evaluate web service environment.
P07	Testing the transactional requirements in web service environment.
P21	Computing the risk index of specified function then verifying the number of functions that have high risks of failure in a source code.
P16	Proposing RBT approaches to derive test cases from critical functions, and requirements.

Table 9: Practiced Strategy in each study.

Paper	Strategy
P18	Formulating an algorithm based on the severity and the probability of risk factors that may be founded in the software requirement.
P02	Formulating a method for finding the most important or poorest parts of the software product based on its cost of failure.
P03	Combining RBT with MBT by using the UML Testing Profile (UTP2).
P22	Deriving test cases from MBST and constructing the critical test cases by applying algorithmic method.
P09	Build a test model and analyze the hazards that obstruct the software safety goals.
P05	Combining combinatorial and model-based techniques to auto-generate test cases by focusing on critical function.
P08	Minimize the number of test cases based upon the risk calculation.
P19	Prototyping a tool to prioritize the system test cases using RBT.
P10	Proposing a tool-based approach that combines the notion of risk-assessment with a pattern-based approach.
P13	Generating test cases after prioritizing use cases based on the identified risks.
P14	Using delta-oriented architectures to prioritize the test cases after computing a failure probability.
P12	Prototyping the tool for generating the test suite. Capturing the runtime traces of test execution and identifying the change points during build update.
P17	Using risk graph to set of statements about the likelihood of occurrence of events and the consequence of events occurring. Then prioritizing and selecting tests based on the severity and confidence of the statement.
P01	Selecting the test cases using Bayesian decision theory to predict the risks.
P20	Proposing methodology to risk-based testing that deals with the transition from risk management and requirements engineering.
P06	Generating test cases by identifying test patterns from different threat scenarios.
P11	Estimating risks by correlating with requirements. Then, calculating the risk exposure for the requirements and risk items to prioritize requirements and test cases.
P15	Proposing risk model to evaluate the software security risks.
P04	Scoring and selecting test cases through identifying risks of software features.
P07	Develop a model to pattern the web service transaction, then applying model-based testing techniques over that model.
P21	Proposing a static and dynamic risk model using metrics that are either related to the structure of source code or test coverage of the code.
P16	Using fault trees method and integrating FTA into state-based behavior models.

4.5 RQ5: Which RBT Techniques Have Been Specifically Used for GUI Testing?

RQ5 addresses the RBT techniques which have been specifically used for GUI testing. Moreover, it points out the future research directions in the domain of risk-based GUI testing. Among our selected papers, only (P08) concentrated on model risk-based testing of GUI. The main objective of (P08) was understanding that how risk analysis can derive the test cases of the critical part of GUI. The critical part of GUI is the portion of GUI that may contain bugs such as GUI crash or some wrongly displayed value which may be discovered during the comparison with the target value.

Combinations of different algorithms are proposed to reduce the size of the test suite, which is: a) Markov chains and random walk algorithms and b) Chinese postman algorithm. Some researchers adopted risk-based prioritization algorithms which are proposed in (P19). Besides that, an algorithm named “Adventurer`s Journey” was proposed to generate risk-based test cases directly instead of prioritizing the test cases (P08). Finally, Entin et al. (2012) claimed, for the future development of risk-based GUI testing, researchers should concentrate on defining the risks in usage models, making the risk calculation more realistic and creating the traceability between requirements and models.

5 THREATS TO VALIDITY

There are several threats to validity in our review. There is a possibility that some papers could not be found because of the design of the search query and time constraints. Moreover, only one researcher was involved in analyzing, filtering and classifying the literature. Consequently, the risk of bias and inaccuracy of data extraction cannot be ignored. Although our selected data sources are well-known sources with the availability of the highest amount of papers in our search domain, there are possibilities of missing papers related to GUI risk-based testing.

6 CONCLUSIONS

In this literature, we identified and studied 22 scientific papers that concentrated on risk-based testing. We recognized different techniques, methods, and algorithms that can be used for RBT. This review has attempted to understand how far RBT has been practiced for GUI testing, how much GUI risk-based testing is advance and what techniques can be applied to it. We confronted with the inadequate collection of the publication in the domain of GUI risk-based testing. Indeed, the number of studies that focus on GUI risk-based testing are few. Among all the papers that we collected in SLR, most of RBT studies was concentrating on regression testing, security testing, and user acceptance testing. We found only one paper (P08) that was specifically discussing an approach to perform GUI risk-based testing.

Our results indicate that the potential of prioritizing and detecting the most critical parts of GUI applications could make RBT an asset for GUI testing. Indeed, it assists testers to identify the dangerous test areas and prioritize the critical GUI features. Moreover, it can be used to estimate the risks values of each feature and specify tests for the highest risk features. Finally, analyzing the risks of the SUT, modeling threat/failure, and presenting the tests for the severe threats are the benefits that it brings to identify the part of a system failure. We listed a set of algorithms such as Markov chains, random walk, Chinese postman that can be used to achieve the above goals.

REFERENCES

Adorf, H., Felderer, M., Varendorff, M., Breu, R., 2015. A Bayesian Prediction Model for Risk-Based Test Selection. *Euromicro Conference on Software Engineering and Advanced Applications,* pp. 374-382.

Alam, M., Irshad Khan, A., 2013. Risk-based Testing Techniques: A Perspective Study. *International Journal of Computer Applications,* Volume 65, pp. 33-41.

Ali, S., Yue, T., Hoffmann, A., Wendland, M., 2014. How does the UML Testing Profile Support Risk-Based Testing. *IEEE International Symposium on Software Reliability Engineering Workshops,* Volume 13, pp. 311-316.

Amland, S., 1999. *Risk Analysis Fundamentals and Metrics for Software Testing.* Barcelona, 5th International Conference EUROSTAR 99.

Bach, J., 1999. *Heuristic Risk-Based Testing.* s.l.:Software Testing and Quality Engineering Magazine.

Bai, X., 2012. Risk Assessment And Adaptive Group Testing of Semantic Web Service. *International Journal of Software Engineering and Knowledge Engineering,* 22(5), pp. 595-620.

Bauer, T., Eschbach, R., Größl, M., Hussain, T. , Streitferdt, D., Kantz, F., 2009. *Combining combinatorial and model-based test approaches for highly configurable safety-critical systems.* Enschede, s.n.

Botella, J., Legeard, B., Peureux, F., Vernotte, A., 2014. Risk-Based Vulnerability Testing Using Security Test Patterns. In: *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*. Corfu: Springer Berlin Heidelberg, pp. 337-352.

Casado, R., Tuya, J., Younas, M., 2010. Testing Long-lived Web Services Transactions Using a Risk-based Approach. *International Conference on Quality Software,* pp. 337-340.

Crispin, L., Gregory, J., 2009. *A practical guide for testers and agile teams.* Boston: Pearson Education, Inc..

Entin, V., Winder, M., Zhang, B., Christmann, S., 2012. Introducing Model-Based Testing in an Industrial Scrum Project. *Proceeding AST '12 Proceedings of the 7th International Workshop on Automation of Software Test,* pp. 43-49.

Felderer, M., Schieferdecker, I., 2014. A taxonomy of risk-based testing. *International Journal on Software Tools for Technology Transfer,* 18(Springer Berlin Heidelberg), p. 559–568.

France, H., 2016. *Defining the right testing strategy,* Toronto: QA Consultants.

Garousi, V., Mäntylä, M., 2016. When and what to automate in software testing? A multi-vocal literature review. *information and Software Technology,* Volume 76, pp. 92-117.

Gleirscher, M., 2011. Hazard-based Selection of Test Cases. *Proceedings of the 6th International Workshop on Automation of Software Test,* pp. 64-70 .

Grood, D., Derk, J., 2008. Test Risk Analysis. In: *TestGoal.* Leiden: Springer, pp. 101-108.

Großmann, J., Schneider, M., Viehmann, J., Wendland, M., 2014. Combining Risk Analysis and Security Testing. In: *Leveraging Applications of Formal Methods Verification and Validation Specialized Techniques and*

Applications. Corfu: Springer-Verlag Berlin Heidelberg, p. 322–336.

Hettiarachchi, C., Do, H., Choi, B., 2016. Risk-based test case prioritization using a fuzzy expert system. *Information and Software Technology,* pp. 1-15.

Huang, S., Zhu, J., Ni, Y., 2009. ORTS: A Tool for Optimized Regression Testing Selection. *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications,* pp. 803-804.

Jørgensen, K., 2008. *A Software Tool for Risk-based Testing,* Trondheim: Department of Computer and Information Science, NTNU.

Kitchenham, B., 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering,* Durham: EBSE Technical Report.

Lachmann, R., Beddig, S., Lity, S., Schulze, S., Schaefer, I., 2017. Risk-Based Integration Testing of Software Product Lines. *VAMOS '17 Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems,* pp. 52-59.

Last, M., 2005. Data Mining For Software Testing. In: *Data Mining and Knowledge Discovery Handbook.* US: Springer Science+Business Media, Inc., pp. 1239-1247.

Lund, M., Solhaug, B., Stølen, K., 2011. *Model-Driven Risk Analysis The CORAS Approach.* Berlin: Springer-Verlag Berlin Heidelberg.

Nazier, R., Bauer, T., 2015. Automated Risk-based Testing by Integrating Safety Analysis Information into System Behavior Models. *23rd International Symposium on Software Reliability Engineering Workshops,* pp. 213-218.

Noor, T., Hemmati, H., 2015. *Mining Test Case Traces to Improve Risk-Driven Testing.* Montréal, s.n.

Papageorgiou, S., 2015. *Risk Acceptance Criteria and Risk Based Damage Stability. Final Report, part 1: Risk Acceptance Criteria,* Høvik: European Maritime Safety Agency.

Petticrew, M., and Roberts, H., 2005. *Systematic Reviews in the Social Sciences: A Practical Guide.* UK: Wiley-Blackwell.

Scarfone, K., Souppaya, M., Cody, A., Orebaugh, A., 2008. *Technical Guide to Information Security Testing and Assessment,* Gaithersburg: National Institute of Standards and Technology.

Seehusen, F., 2014. A Technique for Risk-Based Test Procedure Identification, Prioritization and Selection. In: *Leveraging Applications of Formal Methods, Verification and Validation.* Corfu: Springer Heidelberg, p. 277–291.

Shahamiri, R., Nasir, W., 2008. *Intelligent and Automated Software Testing Methods Classification.* Malaysia, s.n.

Sharma, P., 2014. Risk Based Testing: Technique for Risk-based Test Case Generation and Prioritization. *International Association of Scientific Innovation and Research (IASIR),* Volume 14, pp. 60-65.

Srivastva, P., Kumar, K., Raghurama, G., 2008. Test Case Prioritization Based on Requirements and Risk Factors. *ACM SIGSOFT Software Engineering Notes,* 33(4).

Stallbaum, H., Metzger, A., Pohl, K., 2008. An Automated Technique for Risk-based Test Case Generation and Prioritization. *Proceedings of the 3rd international workshop on Automation of software test,* pp. 67-70.

Sulayman, M., Mendes, E., 2009. A Systematic Literature Review of Software Process Improvement for Small and Medium Web Companies. *International Conference on Advanced Software Engineering and Its Applications,* pp. 1-8.

Suman A., Manoj W., 2014. A Comparative Study of Software Quality Models. *International Journal of Computer Science and Information Technologies,* Volume 5, pp. 5634-5638.

Wendland, M., Kranz, M., Schieferdecker, I., 2012. A Systematic Approach to Risk-Based Testing Using Risk-annotated Requirements Models. *The Seventh International Conference on Software Engineering Advances,* pp. 636-642.

Wong, W., Qi, Y., Cooper, K., 2008. Source Code-Based Software Risk Assessing. *Symposium on Applied Computing,* pp. 1485-1490.

Xu, W., Deng, L., Zheng, Q., 2012. Annotating Resources in Sequence Diagrams for Testing Web Security. *New Generations (ITNG), Ninth International Conference on Information Technology,* pp. 873- 874.

Zimmermann, F., Eschbach, R., Kloos, J., Bauer, T., 2009. *Risk-based Statistical Testing: A Refinement-based Approach to the Reliability Analysis of Safety-Critical Systems.* Toulouse, 12th European Workshop on Dependable Computing.