Evaluating finite state machine based testing methods on RBAC systems

Carlos Diego Nascimento Damasceno

Carlos Diego Nascimento Damasceno

Evaluating finite state machine based testing methods on RBAC systems

Master dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Adenilso da Silva Simão

USP – São Carlos
June 2016

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

D155e

Damasceno, Carlos Diego Nascimento
Evaluating finite state machine based testing methods on RBAC systems / Carlos Diego Nascimento Damasceno; orientador Adenilso da Silva Simão. - São Carlos - SP, 2016.
96 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2016.

1. Model Based Testing. 2. Test Prioritization. 3. Finite State Machine. 4. Access control. 5. RBAC. I. Simão, Adenilso da Silva, orient. II. Título.

Carlos Diego Nascimento Damasceno

Avaliação de métodos de teste baseado em máquinas de estados finitos em sistemas RBAC

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Adenilso da Silva Simão

USP – São Carlos
Junho de 2016

ACKNOWLEDGEMENTS

- Agradeço a Deus, por me conceder a força, determinação e paciência necessárias para superar todos os desafios que enfrentei até agora.
- Agradeço aos meus pais, Eloi e Lucia, por todo o amor e incentivo. Tenho muito orgulho de vocês! Serei eternamente grato por tudo! Amo vocês!!!
- A todos os meus tios, tias, primos e primas, que mesmo estando longe não deixaram de dar palavras de incentivo até nas horas mais difíceis.
- Ao meu orientador Adenilso, por quem tenho uma imensa admiração e respeito, deixo o meu muito obrigado pela sua dedicação.
- Aos demais professores e funcionários do ICMC-USP com quem tive algum contato direto ou indireto, também deixo meus agradecimentos. Em especial, aos professores Delamaro, Masiero, Simone e Elisa.
- A todos os meus amigos da USP e parceiros de laboratório, em especial, Brauner, Daniel, Rachel, Clausius, Stevão, Valdemar, Abdalla, Faimison, Sidgley, Sofia, FCarlos, Aline, Lydia e Frota, que de alguma forma também me ajudaram.
- À Kamila, por toda a sua atenção, carinho e paciência.
- Aos meus amigos de Belém, pelas muitas conversas e que também ajudaram a fazer minha vida em São Carlos mais feliz.
- A todos os meus demais amigos não citados, mas que também são essenciais na minha vida.
- E ao CNPq, pelo apoio financeiro.

“Mude.
Mas comece devagar,
porque a direção é mais importante
que a velocidade.”
(Edson Marques)

RESUMO

DAMASCENO, C. D. N.. **Evaluating finite state machine based testing methods on RBAC systems**. 2016. 96 f. Master dissertation (Master student Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Controle de Acesso (CA) é um dos principais pilares da segurança da informação. Em resumo, CA permite assegurar que somente usuários habilitados terão acesso aos recursos de um sistema, e somente o acesso necessário para a realização de uma dada tarefa será disponibilizado. Neste contexto, o controle de acesso baseado em papel (do inglês, *Role Based Access Control* - RBAC) tem se estabelecido como um dos mais importante paradigmas de controle de acesso. Em uma organização, usuários recebem responsabilidades por meio de cargos e papéis que eles exercem e, em sistemas RBAC, permissões são distribuídas por meio de papéis atribuídos aos usuários. Apesar da aparente simplicidade, enganos podem ocorrer no desenvolvimento de sistemas RBAC e gerar falhas ou até mesmo brechas de segurança. Dessa forma, processos de verificação e validação tornam-se necessários. Teste de CA visa identificar divergências entre a especificação e o comportamento apresentado por um mecanismo de CA. Teste Baseado em Modelos (TBM) é uma variante de teste de software que se baseia em modelos explícitos de especificação para automatizar a geração de casos testes. TBM tem sido aplicado com sucesso no teste funcional, entretanto, ainda existem lacunas de pesquisa no TBM de requisitos não funcionais, tais como controle de acesso, especialmente de critérios de teste. Nesta dissertação de mestrado, dois aspectos do TBM de RBAC são investigados: métodos de geração de teste baseados em Máquinas de Estados Finitos (MEF) para RBAC; e priorização de testes para RBAC. Inicialmente, dois métodos tradicionais de geração de teste, W e HSI, foram comparados ao método de teste mais recente, SPY, em um experimento usando políticas RBAC especificadas como MEFs. As características (número de *resets*, comprimento médio dos casos de teste e comprimento do conjunto de teste) e a efetividade dos conjuntos de teste gerados por cada método para cinco políticas RBAC foram analisadas. Posteriormente, três métodos de priorização de testes foram comparados usando os conjuntos de teste gerados no experimento anterior. Neste caso, um critério baseado em similaridade RBAC foi proposto e comparado com a priorização aleatória e baseada em similaridade simples. Os resultados obtidos mostraram que o método SPY conseguiu superar os métodos W e HSI no teste de sistemas RBAC. A similaridade RBAC também alcançou uma detecção de defeitos superior.

Palavras-chave: Teste Baseado em Modelos, Priorização de testes, Máquinas de Estados Finitos, Controle de Acesso, RBAC.

ABSTRACT

DAMASCENO, C. D. N.. **Evaluating finite state machine based testing methods on RBAC systems**. 2016. 96 f. Master dissertation (Master student Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Access Control (AC) is a major pillar in software security. In short, AC ensures that only intended users can access resources and only the required access to accomplish some task will be given. In this context, Role Based Access Control (RBAC) has been established as one of the most important paradigms of access control. In an organization, users receive responsibilities and privileges through roles and, in AC systems implementing RBAC, permissions are granted through roles assigned to users. Despite the apparent simplicity, mistakes can occur during the development of RBAC systems and lead to faults or either security breaches. Therefore, a careful verification and validation process becomes necessary. Access control testing aims at showing divergences between the actual and the intended behavior of access control mechanisms. Model Based Testing (MBT) is a variant of testing that relies on explicit models, such as Finite State Machines (FSM), for automatizing test generation. MBT has been successfully used for testing functional requirements; however, there is still lacking investigations on testing non-functional requirements, such as access control, specially in test criteria. In this Master Dissertation, two aspects of MBT of RBAC were investigated: FSM-based testing methods on RBAC; and Test prioritization in the domain of RBAC. At first, one recent (SPY) and two traditional (W and HSI) FSM-based testing methods were compared on RBAC policies specified as FSM models. The characteristics (number of resets, average test case length and test suite length) and the effectiveness of test suites generated from the W, HSI and SPY methods to five different RBAC policies were analyzed at an experiment. Later, three test prioritization methods were compared using the test suites generated in the previous investigation. A prioritization criteria based on *RBAC similarity* was introduced and compared to random prioritization and simple similarity. The obtained results pointed out that the SPY method outperformed W and HSI methods on RBAC domain. The RBAC similarity also achieved an Average Percentage Faults Detected (APFD) higher than the other approaches.

Key-words: Model Based Testing, Test Prioritization, Finite State Machine, Access control, RBAC.

LIST OF FIGURES

LIST OF SOURCE CODES

LIST OF TABLES

CONTENTS

CHAPTER

1

INTRODUCTION

1.1 Context

Preserving confidentiality, integrity and availability of personal and critical data has become a mandatory requirement for most industrial-scale information systems. To fulfil this requirement, access control mechanisms can be used to enforce security policies and protect data (JANG-JACCARD; NEPAL, 2014). In short, access control ensures that only intended users can access data and only the permission required to accomplish some task will be given. In this context, the Role-Based Access Control (RBAC) model has been established as one of the most important paradigms of access control and the *de-facto* approach for implementing access control (FERRAIOLO; KUHN; CHANDRAMOULI, 2007). The RBAC model is conceptually simple: in an organization, users receive responsibilities and privileges through roles; analogously, in RBAC domain, permissions are granted via roles assigned to users. Despite its simplicity, the RBAC model can reduce the complexity of security management routines by grouping privileges in roles (SAMARATI; VIMERCATI, 2001). Nevertheless, mistakes can occur during the development of RBAC systems, threatening users' privacy, leading to faults, denial of access or either security breaches. Therefore, a careful verification and validation must be executed.

In verification and validation, software systems can be executed using test inputs and the obtained results can be compared with the expected outcomes to detect non-conformances between the implemented system and the specified behavior (AMMANN; OFFUTT, 2008). This process is called *software testing* and aims at detecting *faults* that are consequences of mistakes occurred during the design and implementation of the System Under Test (SUT) (IEEE, 1990). The software testing process follows four steps (MOUELHI; KATEB; TRAON, 2015): test generation, test selection, test prioritization, and test assessment.

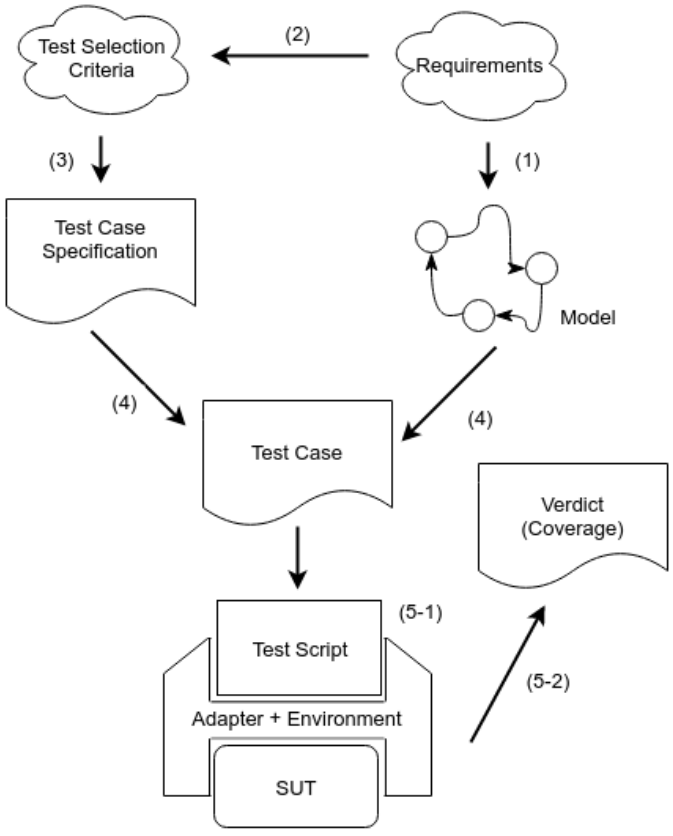
Test generation consists on the stage in which *test cases* are designed based on some *coverage criteria*. A *test case* is composed by test inputs and the expected results necessary for

a complete execution of the SUT. A set of test cases is called *test suite*. A *coverage criteria* consists on a collection of rules which impose requirements that must be satisfied or covered by one test suite.

On real-world applications, a large number of test cases tends to be generated and often, due to time and resources constraints, only a subset of the test cases can be performed. In this case, two solutions are often performed: *Test selection*, which focuses on selecting a fixed number of tests; or *Test prioritization*, which aims at ordering test cases in terms of test criteria. *Test assessment* is the last step in which the fault-detection capability of the test cases is evaluated.

Model Based Testing (MBT) is a variant of software testing which relies on explicit models that encode the intended behavior and/or the environment of the SUT and automatize software testing steps, such as test generation (UTTING; PRETSCHNER; LEGEARD, 2012). The generic process of MBT consist of five steps, illustrated as follows in Figure 1.

Figure 1 – Generic process of Model-Based Testing



Source: Adapted from Utting, Pretschner and Legeard (2012).

The first step consists of (1) creating an explicit *test model* of the SUT based on the existing requirements specification. Software requirements are also used as reference for defining (2) test selection criteria. The test selection criteria are responsible for guiding the automatic test generation process. They can relate to functionalities of the SUT, the structure of the test model, data coverage heuristics, pure randomness, or well defined sets of faults, named *fault model*

(JIA; HARMAN, 2011). Once defined a criterion, test cases can be depicted as (3) test case specifications. Essentially, a test case specification consists on an abstract and non-executable description of the test cases. Given a test case specification and the model of the SUT, the automatic test generation process can be performed to (4) obtain concrete test cases. Concrete test cases are executed as (5-1) test scripts using an adapter which supports the automatic test execution process under a given test environment. The test execution process can also be manual. After execution, a test verdict (5-2) can be given to report test coverage and the acceptance or failure of test cases.

MBT has been investigated as an alternative for testing security requirements, such as access control, specially using state based notations, such as Finite State Machines (FSM). FSM is one of the most used transition based notations in the domain of Model Based Security Testing (MBST), the MBT of security requirements (DAMASCENO; DELAMARO; SIMÃO, 2014; FELDERER *et al.*, 2015). Test generation approaches for RBAC policies specified as FSM models have been investigated and, although costly, they have been very effective on detecting faults (MASOOD *et al.*, 2009). Recently, Endo and Simao (2013) showed that recent FSM-based testing methods, such as SPY (SIMÃO; PETRENKO; YEVTUSHENKO, 2009), can reduce the overall cost of test suites compared to traditional methods, such as W (CHOW, 1978) and HSI (PETRENKO; BOCHMANN, 1995), when randomly generated FSMs are taken as SUT. Besides, Cartaxo, Machado and Neto (2011) investigated test prioritization approaches based on similarity to FSM-based testing and their results pointed out that similarity metrics can be more effective than random approaches. In addition, Bertolino *et al.* (2015) obtained similar results while investigating similarity based approaches for test prioritization in the domain of the eXtensible Access Control Markup Language (XACML), an XML notation for specifying access control policies (OASIS, 2013).

1.2 Problem Statement and Motivation

Although the outcomes of Endo and Simao (2013) are important from a theoretical point of view, they cannot be generalized to the RBAC domain, since the resemblance between randomly generated FSMs and these specifying RBAC policies is unclear. Therefore, there is a lack of evidence about how recent and traditional FSM-based testing methods behave on FSM models specifying RBAC policies. Moreover, since the effectiveness of test coverage criteria is strongly related to its ability to represent specific domain faults (FELDERER *et al.*, 2015), there is no guarantee that test prioritization approaches based on similarity metrics can be as effective in RBAC domain as they were on XACML and random FSM domains. Felderer *et al.* (2015) also emphasize that there is a room for research to better understand which variants of coverage criteria can yield effective and efficient test cases in the context of MBST.

1.3 Research Objectives

Given the previously discussed research gaps on FSM-based testing and similarity-based test prioritization for RBAC, the following general research objective was defined to this Master's Dissertation: *Investigating test criteria to support FSM-based testing processes in the RBAC domain*. Furthermore, the following topics were studied as specific research objectives:

- ***On comparing FSM-based testing methods on RBAC***: Comparing by means of an experiment the characteristics (number of resets, average test case length and test suite length) and effectiveness of test suites generated by recent and traditional FSM-based testing methods from RBAC policies specified as FSM models;
- ***On investigating test prioritization criteria on RBAC***: Investigating and comparing by means of an experiment similarity-based test prioritization approaches for RBAC testing.

1.4 Summary of the Obtained Results

The main contributions of this Master's Dissertation are described bellow:

- Based on (MASOOD *et al.*, 2009) and (ENDO; SIMAO, 2013), an experimental study was designed and performed to compare FSM-based testing methods on RBAC. The W, HSI and SPY methods were applied on five RBAC policies specified as FSM models and the characteristics and effectiveness of the test suites generated were evaluated. As test characteristics, the test suite length, average test case length and number of resets were considered. The test effectiveness was measured using the fault detection ratio on the domain of the RBAC fault model (MASOOD *et al.*, 2009). The proposed experimental protocol and the analysis of the obtained results are respectively presented in Sections 3.1 and 3.2.
- A test prioritization criteria named *RBAC similarity* was proposed based on the similarity metrics for test prioritization in the domains of FSM and XACML testing (CARTAXO; MACHADO; NETO, 2011; BERTOLINO *et al.*, 2015). The *RBAC similarity* consists of a criteria to evaluate the similarity and the applicability degree of test cases to RBAC policies under test. The *RBAC similarity* criteria is presented in Section 4.1.
- An experimental study was designed and performed to investigate test prioritization criteria on the RBAC domain. The *RBAC similarity* was compared to simple similarity and random prioritization. The previously generated test suites were considered. The proposed *RBAC similarity*, the experimental protocol and the analysis of the obtained results are respectively presented in Sections 4.1, 4.2 and 4.3.

1.5 Organization of the Dissertation

The remainder of this Master's Dissertation is organized as follows:

- **Chapter 2**: The background necessary to understand the context of the investigation performed in this master dissertation is presented. It includes concepts of FSM-based testing, the W, HSI and SPY methods, Role Based Access Control, and Test prioritization.
- **Chapter 3**: The proposed experimental protocol for investigating the FSM-based testing methods on RBAC and the results obtained are shown.
- **Chapter 4**: The proposed test prioritization approach, *RBAC similarity*, is introduced in this chapter. The experimental protocol designed to investigate and compare the proposed approach against random prioritization and simple similarity and the results obtained are presented and discussed in this chapter.
- **Chapter 5**: This chapter presents the conclusions obtained with the experimental studies, research limitations, resulting publications and future work.
- **Appendix A**: The protocol and the results obtained from a systematic search designed to collect RBAC policies to the experiments performed are shown in this appendix.
- **Appendix B**: The software named *RBAC-Based Testing (RBAC-BT)* was designed to support FSM-based test prioritization, execution and analysis on RBAC. The main features of this software are presented in this appendix.

CHAPTER

2

BACKGROUND

Access control systems are one of the most critical security mechanism and major concerns for building trustworthy software systems. In this context, the RBAC model has established as one of the most important access control model. Essentially, RBAC uses the concept of grouping privileges and organizational roles to reduce the complexity of administrative security routines and intermediate the assignment of permissions and responsibilities to users. In parallel to these concerns, it is also crucial to guarantee that security mechanisms are correctly designed, implemented and tested. Access control testing is the software testing process for access control systems. Different approaches for testing access control systems have been investigated, although there are still research gaps on the comparison of recent and traditional FSM-based testing methods and test prioritization approaches for RBAC.

This chapter presents the theoretical background considered in this study and is organized as follows: Section 2.1 presents the main concepts of Finite State Machine-Based Testing and three test generation methods: W, HSI and SPY. Section 2.2 introduces the Role Based Access Control model and approaches for modelling and testing RBAC policies using FSM models. At last, Section 2.3 presents and discusses some aspects of test prioritization.

2.1 Finite State Machine Based Testing

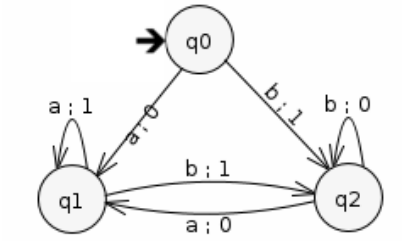
A Finite State Machine (FSM) is an hypothetical machine M composed by states and transitions (GILL, 1962). Formally, an FSM can be defined as a tuple $M =< S, s_0, I, O, D, \delta, \lambda >$ where

- S is a finite set of states,
- $s_0 \in S$ is the initial state,
- I is the finite set of input symbols,

- O is the finite set of output symbols,
- $D \subseteq S \times I$ is the specification domain,
- $\delta : D \to S$ is the transition function, and
- $\lambda : D \to O$ is the output function.

In each given moment, an FSM has one single current state $s_i \in S$ which can change to $s_j \in S$ by applying a defined input symbol $x \in I$ in the transition function, $\delta(s_i, x) = s_j$, and return an output from the output function, $\lambda(s, x) = y$ such that $y \in O$. An input x is defined for s if $(s_i, x) \in D$, which means that in state s there is a *defined transition* consuming input x. A set of tuples (s_i, x, y, s_j) can also be used to represent a defined transition outgoing from s_i to s_j with x and y as input and output symbols, respectively. An FSM is said *complete* if all inputs are defined for all the states, otherwise it is named *partial*. A sequence $\alpha = x_1 x_2 ... x_n \in I$ is an input sequence defined for state $s \in S$, if there exist states $s_1, s_2, ..., s_{n+1}$ such that $s = s_1$ and $\delta(s_i, x_i) = s_{i+1}$, for all $1 \leq i \leq n$. A sequence $\alpha = x_1 x_2 ... x_n \in I$ is a transfer sequence from s to s_{n+1} if $\delta(s, \alpha) = s_{n+1}$. We say that s_{n+1} is reachable from s. When every state is reachable from s_0 the FSM is said *initially connected* and if every state is reachable from all states it is named *strongly connected*. Figure 2 and Table 1 shows the graphical representation and the state transition table of a same FSM where $I = \{a, b\}$, $O = \{0, 1\}$, $S = \{q0, q1, q2\}$ and $s_0 = q0$.

Figure 2 – Example of complete and strong connected FSM in graphical representation



Source: Elaborated by the author.

Table 1 – Example of complete and strong connected FSM represented in state transition table

$s_0 = q0$	λ		δ	
input / state	a	b	a	b
q0	0	1	q1	q2
q1	1	1	q1	q2
q2	0	0	q1	q2

Source: Elaborated by the author.

The symbol $\Omega(s)$ denotes all the input sequences defined for the state s and Ω_M abbreviates $\Omega(s_0)$, refering to all defined input sequences for a given FSM M. An FSM M can have a *reset operation*, denoted by r, which takes to s_0 regardless the current state. The concatenation

of two sequences α and ω is denoted as $\alpha\omega$. A sequence α is a prefix of a sequence β, denoted by $\alpha \leqslant \beta$, if $\beta = \alpha\omega$, for some given sequence ω. An empty sequence is denoted by ε and a sequence α is a proper prefix of a sequence β, denoted by $\alpha < \beta$, if $\beta = \alpha\omega$ for a given $\omega \neq \varepsilon$. The set of prefix sequences of a set T is defined as $pref(T) = \{\alpha \mid \exists\beta \in T \text{ and } \alpha < \beta\}$, if $T = pref(T)$, T is *prefix-closed*. Using the prefix definition and the empty sequence, the concept of transition and output functions can be extended to input sequences. For a state $s_i \in S$, $\delta(s_i, \varepsilon) = s_i$, and $\lambda(s_i, \varepsilon) = \varepsilon$. Given a sequence $\alpha\chi \in \Omega_M$, the output $\lambda(s_0, \alpha\chi)$ is equivalent to $\lambda(s_0, \alpha)\lambda(\delta(s_0, \alpha), \chi)$, and the state reached by $\delta(s_0, \alpha\chi)$ is the same as $\delta(\delta(s_0, \alpha), \chi)$.

A separating sequence for two states s_i and s_j is a sequence γ such that $\gamma \in \Omega(s_i) \cap \Omega(s_j)$ and $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$. The sequence a is a separating sequence for states $q0$ and $q1$ of the FSM in Figure 2. In addition, if γ is able to distinguish every pair of states of a machine, it is a distinguishing sequence, or simply DS. Formally, if $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$ is valid for all pairs of state $s_i, s_j \in S$, then γ is a distinguishing sequence. Considering the FSM presented in Figure 2, the sequence a is a separating sequence for states q_0 and q_1 since $\lambda(q_0, a) = 0$ and $\lambda(q_1, a) = 1$.

Two FSM models $M_S =< S, s_0, I, O, D, \delta, \lambda >$ and $M_I =< S', s'_0, I, O', D', \delta', \lambda' >$ are equivalent ($M_S \equiv M_I$) if for each state of M_S there exists an equivalent state in M_I or, formally, $\forall\, s_i \in S,\ \exists\, s_j \in S' \mid s_i \equiv s_j$. Two states are said equivalent, $s_i \equiv s_j$, if $\forall\, \alpha \in I,\ \lambda(s_i, \alpha) = \lambda'(s_j, \alpha)$. An input sequence $\alpha \in \Omega_M$ starting with a reset symbol r is a *test case* of M. Given two test sequences $\alpha, \beta \in T$, if α is a proper prefix of the test case β, the execution of β implies the execution of α, thus α can be removed from T without altering the test result. A test suite of M consist on a finite set T of test cases of M, such that there are no two sequences $\alpha, \beta \in T$ where $\alpha < \beta$. The number of symbols of a sequence α is represented by $|\alpha|$ and describes the length of the test sequence α. Given a test case α, the cost of executing is calculated as $|\alpha| + 1$, which stands for the length $|\alpha|$ of the test sequence plus one reset operation $(+1)$. The number of test cases of one test suite T is represented by $|T|$ and also describes the number of resets of T.

2.1.1 Mutation Analysis for FSM

The test assessment plays an important role in software testing research and fault injection is one of the most frequently used approaches. These faults can be injected either manually or by automatically generating variants of the SUT, named *mutants* (ANDREWS *et al.*, 2006). Such mutants are generated from the original SUT by performing simple syntactic changes, each containing a different syntactic modification, using *mutation operators*.

In the FSM-based testing domain, given a specification M, the symbol $\Im(M)$ denotes the set of all deterministic FSMs with the same input alphabet of M for which all sequences in Ω_M are defined. Let $m \geq 1$, then $\Im_m(M)$ denotes all FSMs of $\Im(M)$ with at most m states. Given a specification M with n states, a test suite $T \subseteq \Omega_M$ is m-complete if for each $N \in \Im_m$ distinguishable from M, there exists a test $t \in T$ that distinguish M from N. This $\Im(M)$ set is named *fault domain* for M and it can be used to assess the quality of a given test suite. If the

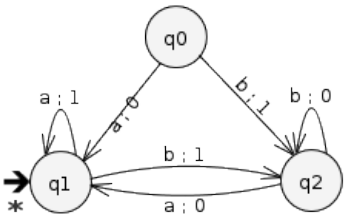
result of running a mutant is different from the result of the original SUT for any test case, the seeded fault denoted by the mutant is detected and the mutant is said *killed*. However, some mutants can be syntactically different but functionally equivalent to the original SUT. These mutants are named *equivalent mutants* (JIA; HARMAN, 2011).

The main outcome of the mutation analysis is the *mutation score*, which indicates the effectiveness of a test suite. Given the test suites T, the mutation score (or effectiveness) can be calculated using the equation $T_{\text{eff}} = \frac{\#km}{(\#tm - \#em)}$. The $\#km$ parameter represents the number of killed mutants, the $\#tm$ defines the total number of generated mutants, and $\#em$ the number of mutants equivalent to the original SUT. Thus, the mutation score consists of the ratio of the number of detected faults over the total number of *non-equivalent* mutants. An *m-complete* test suite has *full fault coverage* for the defined domain and is able to detect all faults in any FSM implementation with at most m states. The process of analyzing when mutants are killed and which test suites trigger such failures is named *mutation analysis*.

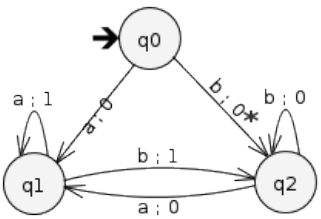
The mutation analysis is frequently used in investigations on FSM-based testing (JIA; HARMAN, 2011; FABBRI *et al.*, 1994). In FSM-based testing, the following mutation operators are often used (CHOW, 1978): *Change Initial State* (CIS), that changes the s_0 of an FSM to s_k, such that $s_0 \neq s_k$; *Change Output* (CO), that modifies the output of a transition (s,x), using a different function $\Lambda(s,x)$ instead of $\lambda(s,x)$; *Change Tail State* (CTS), that modifies the tail state of a transition (s,x), using a different function $\Delta(s,x)$ instead of $\delta(s,x)$; and *Add Extra State* (AES), that inserts a new state such the mutant N is equivalent to M. Figures 3a, 3b, 3c, and 3d respectively show examples of mutants generated from the FSM presented in Figure 2 with the *Change Initial State*, *Change Output*, *Change Tail State*, and *Add Extra State* operators. Changes are marked with an asterisk (*).

Figure 3 – Examples of FSM Mutants

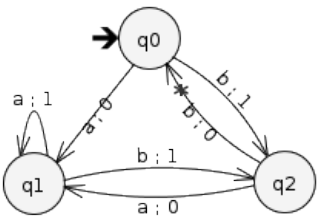
(a) FSM mutant - CIS



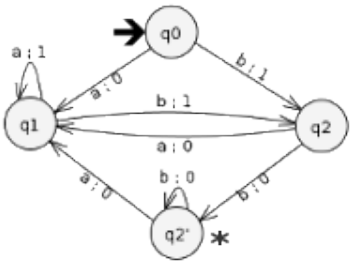
(b) FSM mutant - CO



(c) FSM mutant - CTS



(d) FSM mutant - AES



Source: Elaborated by the author.

2.1.2 FSM-Based Testing Methods

FSM-based testing aims at proving the equivalence, or conformance, between FSM models. In this context, some basic sequences are used to obtain partial information about the test model. The two main sets of basic sequences are the state cover (Q set) and transition cover (P set).

A set of input sequences Q is a *state cover set* of M if for each state $s_i \in S$ there exists a sequence $\alpha \in Q$ such that $\delta(s_0, \alpha) = s_i$ and it includes the empty sequence ε to reach the initial state. A set of inputs P is named *transition cover set* of M if for each transition $(s,x) \in D$ there exist sequences $\alpha, \alpha x \in P$, such that $\delta(s_0, \alpha) = s$, and it includes the empty sequence ε to reach the initial state. The P set can be generated from the *testing tree* of an FSM under test (BROY *et al.*, 2005). The nodes of the testing tree correspond to the states of the FSM, the tree is rooted at the initial state, and each tree edge correspond to one FSM transition that appear exactly one single time. The state cover and transition cover sets of the FSM presented in Figure 2 are respectively $Q = \{\varepsilon, a, b\}$ and $P = \{\varepsilon, a, aa, ba, b, ab, bb\}$.

To identify states and transitions of FSM models, traditional methods, such as W (CHOW, 1978) and HSI (PETRENKO; BOCHMANN, 1995), require some pre-defined sets. These pre-defined sets are the *characterization set* and *separating families*. A characterization set (W set) is a set of defined input sequences containing at least a sequence which distinguishes each pair of states of an FSM. Formally, it means that $\forall s_i, s_j \in S, i \neq j$, $\exists \alpha \in W$ such that $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. A separating family, or harmonized identifiers, is a set of state identifiers H_i for each state $s_i \in S$ that satisfies the condition $\forall s_i, s_j \in S, s_i \neq s_j$ $\exists \beta \in H_i, \gamma \in H_j$ that have a common prefix α such that $\alpha \in \Omega(s_i) \cap \Omega(s_j)$ and $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. The characterization set of the FSM presented in Figure 2 is $W = \{a, b\}$, since a and b inputs are capable of distinguishing every pair of states of the FSM, as shown in Table 2.

Table 2 – Example of characterization set (W set)

state \ input	a	b
q0	0	1
q1	1	1
q2	0	0

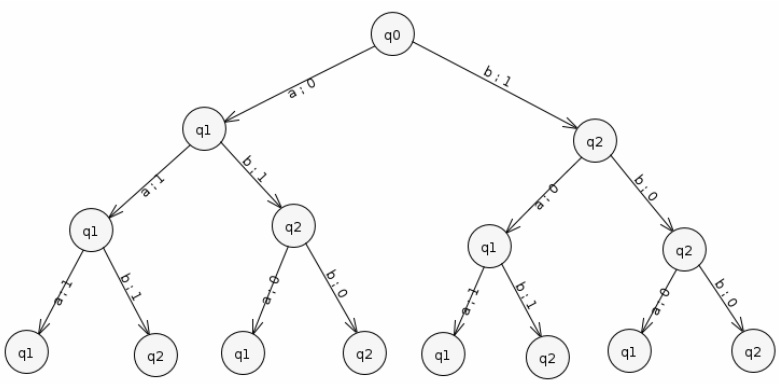
Source: Elaborated by the author.

Recent methods, such as SPY (SIMÃO; PETRENKO; YEVTUSHENKO, 2009), rely on sufficient conditions to support test generation. However, these conditions are not necessary, i.e. if a test suite does not satisfy them, it may still be *m-complete*. The W, HSI and SPY methods are described below.

2.1.2.1 W method

The W method is one of the most classic test generation methods for FSM (CHOW, 1978). It uses the transition cover set (P set) to traverse all the FSM transitions and then it applies the W set to identify the states reached. The W set is concatenated to the leaves of the testing tree, which represents the P set. The W method can also be extended to detect an estimated number of n states in an implementation by using the traversal set $\bigcup_{i=0}^{m-n}(I^i)$, such that $(m-n)$ depicts the number of extra states. The set I^i contains all sequences of length i combining the input symbols of I and the traversal set consists of the union of all sets I^i with sequences of length ranging from 0 to $(m-n)$. The W set is formed by concatenating the P set, the $\bigcup_{i=0}^{m-n}(I^i)$, and the characterization set and is able to detect a total of $(m-n)$ extra states. Assuming the FSM in Figure 2, no extra states ($m=n$) or proper prefixes, the test suite generated by the W method is equals to $T_W = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$, and $|T_W| = 8$. The test tree of T_W is shown in Figure 4.

Figure 4 – Test tree of the test suite generated with W method



Source: Elaborated by the author.

2.1.2.2 HSI method

The Harmonized State Identifiers (HSI) method (PETRENKO; BOCHMANN, 1995) uses state identifiers H_i to distinguish each state $s_i \in S$ of the FSM model. First, the HSI method concatenates the state cover set to the harmonized identifiers set which is, in the worst case, the W set itself. Later, the transition cover set is also concatenated with harmonized identifiers to cover non-traversed transitions. The HSI method can also be used on partial FSM. Assuming the FSM in Figure 2, no extra states or proper prefixes, the test suite generated by the HSI method is equals to $T_{HSI} = \{aaa, aba, abb, baa, bba, bbb\}$, and $|T_{HSI}| = 6$.

2.1.2.3 SPY method

The SPY method (SIMÃO; PETRENKO; YEVTUSHENKO, 2009) is a recent test generation method able to generate m-complete test suites and to reduce test test suite length by

concatenating sequences *on-the-fly*. First, all sequences of the state cover set are concatenated to state identifiers. Later, differently from the existing methods, the traversal set is distributed over the test set obtained from the concatenation of the Q set with the H_i identifiers based on sufficient conditions. Thus, test tree branching can be avoided as much as possible and the test suite length and the number of resets can be reduced.

Experimental studies have indicated that the SPY method can generate test suites on average 40% shorter, and longer test cases compared to traditional methods, such as W and HSI (SIMÃO; PETRENKO; YEVTUSHENKO, 2009). Moreover, SPY method can achieve higher fault detection effectiveness even if the number of extra states is underestimated (ENDO; SIMAO, 2013).

Assuming the FSM in Figure 2, no extra states or proper prefixes, the test suite generated by the SPY method is equals to $T_{SPY} = \{aaaba, abbb, baa, bba\}$, and $|T_{SPY}| = 4$.

2.2 Role Based Access Control

Access Control (AC) is one of the most frequently used approaches to guarantee data confidentiality, integrity and availability (JANG-JACCARD; NEPAL, 2014). Access control mechanisms are used to implement security policies that control users' access to resources and enable access only for authorized personnel based on security models (SAMARATI; VIMERCATI, 2001).

A *security policy* defines the high level rules that must be regulated to control the access to resources. These rules, or simply *policies*, specify the authorizations and access restrictions that AC mechanisms must enforce. A *security model* provides a formal representation of the access control policy and its operation. This formalization allows the proof of security properties provided by access control systems. A *security mechanism* defines the low level functions that support the implementation of control imposed by the policy and formally stated by the security model.

The concepts presented above provide many advantages in the development of access control mechanisms. In particular, the separation between policies and mechanisms introduces an independence between protection requirements and mechanisms enforcing them. Thus, it is possible to independently discuss protection requirements from implementation requirements, compare different access control policies and/or mechanisms enforcing one same policy, and design mechanisms able to enforce multiple kinds of policies.

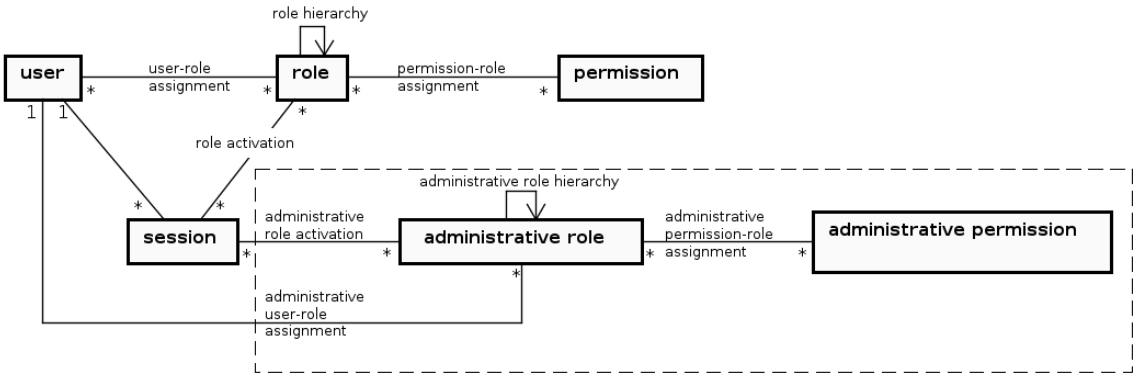
Security policies must capture all the different regulations of an organization in order to be enforced, and guarantee the confidentiality, integrity and availability of data and computational resources. In this context, the Role-Based Access Control (RBAC) security model is considered one of the most important innovations on security management (ANDERSON, 2008). The RBAC

model is a security model that uses the concept of grouping privileges to reduce the complexity of security management tasks (SAMARATI; VIMERCATI, 2001).

In RBAC systems, *roles* consist on organizational figures (e.g., functions or jobs) *assigned* to a set of responsibilities (e.g., *permissions*). In this sense, roles are used to intermediate the assignment and revocation of permissions for *users*. User gain access to the permissions assigned to his/her roles via *sessions* activating roles (*role activation*). *Role hierarchies* can be established in order to enable permission inheritance through relationships between senior and junior roles (e.g., software engineer inherits permissions from programmer). Thus, there is a more natural mapping between security policies and the organizational structure. These elements are the basic concepts of the ANSI RBAC model (ANSI, 2004).

To ease security management, special roles and permissions can be defined for administrative tasks execution. These are named administrative roles and administrative permissions and constitute the Administrative RBAC model (FADHEL; BIANCULLI; BRIAND, 2015). In Figure 5, the ANSI RBAC and, within dashed lines, the Administrative RBAC models are illustrated.

Figure 5 – ANSI RBAC and Administrative RBAC Models



Source: Adapted from Fadhel, Bianculli and Briand (2015).

RBAC policies can also be specified with different types of constraints (FADHEL; BIANCULLI; BRIAND, 2015; MASOOD *et al.*, 2009). Cardinality constraints can be defined to limit the number of user-role assignments and role activations. Mutual exclusion relationships can be specified to avoid the simultaneous assignment or activation of conflicting roles. These constraints are named *Separation of Duty* (SoD) constraints (ANSI, 2004). Essentially, a SoD constraint specifies a set of roles and a number limiting the total number of roles that an user can be assigned from the set of roles. Constraints specified to avoid the assignment of conflicting roles are named Static SoD (SSoD) constraints and the constraints to avoid the activation of conflicting roles are named Dynamic SoD (DSoD) constraints. An example of RBAC policy is shown in Source code 1.

Source code 1: RBAC policy example

```
1 U = {u1,u2}
2 R = {r1}
3 Pr = {pr1,pr2}
4 UR = {(u1,r1)}
5 PR = {(r1,pr1), (r1,pr2)}
6 Su(u1) = Su(u2) = 1
7 Du(u1) = Du(u1) = 1
8 Sr(r1) = 2
9 Dr(r1) = 1
```

The policy in Source code 1 has two users (line 1), one role (line 2), and two permissions (line 3). User $u1$ is assigned to role $r1$ (line 4) that is assigned to the permissions $pr1$ and $pr2$ (line 5). Both users can be assigned to at most one single role (line 6), the same is valid to role activation (line 7). Role $r1$ can be assigned to two users (line 8); however, this role can be activated by one unique user per time (line 9) due to the constraint $Dr(r1) = 1$.

2.2.1 FSM Based Testing of RBAC Systems

According to Masood *et al.* (2009), an RBAC security policy can be defined as a 16-tuple $P = (U, R, Pr, UR, PR, \leq_A, \leq_I, I, S_u, D_u, S_r, D_r, SSoD, DSoD, S_s, D_s)$ where:

- U and R are, respectively, the finite sets of users and roles;
- Pr is the finite set of permissions;
- $UR \subseteq U \times R$ is the set of user-role assignments;
- $PR \subseteq Pr \times R$ is the set of permission-role assignments;
- $\leq_A \subseteq R \times R$ and $\leq_I \subseteq R \times R$ are, respectively, the activation and inheritance role hierarchy relations on roles;
- $I = \{AS, DS, AC, DC, AP, DP\}$ is the finite set of types of requests: user-role assignment (AS), deassignment (DS), activation (AC), and deactivation (DC), and permission-role activation (AP) and deactivation (DP);
- $S_u, D_u : U \rightarrow \mathbb{Z}^+$ are the static and dynamic cardinality constraints on users, respectively;
- $S_r, D_r : R \rightarrow \mathbb{Z}^+$ are the static and dynamic cardinality constraints on roles, respectively;
- $SSoD, DSoD \subseteq 2^R$ are the static and dynamic SoD sets, respectively;
- $S_s : SSoD \rightarrow \mathbb{Z}^+$ specifies the cardinality of SSoD sets; and
- $D_s : DSoD \rightarrow \mathbb{Z}^+$ specifies the cardinality of DSoD sets.

Using this representation, mutation analysis can also be performed in RBAC domain. In RBAC mutation analysis, two types of mutation operators can be used (MASOOD *et al.*, 2009): Mutation operators and Element modification operators. Given a policy P the *mutation operators* are able to generate a set of mutants P' by replacing users, roles and permissions from UR, PR, $\leq_A$, and $\leq_I$, and adding, removing or replacing users from $SSoD$ and $DSoD$ sets; and the *element modification operators* are able to generate a set of mutants P' by incrementing or decrementing cardinality constraints for users (S_u, D_u), roles (S_r, D_r), SSoD (S_s) and DSoD (D_s) sets. These operators can be used to model RBAC faults and can be associated to FSM faults (CHOW, 1978).

2.2.1.1 Modelling RBAC policy as $FSM(P)$

Using the presented definition, Masood *et al.* (2009) propose an approach based on the FSM notation to specify the behavior of RBAC mechanisms enforcing RBAC policies. Given an RBAC policy P, an $FSM(P)$ consists of an FSM describing all access control decisions which a mechanism should enforce given the policy P. Essentially, an $FSM(P)$ consists of a tuple $FSM(P) =< S_P, s_0, I_P, O, D, \delta_P, \lambda_P >$ where

- S_P is the set of states that P allows to reach given its mutable elements;
- $s_0 \in S$ is the initial state where P currently stands given UR and PR;
- I_P is the input domain formed by all combinations of I, U and R elements of P;
- O is the output domain formed by *granted* and *denied* decisions;
- $D \subseteq S_P \times I_P$ is the specification domain;
- $\delta_P : D \rightarrow S_P$ is the state transition function which is total (Complete FSM); and
- $\lambda_P : D \rightarrow O$ is the output function which depends of s_0 and the RBAC constraints.

The states of the $FSM(P)$ are labeled using a sequence of pairs of bits, one for each user-role combination (Table 3). The pair 01 is not used since user-role relationships can only be activated when they are assigned.

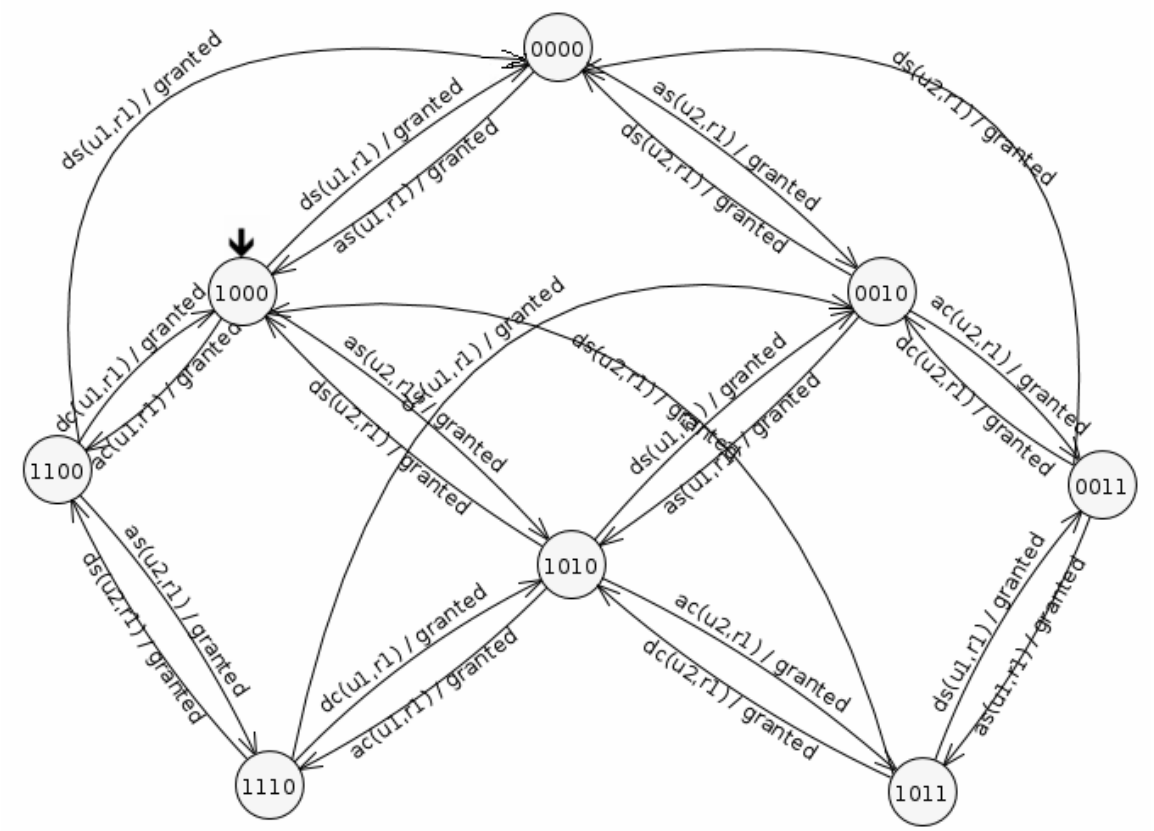
Table 3 – Pattern representing user-role relationships as pairs of bits

Pattern	Role Assigned	Role Activated
00	No	No
10	Yes	No
11	Yes	Yes
01	-	-

Source: Adapted from Masood *et al.* (2009).

This approach has $3^{|U|\times|R|}$ as upper bound limit to the number of states but the real number of reachable states of an $FSM(P)$ depends on the mutable elements of P. The $FSM(P)$ specifying the policy presented in Source code 1 can be seen in Figure 6. Self-loop transitions, corresponding to *denied* requests, are not show to keep the figure uncluttered.

Figure 6 – Complete FSM specifying an RBAC policy



Source: Adapted from Masood *et al.* (2009).

The $FSM(P)$ in Figure 6 has a total of eight states. This number is smaller than the upper bound nine, since the $Dr(r1) = 1$ constraint makes the state 1111 unreachable.

2.2.1.2 Test Generation Methods for $FSM(P)$

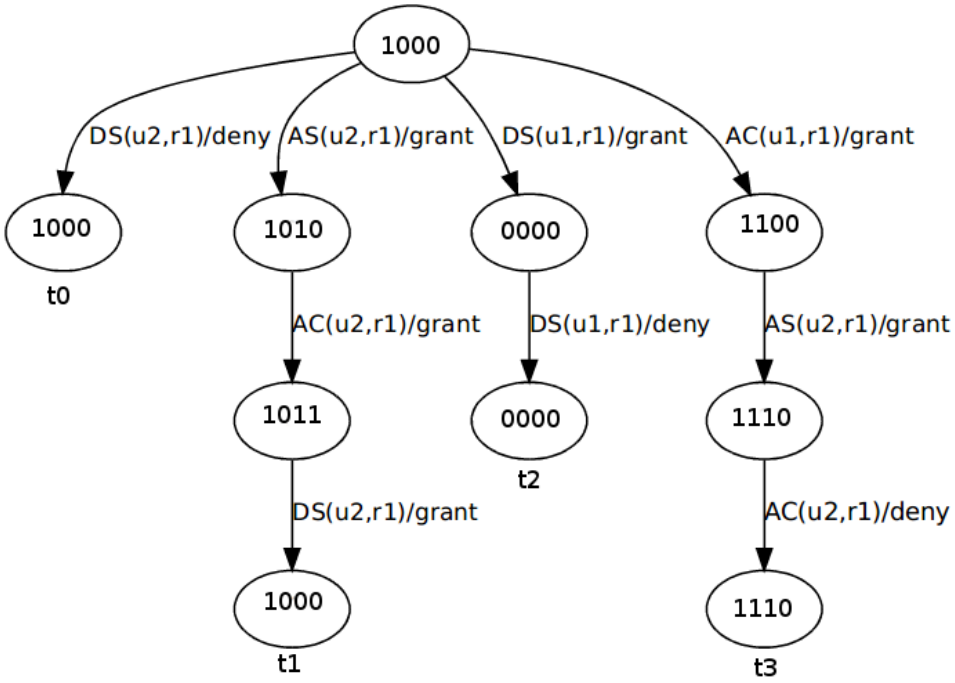
Given an $FSM(P)$, any of the FSM testing method presented in Section 2.1.2 can be applied to test RBAC policies. Figure 7 illustrates the test tree generated from four test cases (Source code 2) applied on the $FSM(P)$ in Figure 6. Each test input is separated by commas.

Source code 2: Test Sequence Example

```
1 DS(u2,r1)/deny // t0
2 AS(u2,r1)/grant , AC(u2,r1)/grant , DS(u2,r1)/grant // t1
3 DS(u1,r1)/grant , DS(u1,r1)/deny  // t2
4 AC(u1,r1)/grant , AS(u2,r1)/grant , AC(u2,r1)/deny // t3
```

Moreover, it is important to highlight that, instead of generating $FSM(P)$ mutants, the RBAC mutation operators are applied on the RBAC policies P. Thus, mutant policies P' are generated and then $FSM(P')$ models can be specified. In this case, a mutant of the RBAC policy presented in Source code 1 where $Dr(r1) = 1$ is incremented to $Dr(r1) = 2$ would generate an $FSM(P)$ similar to the one presented in Figure 6 but with the state 1111 as reachable. The test sequence $t3$ presented in Figure 7 covers the state 1110 and is able to kill a P' mutant where $Dr(r1) = 2$.

Figure 7 – Test Tree of an $FSM(P)$ and four test cases



Source: Elaborated by the author.

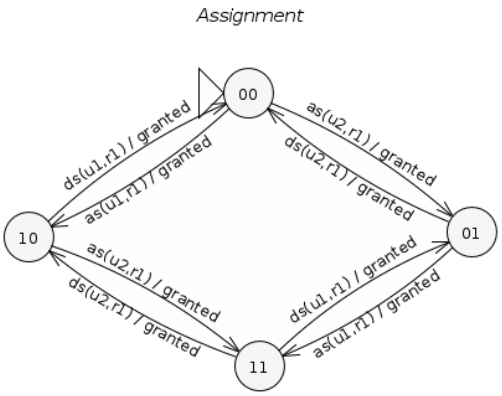
Testing RBAC policies with high number of users and roles can also become very costly since the $FSM(P)$ models are complete. Thus, testing methods also tend to generate very large test suites (MASOOD *et al.*, 2009). In this sense, two alternative approaches were proposed for generating test suites at lower cost, one based on heuristics that generate multiple smaller FSM models instead of a single complete FSM and another based on random test selection.

The heuristic-based approaches allow to reduce test efforts by decreasing the size of the FSM models (Figure 8). The heuristics for testing RBAC propose five different approaches for specifying policies: **(H1)** *separating assignments and activations* – an FSM M_{AS} (Figure 8a) specifies all assignments and for each M_{AS} state an FSM M_{AC_i} (Figure 8b) specifies all possible activations under assumption of each M_{AS} state; **(H2)** *FSM for activation and single test sequence for assignment* – A single FSM M_{AC} (Figure 8b) is specified describing all activation states for a single state q_{max} from M_{AS} which defines one maximum number of assignments; **(H3)** *single test sequence for assignments and activations* – a single test sequence including all user-role pairs for (de)assignments, and (de)activations is built; **(H4)** *FSM for each user* –

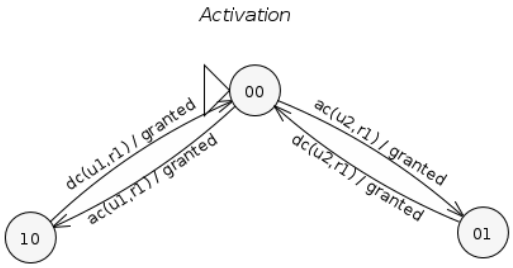
One FSM M_{u_i} is specified for each user u_i of P describing all its possible (de)assignments and (de)activations (Figure 8c); and **(H5)** *FSM for each role* – One FSM M_{r_i} is specified for each role r_i of P describing all its possible (de)assignments and (de)activations.

Figure 8 – $FSM(P)$ models generated using Heuristic-based approaches

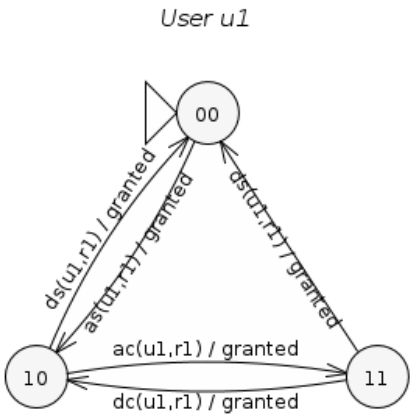
(a) All assignments (M_{AS})



(b) All activations (M_{AC})



(c) All assignments and activations to u_1 (M_{u1})



Source: Adapted from Masood *et al.* (2009).

At last, the random test generation approach, named Constrained Random Test Selection (CRTS), uses a number $k > 0$ to define a fixed value to the test case length and the total number of random inputs selected from I_P.

2.3 Test Case Prioritization

On real-world applications, a large number of test cases tends to be generated and reused as the software evolves. However, due to time and resources constraints, often only a subset of the test cases can be performed. To cope with this issue, researchers have proposed different techniques to improve the cost-effectiveness of test suites. According to Yoo and Harman (2012), these techniques can be classified into three groups: (i) Test Suite Reduction: Techniques for

removing redundant test cases permanently; (ii) Test Case Selection: Techniques for selecting some of the test cases and focus on changed parts of a SUT; and (iii) Test Case Prioritization: Techniques for identifying an efficient ordering of the test cases to maximize certain properties.

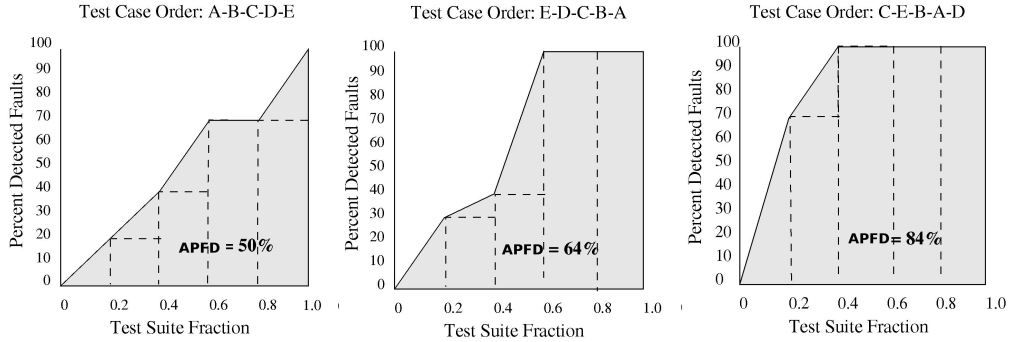
Test suite reduction and test case selection can reduce testing time. However, they also can omit important test cases that can detect certain types of faults and hence they may not work effectively (OURIQUES, 2015). On the other hand, test case prioritization aims at finding an ideal ordering of test cases for testing, so that maximum benefits can be obtained, even if test execution is prematurely halted at some arbitrary point. Formally, the test prioritization problem is defined as follows: *Given a test suite T, the set PT of permutations of T, and a function $f : PT \rightarrow \mathbb{R}$, find a permutation $T' \in PT$ such that $(\forall T'')\ (T'' \in PT)\ (T'' \neq T')\ [f(T') \geq f(T'')]$.* In this definition, PT represents all the possible orderings of the test suite T and the function f quantitatively describes the quality of that ordering via an *award value*. The goal of the prioritization is to maximize (or minimize) this function f which describes a *test criteria* (e.g., test effectiveness). To illustrate test prioritization, consider a hypothetical program with 10 faults, five test cases A, B, C, D, E, and three permutations, as shown in Figure 9.

Figure 9 – Example of test prioritization

(a) Faults detected per test case

test	fault									
	1	2	3	4	5	6	7	8	9	10
A	x				x					
B	x				x	x	x			
C	x	x	x	x	x	x	x			
D					x					
E								x	x	x

(b) Prioritized test suite T1 (c) Prioritized test suite T2 (d) Prioritized test suite T3



Source: Elbaum, Malishevsky and Rothermel (2000).

The faults detected per test case are presented in Figure 9a and the fault detection effectiveness throughout the execution of three different prioritized test suites are presented in Figures 9b, 9c, and 9d. The fault detection effectiveness as a function of the fraction of the test suite $T1 = A, B, C, D, E$ is presented in Figure 9b. After running test case A at first, 20% of the test cases is performed and 20% of the faults are detected. However, 30% of the faults can be detected just by executing test case E at first (Figure 9c) and 70% if the test case C is considered at first (Figure 9d).

After applying some test prioritization approach, the *success* of the ordering can be measured using the Average Percentage Faults Detected (APFD). The APFD is a metric commonly used in test prioritization research (BERTOLINO *et al.*, 2015; ELBAUM; MALISHEVSKY; ROTHERMEL, 2002) and it is defined as follows:

$$\text{APFD} = \frac{\sum_{i=1}^{n-1} F_i}{n \times l} + \frac{1}{2n} \tag{2.1}$$

In Equation 2.1, the n parameter describes the total number of fractions that the test suite is fragmented, the l parameter defines the number of faults under consideration and the F_i value specifies the number of faults detected by a test fragment i. Table 4 shows the APFD value for each one of the prioritized test suites $T1$, $T2$ and $T3$. In this example, the APFD values point that the permutation T3, in Figure 9d, performs better than than $T2$ and $T1$.

Table 4 – APFD value for the test cases example

Prioritized test suite	APFD
T1	0.5
T2	0.64
T3	0.84

Source: Elaborated by the author.

Besides increasing the fault detection effectiveness, many other possible goals can be considered, such as code coverage, reliability.

2.3.1 Similarity based test prioritization

An approach that is currently considered very promising on test prioritization is based on the concept of *test similarity*. Test similarity (or dissimilarity) calculates the resemblance (or difference) between test cases and, based on this information, permutations are performed on test suites. In test similarity, the following hypotheses are assumed (BERTOLINO *et al.*, 2015):

- Similar test cases are redundant in the sense they cover the same set of functionalities of an SUT and have resembling capabilities of detecting faults.
- There is no additional gain to keep similar test cases, since they do not significantly impact on the fault detection.

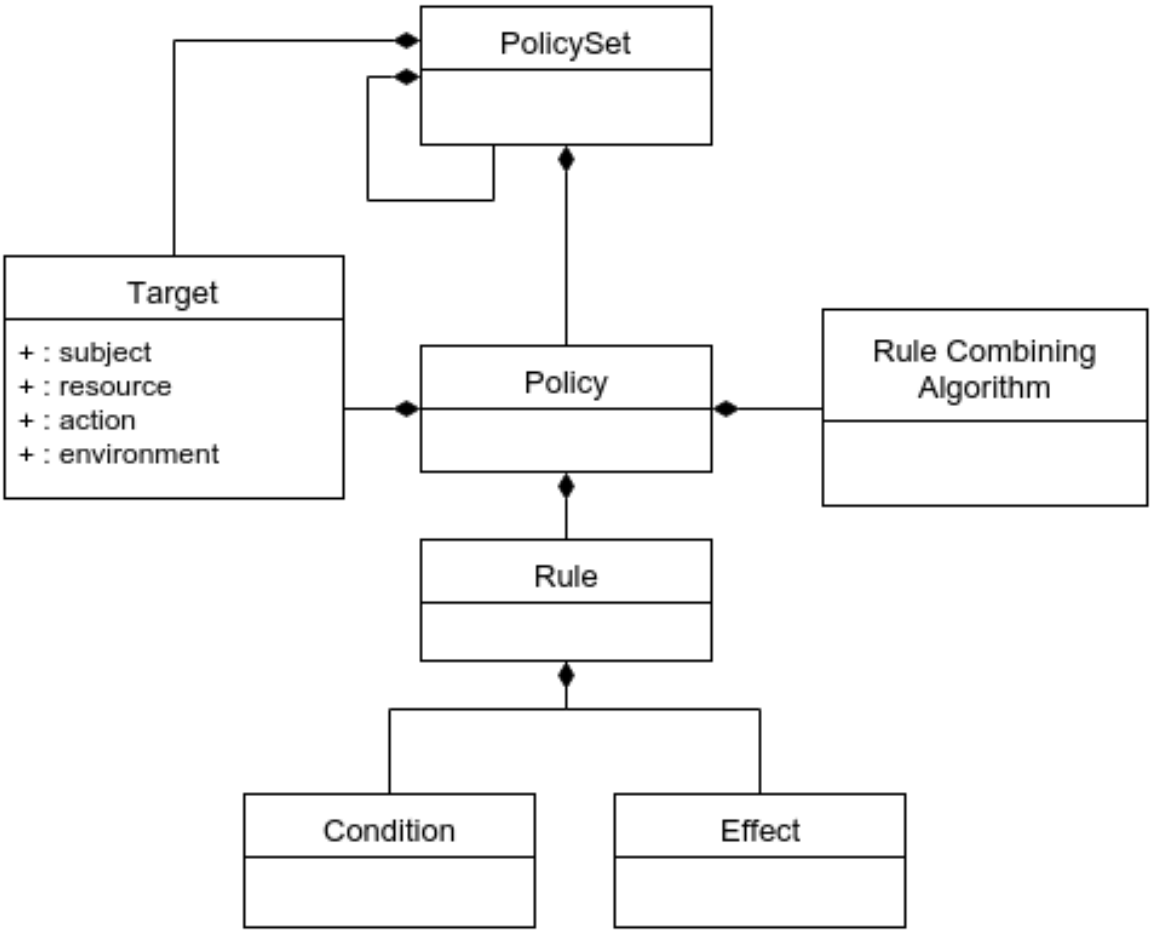
Cartaxo, Machado and Neto (2011) have pointed that similarity-based test prioritization approaches can be more effective than random prioritization when applied to test sequences automatically generated for state models, such as Labelled Transition Systems (LTS) (CARTAXO; MACHADO; NETO, 2011). In this case, the similarity degree (d_{sd}) between two test sequences i and j is calculated based on the number of identical transitions (*nit*) and divided by the average

length of the pair of test cases (Equation 2.2). The average test sequence length is used to avoid small (or large) similarity degrees due to similar short (or long) test sequence lengths. More extensive investigations on test similarity in LTS domain can be found in Coutinho, Cartaxo and Machado (2014).

$$d_{sd}(i,j) = \frac{nit(i,j)}{average(|i|+|j|)} \tag{2.2}$$

Recently, Bertolino *et al.* (2015) have also investigated how similarity-based test prioritization can be improved for testing access control systems based on the eXtensible Access Control Markup Language (XACML). The XACML standard is an XML-based declarative notation for specifying access control policies and evaluating access requests (OASIS, 2013). The standard model of a XACML policy is illustrated in Figure 10.

Figure 10 – XACML policy model



Source: Adapted from OASIS (2013).

Essentially, a XACML policy consists of a hierarchical structure with a *PolicySet* element at the top level, which can contain *PolicySet*, *Policy*, or *Target* elements. A *Policy* element consists of a *Target*, a set of *Rule* elements and a *Rule combining algorithm*. A *Target* element specifies the *subject*, *resource*, *action* and *environment* that a *Policy* can be applied. A *Rule* is composed by a *Condition* that is evaluated when a XACML request is *applicable to* one *Policy*

and an *Effect* that is performed if a *Condition* is evaluated as true. The *Rule combining algorithm* defines what decisions are taken when conflicts happen. A XACML request consists of a 4-tuple $(subject, resource, action, environment)$ which specifies a *subject* requesting to perform an *action* over a *resource* in some given *environment*. If a XACML request satisfies any *Target* of a *Policy*, then the *Rule* elements of the given *Policy* are checked, otherwise it is skipped.

The test prioritization approach proposed by Bertolino *et al.* (2015) is named *XACML similarity* and consist of the following Equation 2.3:

$$d_{xs}(R_i,R_j) = FSM(\begin{cases} 0 & \text{if } d_{ss}(R_i,R_j) = 0 \\ d_{ss}(R_i,R_j) + & \\ AppValue_{(R_i,R_j)} + & \\ PriorityValue_{(R_i,R_j)} & \text{otherwise} \end{cases} \tag{2.3}$$

The XACML similarity considers the (simple) similarity (d_{ss}), the applicability degree ($AppValue$), and a priority value ($PriorityValue$) between two XACML requests R_i and R_j as criteria for prioritizing test suites. The simple similarity can be generally defined as follows:

$$d_{ss}: \begin{array}{l} R \times R \rightarrow \{0,1,2,3,4\} \\ (R_i,R_j) \mapsto d_{ss}(R_i,R_j) \end{array} \tag{2.4}$$

The simple similarity gives a value ranging from 0 to 4 which describes the number of distinct parameters for pairs of access control request. Given two XACML requests $R_1 = (student, book, borrow, null)$ and $R_2 = (professor, book, buy, null)$, since $subject$ and $action$ are the only distinct parameters, the simple similarity is equals to $d_{ss}(R_1,R_2) = 2$. The $AppValue$ consist on a value which describes how much applicable a pair of XACML requests are to one XACML policy. And finally, the *PriorityValue* establishes a priority degree to the pair of XACML requests. These three values are added in one single number which describes not simply how much dissimilar are two XACML requests, but how much relevant they are to one given policy. Experimental investigations have shown simple similarity again outperforming the random prioritization. The joint calculation of the simple similarity degree and the relevance (or applicability) of test cases to XACML policies have also shown improvements significantly superior to the (simple) similarity and comparable to optimal solutions.

2.4 Final Remarks

In this section, we discussed the theoretical foundation considered in this master dissertation. FSM-based testing approaches have been applied for testing RBAC systems but, although very effective, they also tend to generate significantly large test suites. Recent FSM-based test methods also tend to generate better test suites with lower number of resets, longer test cases and shorter test suites for random FSM models. Test prioritization approaches based on test

similarity have been proposed for both contexts, FSM and XACML testing, and experimental investigations have shown that test similarity can outperform random prioritization, but these results can be still improved by measuring the relevance (or applicability degree) of test cases to the SUTs. In this sense, the evidences on FSM and XACML domains motivated the investigation of recent FSM-based testing methods and test prioritization on RBAC. The concepts discussed in this chapter were used to design a novel test prioritization approach and two experiments to compare FSM-based testing and test prioritization approaches for RBAC.

CHAPTER

3

COMPARING FSM-BASED TESTING METHODS ON RBAC

Masood *et al.* (2009) proposed three testing approaches and a fault model for RBAC and performed experiments to compare the cost and effectiveness for each of the investigated approaches. One of the approaches proposes the specification of RBAC policies as complete FSM models. Although test suites generated from complete FSM are able to detect all faults from the RBAC fault domain, they also tend to be very large. Besides, a recent study published by Endo and Simao (2013) pointed out that recent FSM testing methods can reduce the overall test suite length. In this sense, there is no concrete evidence about how recent and traditional FSM-based testing methods behave on FSM models specifying RBAC policies.

An experiment was designed to investigate characteristics and effectiveness of test suites generated by the methods W, HSI and SPY on RBAC. As test characteristics, we considered the *test suite length*, the *number of resets*, and the *average test case length*. As effectiveness, we considered the ratio of killed policy mutants over the total number of non-equivalent policy mutants generated with the RBAC fault model (MASOOD *et al.*, 2009). The following research question was investigated:

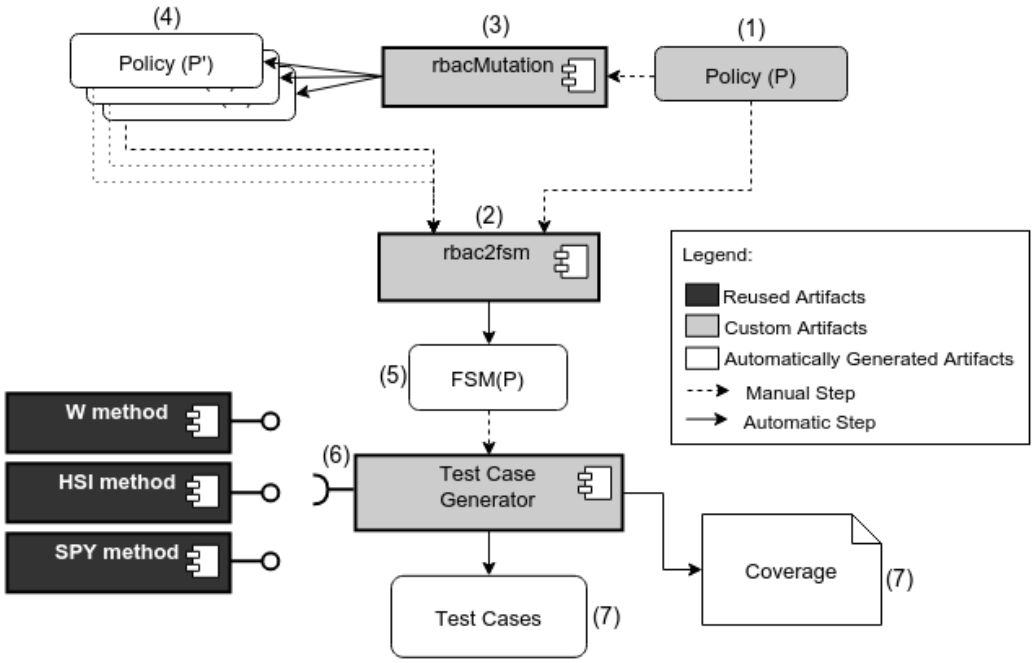
> *Is there any difference on the characteristics of the test suites generated by SPY, HSI, and W methods from RBAC policies specified as FSM models?*

Since Masood *et al.* (2009) claim that 100% of effectiveness can always be achieved as long as all transitions and states of the FSM are covered, no difference was expected on the effectiveness of the generated test suites. This chapter is organized as follows: In Section 3.1, the protocol of the proposed experiment is introduced and the results obtained are presented and analyzed in Section 3.2. Later, discussions about the results are presented in Section 3.3, followed by the threats to validity (Section 3.4) and the final remarks (Section 3.5).

3.1 Experiment Protocol

This investigation was designed as a process composed of seven steps (Figure 11): (1) Selection, analysis, and documentation of RBAC policies, (2) Design and implementation of one component (*rbac2fsm*) to automate the conversion of RBAC policies to FSM models, (3) Design and implementation of one component (*rbacMutation*) to automate the mutation analysis of RBAC policies, (4) Generation of RBAC policies mutants, (5) Generation of FSM models from RBAC policies, (6) Generation of test suites, and (7) Test analysis.

Figure 11 – Comparison of FSM Testing Methods - Schematic overview



Source: Elaborated by the author.

In the first (1) step, scientific papers discussing RBAC testing were systematically searched and analyzed to extract the RBAC policies to be used in this investigation. In the second (2) and third (3) steps, the *rbac2fsm* and *rbacMutation* modules were designed and developed to automate the conversion of RBAC policies to FSMs and the mutation analysis, respectively. Later, these modules were merged into one single software named RBAC-Based Testing (RBAC-BT). A brief description of the RBAC-BT software can be found in Appendix B. In the fourth (4) and fifth (5) steps, the RBAC-BT was respectively used to generate RBAC mutants and FSMs from the selected policies. In case of state explosion, it was decided that constraints should be included to reduce the number of reachable states. In the sixth (6) step, three FSM testing methods (W, HSI and SPY) were used to generate test suites from each $FSM(P)$. The same set of test generation tools used by Endo and Simao (2013) were adopted. A time limit of 24h for processing each $FSM(P)$ was defined, thus any test process with duration above this limit should be canceled due to state explosion. In the last seventh (7) step, the characteristics and effectiveness of the test suites were evaluated using the RBAC-BT tool.

3.2 Analysis of Results

The computational environment used in this experiment was an Intel Core i7-4770 CPU 3.40GHz, 8 Gb RAM, 1Tb of hard disk running Ubuntu 14.04 LTS 64 bits.

3.2.1 Access Control Policies Under Test

Seven RBAC policies were extracted from scientific papers discussing policy testing. Due to the state explosion problem, five policies were adapted with cardinality constraints to reduce the number of reachable states. After refinement, the adapted version of the RBAC policies were converted to FSM models. A summary of the policies is presented in Table 5. The original RBAC policies that had to be adapted are marked with '+', the adapted policies are identified with *v2* and the occurrence of a given type of constraint (e.g., *Su* constraints) or characteristic (e.g., more users than roles (U > R)) are marked with an *x*. The protocol designed to search for papers and RBAC policies, the seven original policies identified, and the five refined versions are presented in Appendix A.

Table 5 – Summary of the characteristics of the RBAC policies

Policy	$\|U\|$	$\|R\|$	$\|I_P\|$	$log_{10}(3^{\|U\|\times\|R\|})$	$\|U\| > \|R\|$	$\|U\| < \|R\|$	$\|U\| = \|R\|$	$\leq_A$	$\leq_I$	S_u	D_u	S_r	D_r	*SSoD*	*DSoD*
01_Masood2010Example1	2	1	8	0.9542	x					x	x	x	x		
02_SeniorTraineeDoctor	2	2	16	1.9084			x			x	x	x	x	x	
03_ExperiencePointsv2	2	4	32	2.7092		x						x		x	x
04_users11roles2_v2	11	2	88	10.4966	x					x		x		x	
05_Masood2009P2v2	2	5	40	3		x				x	x		x	x	x
06_Masood2009P1v2	3	4	48	3.2375		x					x		x	x	
07_ProcureToStockv2	3	5	60	3.5282		x				x		x		x	
03_ExperiencePoints+	3	4	48	3.2375		x				x					x
04_users11roles2+	11	3	132	15.745	x										
05_Masood2009P2+	2	6	48	5.7254		x					x		x	x	x
06_Masood2009P1+	5	4	80	9.5424	x						x		x	x	x
07_ProcureToStock+	4	5	80	9.5424		x									

Source: Research data.

3.2.2 FSM and RBAC Mutants Generation

Initially, all seven RBAC policies were included in the experiment. Thus, we specified them in XML format, mutated using the RBAC-BT tool and attempted to convert them to FSM models. However, due to the state explosion problem, we were able to generate FSM models only from the policies *01_Masood2010Example1* and *02_SeniorTraineeDoctor*. The five remaining policies spent more than 24h being processed by the RBAC-BT tool. Thus, we refined these remaining policies with RBAC constraints. The refined RBAC policies were named as 03_-ExperiencePointsv2, *04_users11roles2_v2*, *05_Masood2009P2v2*, *06_Masood2009P1v2*, and *07_ProcureToStockv2*. Essentially, these **-v2* versions were redesigned by including cardinality constraints, SSoD and DSoD sets and by removing users and roles of the original policies to reduce the number of reachable states. A summary of the FSMs generated from the RBAC policies and the total number of mutants generated are shown in Table 6.

Table 6 – Summary of the FSM and mutants generated from the RBAC policies

Policy name	U	R	$log_{10}(3^{\|U\|\times\|R\|})$	States	Transitions	Mutants
01_Masood2010Example1	2	1	0.9542	8	64	9
02_SeniorTraineeDoctor	2	2	1.9084	21	336	17
03_ExperiencePointsv2	2	4	2.7092	203	6496	11
04_users11roles2_v2	11	2	10.4966	485	42680	28
05_Masood2009P2v2	2	5	3	857	34280	48
06_Masood2009P1v2	3	4	3.2375	1880	90240	40
07_ProcureToStockv2	3	5	3.5282	5859	351540	14

Source: Research data.

As shown in Table 6, the number of states of the generated FSM models ranged from 8 to 5859 and the number of transitions ranged from 64 to 351540. The number of outputs was omitted since this modeling approach assumes only two outputs, $O = \{granted, denied\}$. The number of RBAC mutants generated ranged from 9 to 48. The upper limit of the number of states is shown in logarithmic scale in column $log_{10}(3^{|U|\times|R|})$. Although 04_users11roles2_v2 had the highest upper limit, cardinality constraints were defined to all the 11 users and 2 roles limiting the number of assignments to one. Consequently, it significantly reduced the total number of reachable states. The RBAC-BT software took around one minute to generate the largest FSM model, obtained from the policy 07_ProcureToStockv2. All the artifacts used in this experiment are available on-line[1].

3.2.3 Test Suite Generation

Using the same implementations of Endo and Simao (2013), we executed the W, HSI and SPY methods on each of the seven FSM models generated from the RBAC policies. All methods were executed assuming no extra states in the implementation. In the end of the process, only five FSMs were included since the time for test generation to the policies 06_Masood2009P1v2 and 07_ProcureToStockv2 exceeded the limit of 24 hours. The duration of the test generation ranged from 5 milliseconds, in the fastest case (01_Masood2010Example1), to 21 hours, in the longest case (05_Masood2009P2v2), and the complete process took approximately 63 hours. Table 7 shows the duration of the test generation process on each scenario.

Table 7 – Test generation duration

ACUT	W	HSI	SPY
01_Masood2010Example1	137 ms	187 ms	5 ms
02_SeniorTraineeDoctor	306 ms	54 ms	116 ms
03_ExperiencePointsv2	4.3 min	3.2 min	3.4 min
04_users11roles2_v2	10.5 h	1.97 h	2.58 h
05_Masood2009P2v2	5.03 h	21.36 h	21.92 h

Source: Research data.

[1] RBAC-BT project: <https://github.com/damascenodiego/rbac-bt/>

After test generation, the RBAC-BT software was used to evaluate the W, HSI, and SPY methods. The average test case length, number of resets, and test suite length are presented in the following sections. Three correlation coefficients were calculated using the R statistical software: Pearson Correlation Coefficient (PCC), Spearman Correlation Coefficient (SCC) and Kendall Correlation Coefficient (KCC).

3.2.4 Test Suite Length

Table 8 shows the correlation between the Test Suite Length (TS) and the numbers of states and inputs of the $FSM(P)$ models. On average, there was a very strong positive correlation between test suite length and both the number of inputs, and number of states. Although by using cardinality constraints and SoD sets the number of reachable states can be reduced, such as in $P04$, the number of states and inputs of $FSM(P)$ will be always directly proportional to $|U| \times |R|$ and, consequently, the test suite length will also tend to increase together.

Table 8 – Correlation between test suite length and the numbers of inputs and states of the $FSM(P)$

	Correlation Inputs / TS			Correlation States / TS		
method	PCC	SCC	KCC	PCC	SCC	KCC
W	0.9502413749	1	1	0.7174809988	0.9	0.8
HSI	0.9589815946	1	1	0.6821071317	0.9	0.8
SPY	0.9066627043	1	1	0.8233373615	0.9	0.8

TS: Test Suite Length

Source: Research data.

In Table 9 and Figure 12, the number of resets for each method and policy are respectively presented in tabular and graphical format. The length of the test suites generated by W and HSI for P04 were approximately 5.48 and 2.54 times greater than the SPY test suites. Moreover, a nonlinear behavior can also be identified when the test suite length is analysed as a function of the number of states in $P04$, reason why the correlation values were lower while compared to the inputs. In this sense, SPY generated shorter test suites and the following order can be observed $SPY_{TS} < HSI_{TS} < W_{TS}$.

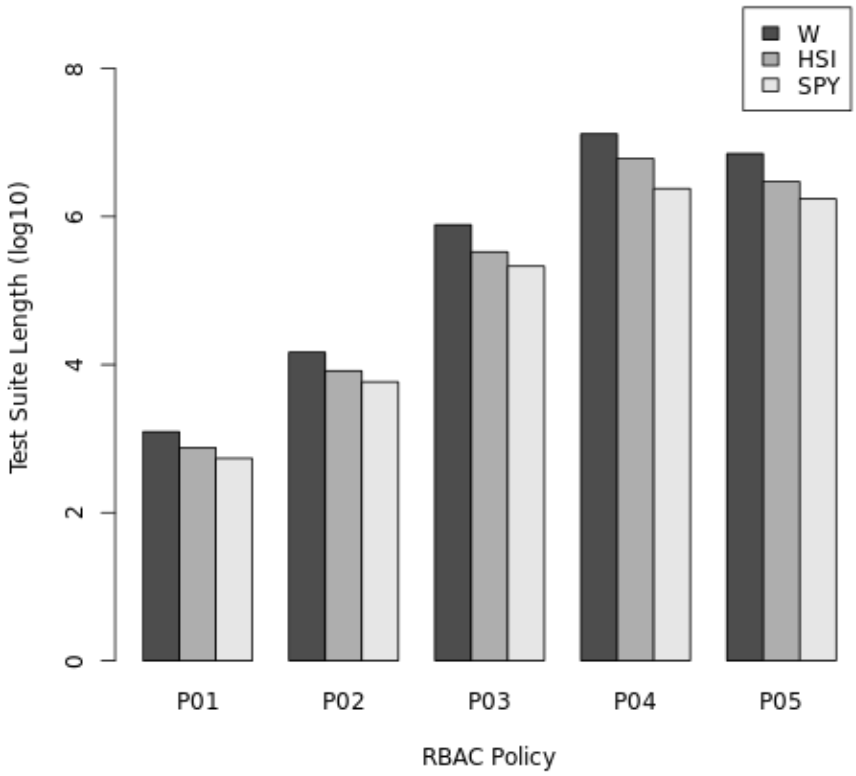
Table 9 – Test suite length, numbers of states and inputs of the $FSM(P)$

Policy	States	Inputs	W - TS	HSI - TS	SPY - TS
01_Masood2010Example1	8	8	1240	753	542
02_SeniorTraineeDoctor	21	16	14704	8238	5841
03_ExperiencePointsv2	203	32	776074	333550	213799
04_users11roles2_v2	485	88	13125662	6085633	2392981
05_Masood2009P2v2	857	40	7086325	2970528	1735818

TS: Test Suite Length

Source: Research data.

Figure 12 – Test suite length for each policy (log_{10})



Source: Research data.

3.2.5 Number of Resets

Table 10 shows the correlation between the Number of Resets (NR) and the numbers of states and inputs of the $FSM(P)$ models. On average, there was a very high positive correlation between the number of resets and both number of inputs and states. The methods W and HSI presented a stronger correlation between the numbers of resets and inputs. The nonlinearity detected on $P04$ test suite length only persisted for the W and HSI testing methods. In SPY method there was a very strong correlation between states and resets. Thus, our results pointed out that the number of resets tends to increase as long as the numbers of states and inputs increase.

Table 10 – Correlation between the number of resets and the numbers of inputs and states of the $FSM(P)$

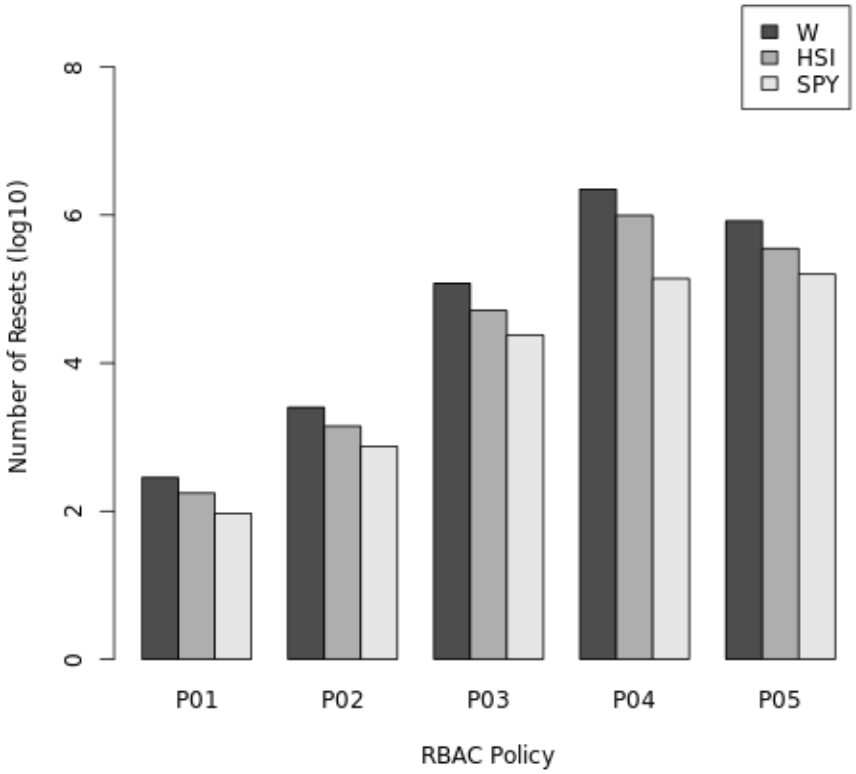
	Correlation Inputs / NR			Correlation States / NR		
method	PCC	SCC	KCC	PCC	SCC	KCC
W	0.9716244957	1	1	0.5953165293	0.9	0.8
HSI	0.9720827105	1	1	0.5809491555	0.9	0.8
SPY	0.7655276825	0.9	0.8	0.9552781412	1	1

NR: Number of Resets

Source: Research data.

Table 11 and Figure 13 show the number of resets of each test suites generated to the five policies in tabular and graphical format. On average, SPY tends to generate test suites with 42.3% the lenght of HSI test suites and 21.5% the length of the W ones. These outcomes corroborate with Simão, Petrenko and Yevtushenko (2009)'s results where SPY test suites presented a number of resets approximately 40% lower than HSI test suites. Since SPY focuses on reducing branches of testing trees, a decrement of resets can be expected.

Figure 13 – Number of resets for each policy (log_{10})



Source: Research data.

Table 11 – Number of resets, numbers of states and inputs of the $FSM(P)$

Policy	States	Inputs	W - NR	HSI - NR	SPY - NR
01_Masood2010Example1	8	8	285	176	93
02_SeniorTraineeDoctor	21	16	2528	1408	751
03_ExperiencePointsv2	203	32	119586	51451	24001
04_users11roles2_v2	485	88	2236388	993492	138766
05_Masood2009P2v2	857	40	835600	353836	159463

NR: Number of Resets

Source: Research data.

In $P04$ scenario, the methods W and HSI generated test suites with the highest number of resets. A reason for that can be the disparity of self-loops in policy $P04$ resulted from the cardinality constraints which significantly limit user-role assignments and, consequently, the

number of reachable states. These outcomes corroborated previous investigations where SPY method generated test suites with lower number of resets when compared to W and HSI (ENDO; SIMAO, 2013), thus Endo and Simao (2013)'s order $SPY_{NR} < HSI_{NR} < W_{NR}$ remained valid.

3.2.6 Average Test Case Length

Table 12 shows the correlation between the Average Test Case Length (L) and the numbers of states and inputs of the $FSM(P)$ models. On average, there was a very strong positive correlation between average test case length and the number of states while the correlation between average test case length and inputs was strong.

Table 12 – Correlation between average test case length and the numbers of inputs and states of the $FSM(P)$

	Correlation Inputs / L			Correlation States / L		
Method	PCC	SCC	KCC	PCC	SCC	KCC
W	0.2679576377	0.7	0.6	0.8501227411	0.9	0.8
HSI	0.3488648294	0.7	0.6	0.8668356622	0.9	0.8
SPY	0.9948791896	1	1	0.5989328388	0.9	0.8

L: Average Test Case Length

Source: Research data.

SPY test suites presented a very strong correlation between inputs and average test case length (over 0.99) and moderate correlation to the number of states. In this sense, it can be expected longer test sequences from SPY method as long as the numbers of inputs and states increase. Thus, even in RBAC domain it can be expected SPY test suites longer than W and HSI as long as the number of users and roles increase. Methods W and HSI presented a weak correlation to the number of inputs. A reason for that can be the low variation of the average test case length values. This behavior can be seen in Table 13 and Figure 14 where the average length does not change significantly for W and HSI. On the other hand, the average length of SPY test cases increases in a higher rate. These outcomes contradict some results of previous investigations where a negative correlation was found between average test case length and number of inputs (ENDO; SIMAO, 2013).

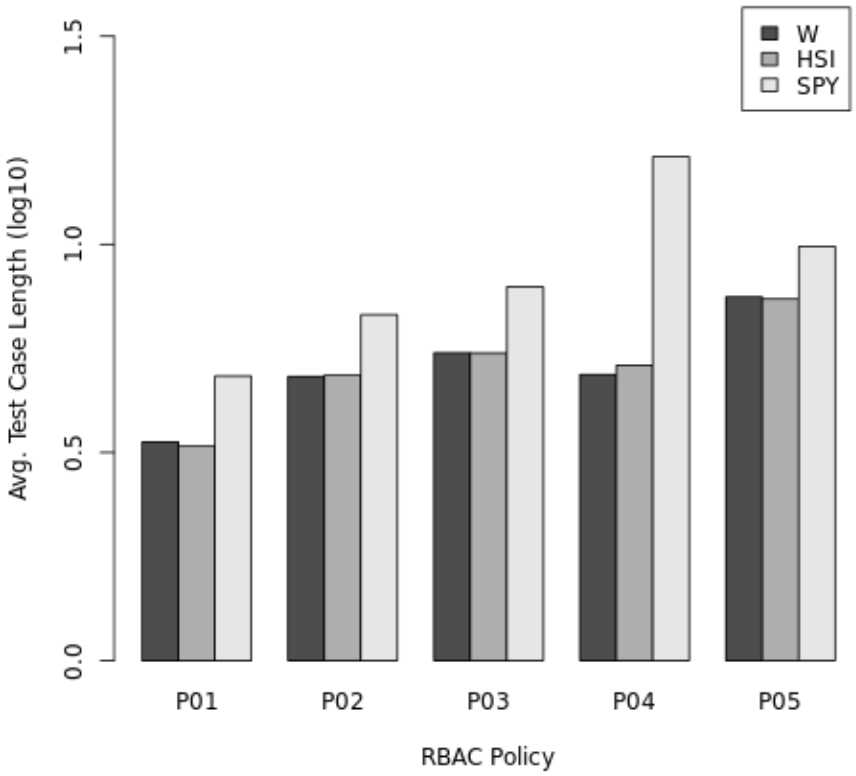
Table 13 – Average test case length, numbers of states and inputs of the $FSM(P)$

Policy	States	Inputs	W - L	HSI - L	SPY - L
01_Masood2010Example1	8	8	3.350877193	3.278409091	4.827956989
02_SeniorTraineeDoctor	21	16	4.816455696	4.850852273	6.777629827
03_ExperiencePointsv2	203	32	5.489672704	5.482867194	7.907920503
04_users11roles2_v2	485	88	4.869134515	5.125497739	16.24472133
05_Masood2009P2v2	857	40	7.480522978	7.395211341	9.885396612

L: Average Test Case Length

Source: Research data.

Figure 14 – Average test case length for each policy (log_{10})



Source: Research data.

The maximum test case length also did not change significantly. Table 14 shows that the maximum test case length of W and HSI methods were similar. On the other hand, SPY method presented a maximum test case length 15 times greater than W and HSI, on average. In $P04$ case, SPY test cases were 43 times longer than the ones generated by traditional methods. Moreover, although SPY method tends to generate longer test cases, their overall cost (test suite length) also tend to decrease. These characteristics are useful on verification and validation processes where test automation is performed. In this sense, the order $W_L < HSI_L < SPY_L$ also remains valid.

Table 14 – Maximum test case length, numbers of states and inputs of the $FSM(P)$

Policy	States	Inputs	W - max(L)	HSI - max(L)	SPY - max(L)
01_Masood2010Example1	8	8	4	4	14
02_SeniorTraineeDoctor	21	16	8	8	32
03_ExperiencePointsv2	203	32	7	7	98
04_users11roles2_v2	485	88	6	6	261
05_Masood2009P2v2	857	40	12	12	106

max(L): Maximum Test Case Length

Source: Research data.

3.2.7 Test Effectiveness

Given the test suites T generated by the W, HSI and SPY methods, the test effectiveness was calculated using the mutation score formula $T_{\text{eff}} = \#km/(\#tm - \#em)$ where $\#km$ represented the number of killed mutants, $\#tm$ the total number of generated mutants, and $\#em$ the number of mutants equivalent to the original policy. Table 15 shows the number of generated and non-equivalent mutants for each policy.

Table 15 – Generated and equivalent RBAC mutants

Policy name	Generated Mutants	Non-Equivalent Mutants
01_Masood2010Example1	9	7
02_SeniorTraineeDoctor	17	9
03_ExperiencePointsv2	11	8
04_users11roles2_v2	28	26
05_Masood2009P2v2	48	44

Source: Research data.

Each test suite generated by the W, HSI and SPY testing methods were applied on the RBAC policies and the number of killed mutants was measured using the RBAC-BT software. At the end, 100% of effectiveness was obtained on all scenarios, corroborating with Masood *et al.* (2009) assertion that as long as one testing method achieves transition and state coverage, the fault detection effectiveness for simple RBAC faults will remain the same. In this sense, any of the testing methods can be used for testing RBAC policies and guarantee complete fault detection in RBAC fault domain. At the same time, the overall cost assigned to test execution can be reduced by adopting recent testing methods, such as SPY, instead of the most traditional ones.

3.3 Discussion

Previous experimental investigations on FSM-based testing methods on RBAC (MASOOD *et al.*, 2009) showed that FSM models specifying RBAC policies can be used to generate complete test suites considering the RBAC fault domain. However, the test suites generated by these FSM testing methods tend to become very large. At the same time, recent studies also showed that modern testing methods can reduce overall test suite length for randomly generated state machines (ENDO; SIMAO, 2013), but no empirical studies evaluate these testing methods in the RBAC domain. Moreover, as the existing evidence on evaluating FSM-based testing methods were generated considering random FSM models, these conclusions can not be necessarily generalized to $FSM(P)$, thus we detected that empirical investigations in this domain were necessary.

In this experiment, our results corroborate Endo and Simao (2013)'s conclusions, but some divergences were also detected. The SPY testing method enabled significant reduction of the overall test costs. On average, the SPY method reduced the test suite length by 61%

compared to HSI, and by 31% compared to W. The length of the SPY test cases also duplicated, on average, compared to W and HSI. Since $FSM(P)$ has both the number of states and inputs directly proportional to $|U| \times |R|$, test dimensions will always tend to increase as long as the number of users and roles increase, even if RBAC constraints are defined.

The effectiveness of the test suites also corroborated with Masood *et al.* (2009) where 100% effectiveness was found. In this sense, as long as an FSM testing method is able to generate test suites and guarantees state and transition coverage, test effectiveness will remain the same. However, the SPY method was able to generate longer test cases and lower number of resets, thus a significant reduction of test effort can be achieved. These outcomes point out that SPY can significantly improve access control testing processes by reducing test cost.

3.4 Threats to Validity

The threats to validity (WOHLIN *et al.*, 2014) identified to this investigation are discussed in this section.

Conclusion Validity: Threats to this validity relate with the ability draw correct conclusions about the relation between the treatment (e.g., test generation methods and policies under test) and the outcomes (e.g., test characteristics and effectiveness). The case studies used five RBAC policies using different types of constraints (See Table 5). None of the five policies used constraints related to the hierarchical RBAC. The experiments selected seven policies but only tested five policies due to the state explosion problem during the FSM generation process, thus some of the policies had to be refined in order to balance the number of states and transitions. Test lengths and number of resets can increase on production systems (e.g., RBAC policies with high numbers of users, roles and permissions) but effectiveness will not change as long as tests cover all transitions and states, although it the costs can significantly increase.

Internal Validity: Threats to internal validity are related with influences that can affect independent variables with respect to causality. They threat conclusions about a possible causal relationship between treatment and outcome, such as effectiveness of FSM testing methods in RBAC domain. In order to mitigate this category of threats, we have designed the RBAC-BT tool based on standards, such as ANSI RBAC (ANSI, 2004), and peer reviewed scientific papers. A verification and validation process was also performed during the development of the RBAC-BT tool. The testing process of the RBAC-BT tool was performed considering some invariants of FSM models describing RBAC policies (e.g., *transitions with deny output symbol are self-loops*, *transitions with grant output symbol are never self-loops*).

External Validity: It concerns with the generalization of the outcomes to other scenarios (e.g., policies or methods). The evaluation was performed using RBAC policies using different types of constraints. Test characteristics may change on different policies, specially policies with greater number of users and roles. Moreover, we did not perform tests on prototypes, production

software or real world policies.

Construct Validity: Construct validity concerns with generalizing outcomes to the concept or theory behind the experiment. Fault detection was measured using mutation analysis (e.g., simple RBAC faults) which is a common assessment approach on software testing investigations (JIA; HARMAN, 2011).

3.5 Final Remarks

This investigation focused exclusively on simple RBAC faults, which refer to functional testing of RBAC mechanisms. Non-functional aspects, such as scalability, and vulnerability analysis were out of the scope of this investigation. Moreover, it is important to highlight that, although SPY method enabled to drastically reduce test dimensions and cost, it can still experience the state explosion problem. In this sense, if we take under consideration test environments with rigorous time and resources constraints it may become hard to execute a whole test suite. Thus, approaches for test reduction become necessary. This scenario motivated the execution of a second experiments to investigate test prioritization approaches for RBAC, discussed in the next chapter. All test artifacts are available on-line[2] and can be used to replicate this experiment.

[2] <https://github.com/damascenodiego/rbac-bt>

CHAPTER

4

INVESTIGATING TEST PRIORITIZATION ON RBAC

Researches have conducted studies on Model Based Testing (MBT) aiming at reducing the costs and time consumption of software testing. Test case prioritization aims at finding an ideal ordering of test cases so that maximum benefits can be obtained, even if test execution is prematurely halted at some arbitrary point (YOO; HARMAN, 2012). Recently, an approach that is becoming very promising on test case prioritization uses the concept of *test similarity*. In test similarity, it is assumed that similar test cases tend to cover the same parts of the SUT and have equivalent fault detection capability. Thus, no additional gain can be expected when similar tests are executed. Cartaxo, Machado and Neto (2011) showed that test similarity approaches can be more effective than random prioritization on MBT. Bertolino *et al.* (2015) also investigated if test similarity can improve test prioritization on XACML testing domain. A test prioritization approach named *XACML similarity* was proposed and compared to other prioritization approaches, including simple similarity and random prioritization. The XACML similarity uses a (simple) similarity degree in conjunction with other values depicting the degree of relevance (or applicability degree) of the test cases to the XACML policies under test. Experimental results pointed out improvements significantly superior to the simple similarity and comparable to optimal solutions. However, there is no test similarity approach comparable to Cartaxo, Machado and Neto (2011) and Bertolino *et al.* (2015) to the RBAC domain. Thus, based on the XACML similarity, a test criteria named *RBAC similarity* was proposed and compared to simple similarity and random prioritization approaches. The RBAC-BT tool was extended to support the RBAC similarity, simple similarity and random test prioritization and the following research question was investigated:

Can the RBAC similarity technique outperform simple similarity and random prioritization in terms of fault detection rate?

This chapter is organized as follows: In Section 4.1, the simple similarity and the proposed RBAC similarity are introduced. In Section 4.2, the proposed experiment is discussed in details. In Section 4.3, the results obtained from the experiments are analyzed, followed by some discussions about the outcomes in Section 4.4. The threats to validity and the final remarks about this chapter are presented in Sections 4.5 and 4.6, respectively.

4.1 Similarity-Based Test Prioritization

In this section, the simple and RBAC similarity are introduced. The prioritization algorithm that performs the test case permutation based on a similarity criterion is also presented. The random prioritization procedure is briefly described as well.

4.1.1 Simple Dissimilarity

The simple dissimilarity (d_{sd}) test prioritization technique uses Cartaxo, Machado and Neto (2011) approach to prioritize test cases for FSM models. Essentially, the execution of test cases starts from the most distinct and less redundant tests to the most similar ones. In this study, the similarity distance proposed by Cartaxo, Machado and Neto (2011) was adapted to calculate the dissimilarity between pairs of test cases, thus instead of counting the number of identical transitions (nit), the number of distinct transitions (*ndt*) is measured.

Given two test cases t_i and t_j, the degree of simple dissimilarity (d_{sd}) is calculated as presented in Equation 4.1.

$$d_{sd}(t_i,t_j) = \frac{ndt(t_i,t_j)}{avg(length(t_i)+length(t_j))} \tag{4.1}$$

The number of distinct transitions (*ndt*) between two test cases (t_i,t_j) is counted and then divided by the average length of the test cases t_i and t_j.

Transitions are considered distinct when there is a mismatch between origin states, input or output symbols, or destination (tail) states. The average test cases length is used to avoid small (or large) similarity degrees due to similar short (or long) test case lengths. In Source code 3 four test cases are presented with the transitions and states covered depicted. The d_{sd} of each pair of test cases is presented in Table 16.

Source code 3: Test Cases Example

```
1 1000 - DS(u2,r1)/deny -> 1000 // t0
2 1000 - AS(u2,r1)/grant -> 1010 - AC(u2,r1)/grant -> 1011 - DS(u2,r1)/
      grant -> 1000 // t1
3 1000 - DS(u1,r1)/grant -> 0000 - DS(u1,r1)/deny  -> 1000 // t2
4 1000 - AC(u1,r1)/grant -> 1100 - AS(u2,r1)/grant -> 1110 - AC(u2,r1)/
      deny  -> 1110 // t3
```

Table 16 – Simple Dissimilarity of each pair of test cases

Pairs (t_i, t_j)	ndt	avg	$d_{sd}(t_i, t_j)$
(t_0, t_1)	4.0	2.0	2.0
(t_0, t_2)	3.0	1.5	2.0
(t_0, t_3)	4.0	2.0	2.0
(t_1, t_2)	5.0	2.5	2.0
(t_1, t_3)	6.0	3.0	2.0
(t_2, t_3)	5.0	2.0	2.0

Source: Elaborated by the author.

4.1.2 RBAC Similarity

The RBAC similarity (d_{rs}) is a criteria proposed to support test prioritization in the RBAC domain. The RBAC similarity considers not just the dissimilarity between test cases, but also how applicable is one test case for a given policy. This criteria was designed based on a test prioritization technique proposed by Bertolino *et al.* (2015) for XACML systems (OASIS, 2013). XACML is an XML-based declarative notation for specifying access control policies and evaluating access requests. Although RBAC policies can be specified using XACML, the current version of the XACML standard does not support yet the specification of SSoD and DSoD constraints (OASIS, 2014). Moreover, the fault detection effectiveness of testing criteria is strongly related to its ability to represent faults (FELDERER *et al.*, 2015). Thus, even though XACML test criteria may be good on detecting XACML faults, they may not be necessarily good on detecting faults related to the RBAC domain. In this sense, our RBAC similarity makes the verification and validation process more suitable to the specificities of the RBAC model.

As presented in Section 2.2.1, an RBAC security policy can be specified as a 16-tuple $P = (U, R, Pr, UR, PR, \leq_A, \leq_I, I, S_u, D_u, S_r, D_r, SSoD, DSoD, S_s, D_s)$. The current state of a policy P is described based on the UR and PR sets which specifies the existing relationships between users (U), roles (R), and permissions (Pr) and the UR, PR, $\leq_A$, $\leq_I$, S_u, D_u, S_r, D_r, $SSoD$, $DSoD$, S_s, and D_s are the mutable elements which support access control decisions. In this sense, given a policy P and an RBAC request $rq(up, r)$, such that $rq \in \{AS, AC, DS, DC, AP, DP\}$, $up \in U \cup P$ and $r \in R$, the following definition is proposed:

Definition 1. A mutable element of a policy P *is applicable to* one RBAC request $rq(up, r)$ if there is a match between the user, role or permission of a request and one mutable element of P.

For example, if a policy contains a static cardinality constraint $S_u(u1) = 1$, this constraint supports access control decisions of any request $rq(up, r)$ where $up = u1$. Therefore, $S_u(u1) = 1$ can be said *applicable to* the request $AS(u1, r2)$. This concept enables to measure how a test case t relates to a given policy by counting the number of requests of t applicable to the mutable elements of P. This definition can be used to describe the static coverage of a policy given one test case, since transitions are not traversed.

However, since RBAC mechanisms are essentially reactive systems, one value for measuring dynamic aspects of policies under test is also necessary. In order to satisfy this requirement, a second definition was proposed:

Definition 2. A mutable element of a policy *P reacts to* a test case when it is applicable to one or more of its requests and enforces access control decisions.

For example, if a policy contains a static cardinality constraint $S_u(u1) = 1$ and there is one role already assigned to $u1$, this constraint will react to any test case attempting to assign a second role not equals to $u1$, and enforce the *deny* response. By counting the number of RBAC elements reacting to a given test cases, it can be measured the dynamic coverage of a policy P, since it considers the transitions traversed.

In this sense, given an RBAC policy P and a test case t, the ratio of requests of a test case applicable to the mutable elements of P and the number of mutable elements reacting on test case execution can be quantified. Here the concept of XACML Applicability Degree proposed by Bertolino *et al.* (2015) was adapted to the RBAC domain. Therefore, RBAC Applicability Degree (*AD*) is defined as an array of four values (Equation 4.2).

$$AD_{P(t)} = \begin{bmatrix} pad_{P(t)} & asad_{P(t)} & acad_{P(t)} & prad_{P(t)} \end{bmatrix} \tag{4.2}$$

The RBAC Applicability Degree ($AD_{P(t)}$) of a test t for a policy P consists of four values:

- **Policy Applicability Degree** ($pad_{P(t)}$): It shows the ratio of requests applicable to all RBAC mutable element over the total of requests;
- **Assignment Applicability Degree** ($asad_{P(t)}$): It shows the number of RBAC mutable elements related to assignment faults reacting to t;
- **Activation Applicability Degree** ($acad_{P(t)}$): It shows the number of RBAC mutable elements related to activation faults reacting to t; and
- **Permission Applicability Degree** ($prad_{P(t)}$): It shows the number of RBAC mutable elements related to permission faults reacting to t.

The $pad_{P(t)}$ measures how applicable one test case is to a given policy considering its mutable elements. It calculates the ratio of requests matching any constraint. The $asad_{P(t)}$ gives a quantitative information about how many RBAC mutable elements related to assignment faults (UR, S_u, S_r, $SSoD$, and S_s) are reacting to t. The $acad_{P(t)}$ gives a quantitative information about how many RBAC mutable elements related to activation faults ($\leq_A$, D_u, D_r, $DSoD$, and D_s) are reacting to t. Finally, the $prad_{P(t)}$ gives a quantitative information about how many RBAC mutable elements related to permission faults (PR, $\leq_I$) react to a test t.

Using the RBAC Applicability Degree of a test case, we calculate the *RBAC Request Applicability Value* ($RA_{P(t)}$). The $RA_{P(t)}$ value summarizes the applicability degree of a test case in a single value calculated as presented in Equation 4.3.

$$RA_{P(t)} = pad_{P(t)} + asad_{P(t)} + acad_{P(t)} + prad_{P(t)} \tag{4.3}$$

The $RA_{P(t)}$ gives a single quantitative attribute which describes how much a test case t relates to one policy P. However, since similarities are calculated for pairs of test cases, it is also required one value for describing the similarity of two test cases, thus the *RBAC Applicability Value* was defined. The RBAC Applicability Value ($AppValue_{P(t_i,t_j)}$) sums the applicability degree of two test cases, as presented in Equation 4.4.

$$AppValue_{P(t_i,t_j)} = RA_{P(t_i)} + RA_{P(t_j)} \tag{4.4}$$

After calculating AppValue, the priority degree of a pair of test cases, or simply the *RBAC Priority Value* ($PriorityValue_{P(t_i,t_j)}$) is calculated. In Equation 4.5, the formula to calculate the *PriorityValue* is presented.

$$PriorityValue_{P(t_i,t_j)} = \begin{cases} \alpha \text{ if } (pad_{P(t_i)} = pad_{P(t_j)} = 1) \\ \beta \text{ if } (pad_{P(t_i)} \; XOR \; pad_{P(t_j)}) \\ \gamma \text{ if } (0 < pad_{P(t_i)}, pad_{P(t_j)} < 1) \\ \delta \text{ otherwise} \end{cases} \tag{4.5}$$

Essentially, the RBAC Priority Value formula gives a constant value based on $pad_{P(t_i)}$ and $pad_{P(t_j)}$ values. These constants are some α, β, γ, or δ defined by the user, such that $\alpha > \beta > \gamma > \delta$. The values 3, 2, 1 and 0 were respectively used in this experiment (BERTOLINO *et al.*, 2015) .

Finally, the *RBAC Similarity* ($d_{rs}(P,t_i,t_j)$ consists on the sum of all the previously calculated values (Equation 4.6). The RBAC Similarity sums the simple dissimilarity, the RBAC applicability value, and the priority value when a pair of test cases is not equal ($d_{sd}(t_i,t_j) \neq 0$).

$$d_{rs}(P,t_i,t_j) = \begin{cases} 0 & \text{if } d_{sd}(t_i,t_j) = 0 \\ d_{sd}(t_i,t_j) + & \\ AppValue_{(P,t_i,t_j)} + & \\ PriorityValue_{(P,t_i,t_j)} & \text{otherwise} \end{cases} \tag{4.6}$$

As an example, the RBAC similarity for the test suite presented in Source code 3 is calculated considering the policy in Source code 1. First, the RBAC applicability degree (RA) must be calculated for each test case. The applicability degree for each test case is presented in Table 17. Afterwards, the simple dissimilarity (d_{sd}), the RBAC application value ($AppValue$), the priority value ($PriorityValue$), and the RBAC similarity (d_{rs}) are calculated for all pairs of test cases. The values for each pair of test cases are presented in Table 18.

Table 17 – RBAC Applicability Degree of each test case

Test (t_i)	$pad_{P(t_i)}$	$asad_{P(t_i)}$	$acad_{P(t_i)}$	$prad_{P(t_i)}$	$RA(t_i)$
t_0	0.77	0.0	0.0	0.0	0.77
t_1	0.77	0.0	0.0	0.0	0.77
t_2	0.77	1.0	0.0	0.0	1.77
t_3	1.0	1.0	1.0	0.0	3.0

Source: Elaborated by the author.

Table 18 – RBAC Similarity of each pair of test cases

Pairs (t_i,t_j)	$d_{sd}(t_i,t_j)$	$AppValue_{(P,t_i,t_j)}$	$PriorityValue_{(P,t_i,t_j)}$	$d_{rs}(P,t_i,t_j)$
(t_0,t_1)	2.0	1.55	1.0	4.55
(t_0,t_2)	2.0	2.55	1.0	5.55
(t_0,t_3)	2.0	3.77	2.0	7.77
(t_1,t_2)	2.0	2.55	1.0	5.55
(t_1,t_3)	2.0	3.77	2.0	7.77
(t_2,t_3)	2.0	4.77	2.0	8.77

Source: Elaborated by the author.

4.1.3 Test Prioritization Algorithm

After calculating the similarity between all pairs of test cases, the sorting algorithm for test prioritization used by Cartaxo, Machado and Neto (2011) was adapted to this experiment. Essentially, the prioritization algorithm receives a list of test cases S as input and returns another list of test cases L prioritized according to the similarity degree of each pair of test case using a similarity function d_x. The similarity function d_x can be any of the previously discussed. The test prioritization procedure is presented in Algorithm 1.

Algorithm 1: Algorithm for Test Prioritization

Input: S = $\{t_1, t_2, \ldots, t_n\}$ `// List of n test cases`
Output: L `// List of n prioritized test cases`

1 $L \leftarrow [\,]$; $S_{copy} \leftarrow clone(S)$
2 Generate the similarity matrix $d_x[n][n-1]$ from S
3 **foreach** $t_a, t_b \in S$ **where** $next_{dec}(d_x[a][b])$
4 **if** $(longest(t_a,t_b) \in S_{copy})$
5 $L.add(longest(t_a,t_b))$; $S_{copy}.remove(longest(t_a,t_b))$;
6 **else** $(shortest(t_a,t_b) \in S_{copy})$
7 $L.add(shortest(t_a,t_b))$; $S_{copy}.remove(shortest(t_a,t_b))$;
8 **end foreach**
9 **return** L

First, an empty list L is instantiated (line 1) and a copy of the S set S_{copy} is created in order to maintain an information about the test cases not included in L. After that, the similarity matrix for all combinations of two of different test cases (line 2) from S is calculated. The

similarity matrix is iterated starting from the most dissimilar pair of test cases (line 3) and then or the longest (line 4) or the shortest (line 6) test case is added to L and removed from S. Using the RBAC similarity on the test suite presented in Source Code 3, the similarity matrix shown in Equation 4.7 can be obtained.

$$\begin{array}{c|cccc} & t_0 & t_1 & t_2 & t_3 \\ t_0 & & 4.55 & 5.55 & 7.77 \\ t_1 & & & 5.55 & 7.77 \\ t_2 & & & & 8.77 \\ t_3 & & & & \end{array} \tag{4.7}$$

In this example, the first most dissimilar pair of test cases is (t_2, t_3) and the test case t_3 is added to L since it is the longest not included. Afterwards, the test case t_0 since it is the longest test case from the next most dissimilar pair (t_0, t_3). The last pair considered is (t_1, t_3) and t_1 is the next to be included as it is the longest non included. The test case t_2 is the last remaining test case to be included in L and it is included when the pair (t_0, t_2) is accessed during the iteration. At the end, the L list will contain all the test cases prioritized according to their pairwise dissimilarity. The Source code 4 shows the test suite from Source code 3 prioritized with the RBAC similarity approach.

Source code 4: Test Cases Example - RBAC similarity

```
1 1000 - AC(u1,r1)/grant -> 1100 - AS(u2,r1)/grant -> 1110 - AC(u2,r1)/
      deny  -> 1110 // t3
2 1000 - DS(u2,r1)/deny -> 1000 // t0
3 1000 - AS(u2,r1)/grant -> 1010 - AC(u2,r1)/grant -> 1011 - DS(u2,r1)/
      grant -> 1000 // t1
4 1000 - DS(u1,r1)/grant -> 0000 - DS(u1,r1)/deny  -> 1000 // t2
```

4.1.4 Random Prioritization

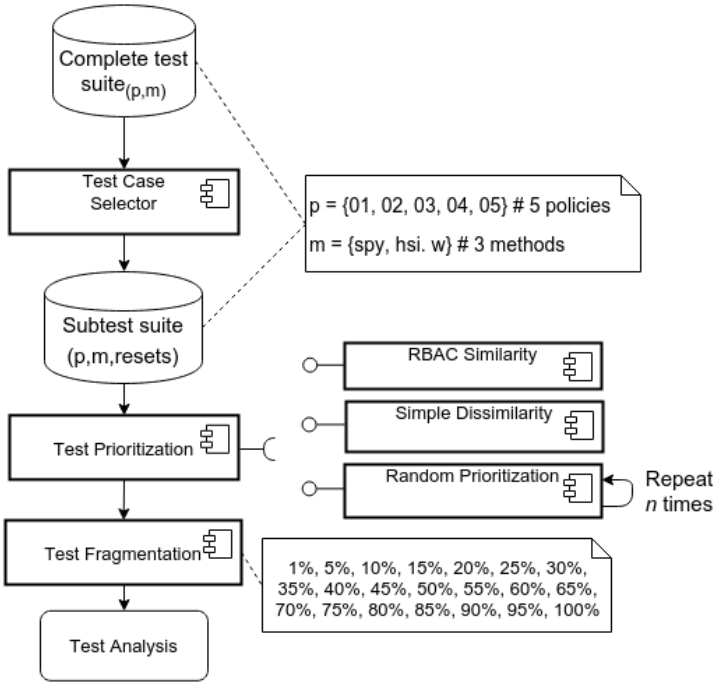
The random prioritization is performed by randomly sorting a list of test cases. If test suites are saved in text files with one test case per line, tools such as *shuf*[1] can perform this task.

4.2 Experiment Protocol

In the second experiment, the three test prioritization approaches previously discussed were evaluated using the fifteen test suites generated in the previous experiment. A schematic overview of the steps of the experiment is presented in Figure 15.

[1] Linux shuf: <http://linux.die.net/man/1/shuf>

Figure 15 – Comparison of Test Prioritization Approaches - Schematic overview



Source: Elaborated by the author.

Initially, the test prioritization approaches were performed on the *complete test suites*, without discarding any test case ($Complete\ test\ suite_{(p,m)} = Subtest\ suite_{(p,m,resets)}$), and the effectiveness of each of the test fragments was measured (Figure 15). The random prioritization was performed 30 times ($n = 30$) on each complete test suite and the average effectiveness of its fragments was calculated. However, since complete FSM-based testing methods tend to generate significantly large test suites (MASOOD *et al.*, 2009), a time limit of 24h for each test prioritization was defined and any prioritization process unsatisfying this constraint would be terminated. Test suites with number of resets greater than those terminated would be discarded as well. In case of any termination, a second experiment was defined using *subtest suites* randomly selected from the W, HSI, and SPY complete test suites. The number of *resets* of the subtest suites would be equals to the number of resets of the largest complete test suites that we were able to prioritize, thus $resets = max(NR_{\text{Complete test suite}_{(p,m)}})$. The whole comparison of the test prioritization approaches was performed 30 times using different random subtest suites and the random prioritization was performed 10 times ($n = 10$). The cumulative effectiveness and the Average Percentage Faults Detected (APFD) were calculated for each prioritized complete/sub test suite in order to perform test analysis. The cumulative effectiveness consists on the effectiveness resulted from the execution of one given fragment of a complete/sub test suite. The cumulative effectiveness was measured considering twenty one fragments containing respectively 1%, 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50%, 55%, 60%, 65%, 70%, 75%, 80%, 85%, 90%, 95%, and 100% of the total number of test cases under consideration. The APFD is a metric commonly used in test prioritization research (BERTOLINO *et al.*, 2015; ELBAUM;

MALISHEVSKY; ROTHERMEL, 2002) and is defined as follows (Equation 4.8):

$$\text{APFD} = \frac{\sum_{i=1}^{n-1} F_i}{n \times l} + \frac{1}{2n} \tag{4.8}$$

In this experiment, the n parameter was considered as the number of fragments ($n = 21$), l as the number of mutants generated from each policy, and F_i as the number of faults detected by a given test fragment i.

4.3 Analysis of Results

In this section, we discuss the results of the experiment performed to compare the RBAC similarity, simple dissimilarity, and random prioritization using the test suites generated by the W, HSI, and SPY methods to the policies P01, P02, P03, P04, and P05. The cumulative effectiveness and the APFD value were calculated for each test suite generated and will be presented below.

4.3.1 Analysis of the Complete Test Suites

First, we attempted to perform the test prioritization considering all 15 complete test suites. However, since the prioritization of the *P03 + SPY* spent more than 24 hours without successful completion, all test suites with total number of resets greater than or equals to $NR_{SPY}(P03)$ were discarded, and only P01 and P02 were considered in this first stage.

4.3.1.1 Cumulative Effectiveness

The cumulative effectiveness of the complete test suites generated by the W, HSI and SPY methods to P01 is presented in Table 19 and Figure 16 and to P02 is presented in Table 20 and Figure 17.

Table 19 – Cumulative effectiveness of the P01 complete test suites

P01 + W	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	0.5	0.75	1	1	1	1	1	1	1	1
	RBAC	1	1	1	1	1	1	1	1	1	1
	Random	0.983	1	1	1	1	1	1	1	1	1
P01 + HSI	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	0.5	0.5	0.5	0.75	1	1	1	1	1	1
	RBAC	1	1	1	1	1	1	1	1	1	1
	Random	0.866	0.975	0.983	1	1	1	1	1	1	1
P01 + SPY	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	0.5	0.5	0.5	1	1	1	1	1	1	1
	RBAC	1	1	1	1	1	1	1	1	1	1
	Random	0.916	0.958	0.991	1	1	1	1	1	1	1

Source: Research data.

Table 20 – Cumulative effectiveness of the P02 complete test suites

P02 + W	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	0.714	0.714	1	1	1	1	1	1	1	1
	RBAC	1	1	1	1	1	1	1	1	1	1
	Random	1	1	1	1	1	1	1	1	1	1
P02 + HSI	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	0.571	0.714	0.714	0.857	0.857	0.857	0.857	1	1	1
	RBAC	0.714	1	1	1	1	1	1	1	1	1
	Random	0.985	1	1	1	1	1	1	1	1	1
P02 + SPY	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	0.714	0.714	0.857	0.857	0.857	0.857	0.857	1	1	1
	RBAC	1	1	1	1	1	1	1	1	1	1
	Random	0.966	1	1	1	1	1	1	1	1	1

Source: Research data.

In five of the six test scenarios the RBAC prioritization outperformed random prioritization and simple similarity. In P01 (01_Masood2010Example1) test scenarios, the RBAC similarity enabled to reach 100% of effectiveness using only 10% of the total number of test cases. In P02 (02_SeniorTraineeDoctor) scenario, respectively 5%, 15% and 10% of the W, HSI and SPY subtest suites became sufficient to reach 100% of effectiveness. The *P02 + HSI* scenario was the only scenario where the RBAC prioritization did not present the best results and the random approach overcomed the RBAC. Moreover, random prioritization performed better than simple similarity in all the scenarios.

4.3.1.2 Average Percentage Faults Detected

In order to evaluate the performance of the investigated test prioritization approaches, we also computed the APFD value on test scenario. The results are shown in Table 21 and the highest APFD values per line are highlighted in **bold**. The APFD values obtained consolidate the observations from Figures 16 and 17. Indeed, the RBAC prioritization outperformed simple similarity and random prioritization in most of the cases, and simple similarity did not improve the cumulative effectiveness compared to random approach.

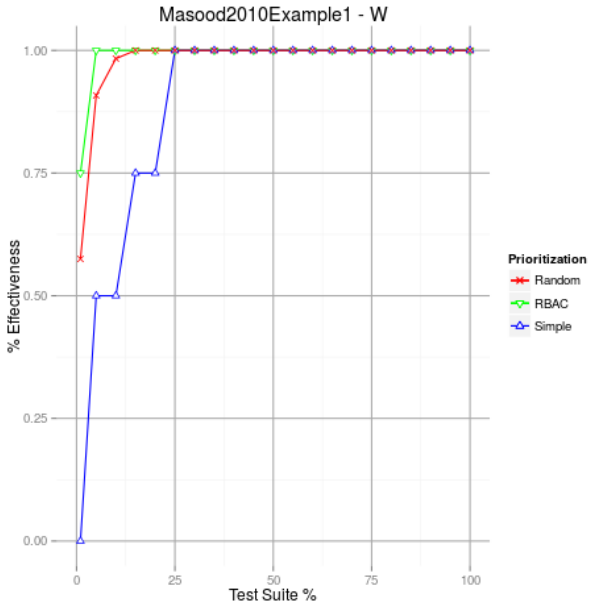
Table 21 – APFD of the complete test suites

Scenario	$APFD_{RBAC}$	$APFD_{Simple}$	$APFD_{Random}$
P01 + W	**0.964**	0.857	0.95
P02 + W	**0.969**	0.874	0.965
P01 + HSI	**0.952**	0.726	0.917
P02 + HSI	0.921	0.778	**0.959**
P01 + SPY	**0.916**	0.785	0.907
P02 + SPY	**0.962**	0.826	0.957

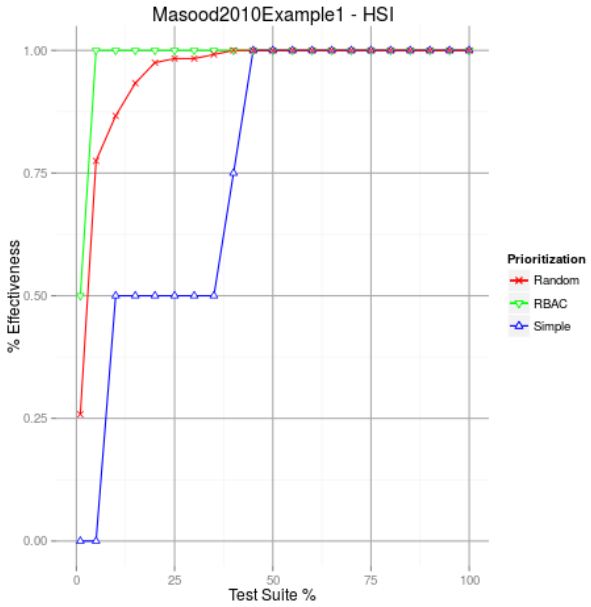
Source: Research data.

Figure 16 – Cumulative effectiveness of the complete test suites for P01

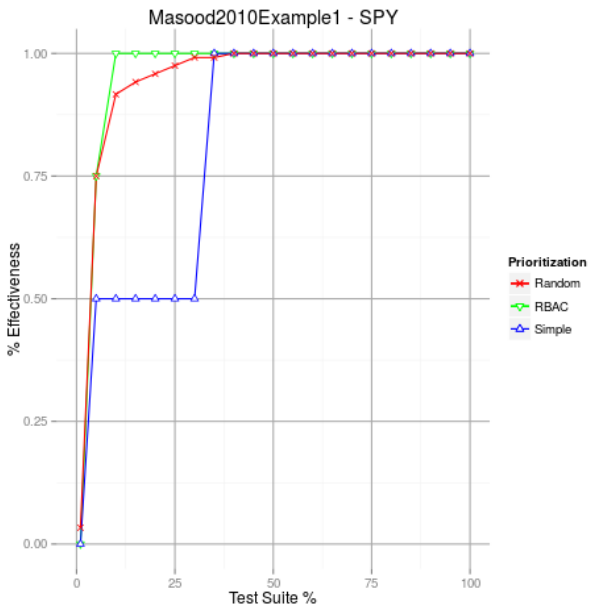
(a) P01 + W



(b) P01 + HSI



(c) P01 + SPY



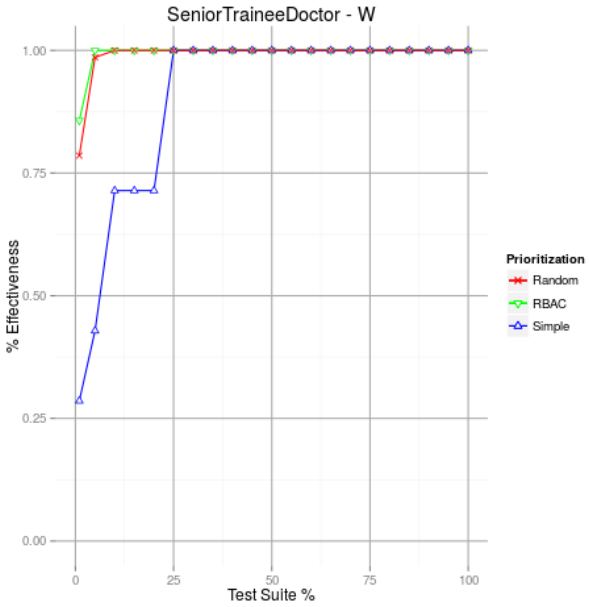
Source: Research data.

4.3.2 Analysis of the Subtest Suites

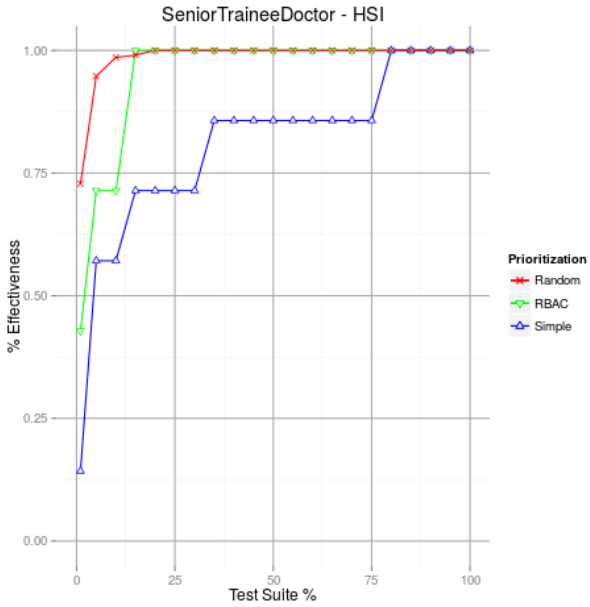
As the prioritization of complete test suites was infeasible on P03, P04 and P05 scenarios, in these scenarios we performed our investigation considering 30 random subtest suites containing 2528 test cases selected from the complete test suites. The number 2528 corresponds to the highest number of resets of the complete test suites we were able to prioritize, *P02 + W* (See Table 11).

Figure 17 – Cumulative effectiveness of the complete test suites for P02

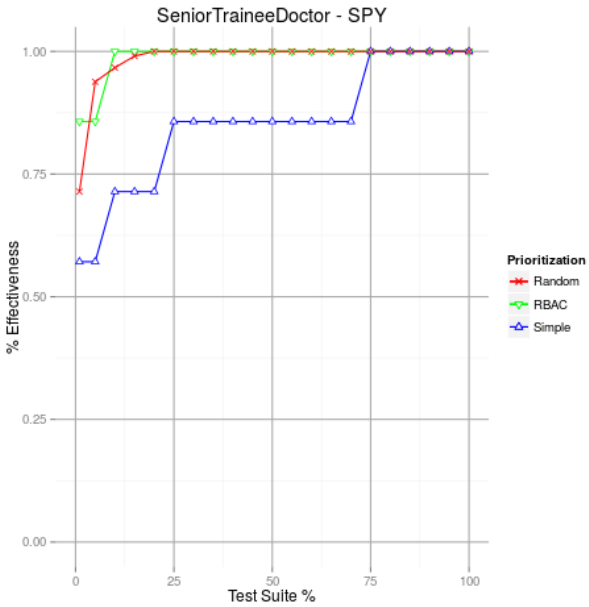
(a) P02 + W



(b) P02 + HSI



(c) P02 + SPY



Source: Research data.

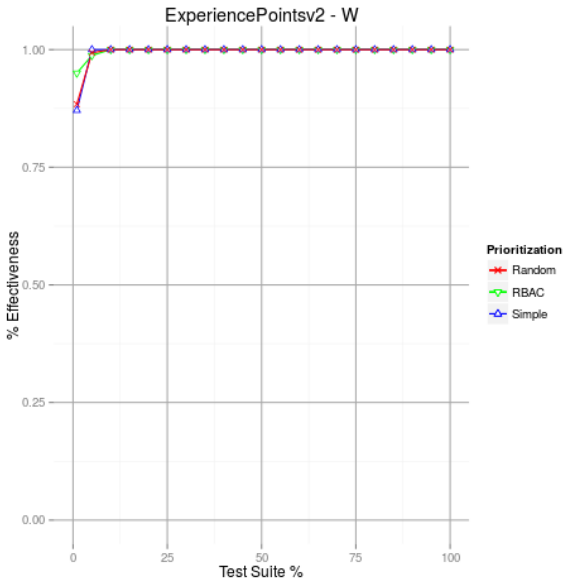
4.3.2.1 *Cumulative Effectiveness*

The cumulative effectiveness of the subtest suites generated to the policies P03, P04, and P05 using W, HSI and SPY methods are respectively presented in Figures 18, 19, and 20, and Tables 22, 23, and 24.

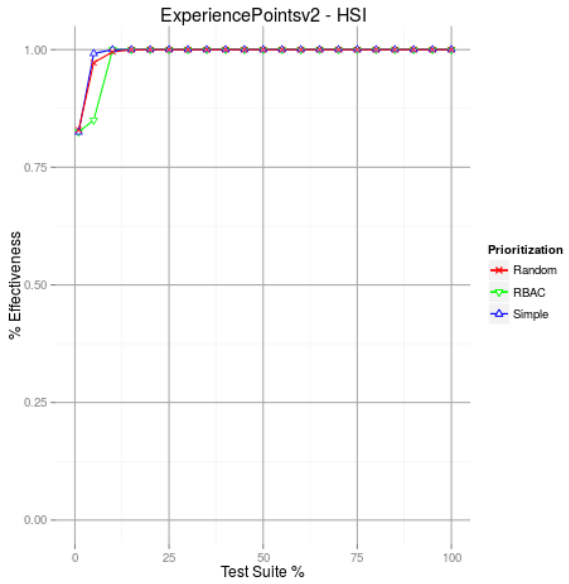
In policy P03 (03_ExperiencePointsv2) scenarios, there was no significant difference among the test prioritization approaches, as presented in Figures 18a, 18b, and 18c and Table 22. After prioritization, around 5 to 10 % of the total number of test cases of the W, HSI and SPY subtest suites became sufficient to reach the highest effectiveness (1.00). In this sense, there was a subset with 125 to 250 test cases capable of detecting all the faults injected in the policy P03 that the prioritization approaches were able put among the first test cases to be performed.

Figure 18 – Cumulative effectiveness of the subtest suites for P03

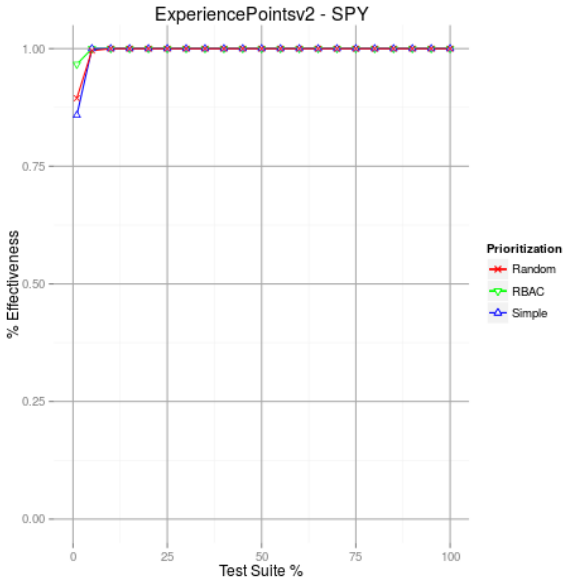
(a) P03 + W



(b) P03 + HSI



(c) P03 + SPY



Source: Research data.

Table 22 – Cumulative effectiveness of the P03 subtest suites

P03 + W	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	1	1	1	1	1	1	1	1	1	1
	RBAC	1	1	1	1	1	1	1	1	1	1
	Random	1	1	1	1	1	1	1	1	1	1
P03 + HSI	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	1	1	1	1	1	1	1	1	1	1
	RBAC	1	1	1	1	1	1	1	1	1	1
	Random	0.995	1	1	1	1	1	1	1	1	1
P03 + SPY	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	1	1	1	1	1	1	1	1	1	1
	RBAC	1	1	1	1	1	1	1	1	1	1
	Random	1	1	1	1	1	1	1	1	1	1

Source: Research data.

In policy P04 (04_users11roles2_v2) scenarios, the differences among the techniques started to become remarkable, as shown in Figures 19a, 19b, and 19c and Table 23. Although there were some oscillations between 25% and 50% in *P04 + W* and *P04 + HSI* scenarios, the subtest suites prioritized using RBAC similarity presented better cumulative effectiveness, reaching the maximum effectiveness (0.845 and 0.839, respectively) by using 65% of the W test cases and 75% of the HSI test cases. At the same time, random and simple prioritization presented quite similar cumulative effectivenesses. In *P04 + SPY* scenario, the RBAC similarity significantly reduced the effort necessary to reach the highest effectiveness. After RBAC prioritization, around 80% of the test cases became able to reach the highest effectiveness *0.846*, although since 30% there was already the guarantee of *0.840* effectiveness.

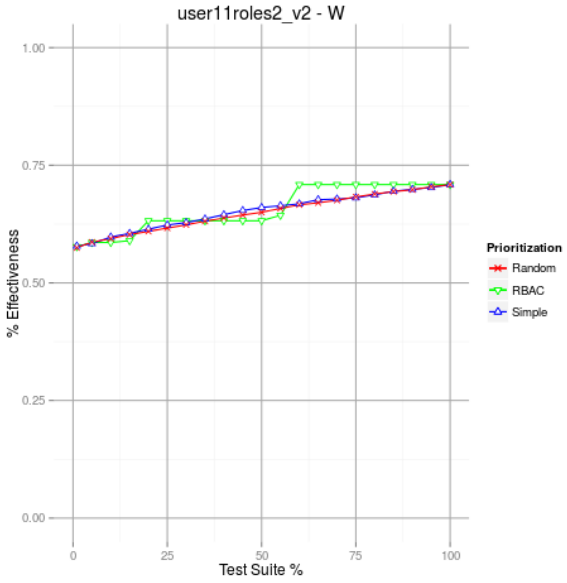
Table 23 – Cumulative effectiveness of the P04 subtest suites

P04 + W	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	0.748	0.772	0.805	0.818	0.827	0.836	0.841	0.841	0.844	0.845
	RBAC	0.8	0.829	0.834	0.837	0.844	0.844	0.845	0.845	0.845	0.845
	Random	0.764	0.801	0.819	0.831	0.837	0.840	0.842	0.843	0.845	0.845
P04 + HSI	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	0.726	0.746	0.771	0.791	0.813	0.826	0.830	0.836	0.838	0.839
	RBAC	0.773	0.801	0.809	0.827	0.832	0.834	0.837	0.839	0.839	0.839
	Random	0.738	0.766	0.787	0.802	0.812	0.820	0.828	0.833	0.836	0.839
P04 + SPY	Percent	10	20	30	40	50	60	70	80	90	100
	Simple	0.75	0.782	0.815	0.826	0.836	0.839	0.842	0.844	0.844	0.846
	RBAC	0.826	0.833	0.840	0.841	0.842	0.844	0.844	0.846	0.846	0.846
	Random	0.770	0.806	0.823	0.833	0.838	0.842	0.843	0.845	0.845	0.846

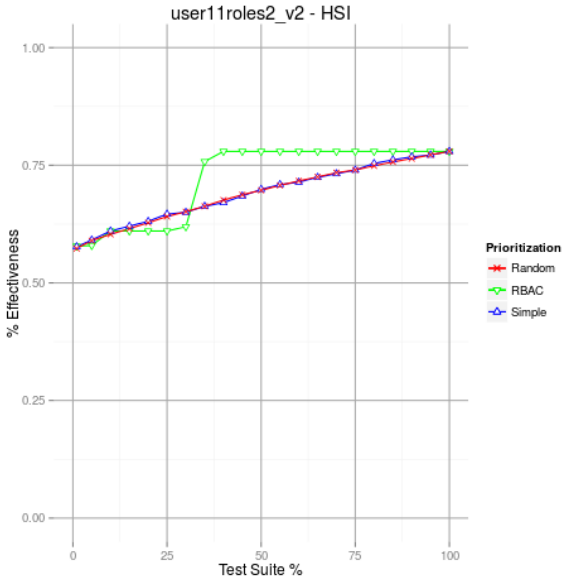
Source: Research data.

Figure 19 – Cumulative effectiveness of the subtest suites for P04

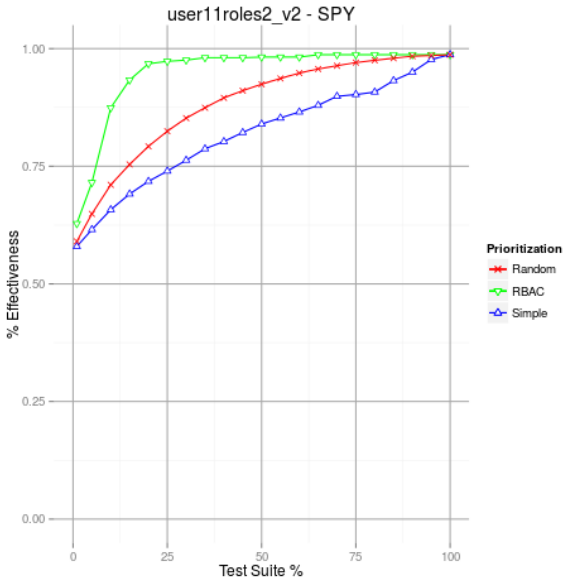
(a) P04 + W



(b) P04 + HSI



(c) P04 + SPY

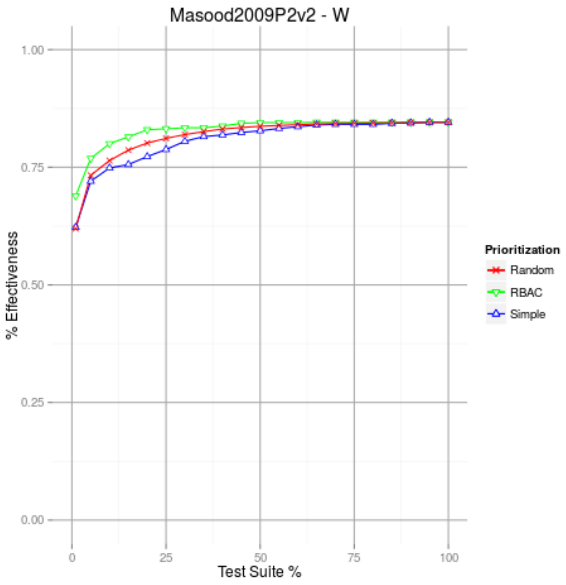


Source: Research data.

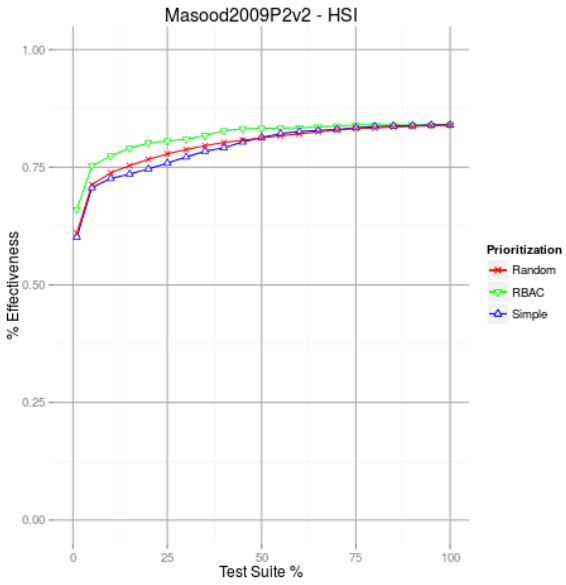
In policy P05 (05_Masood2009P2v2) scenario, the difference among the test prioritization approaches became more visible, as presented in Figures 20a, 20b, and 20c, and Table 24. Again, the simple similarity remained less effective than random prioritization and the RBAC similarity presented cumulative effectiveness higher than both approaches, specially on *P04 + SPY* scenario. Respectively, 80%, 40% and 65% of the W, HSI, and SPY subtest suites prioritized using RBAC became able to reach the highest effectiveness.

Figure 20 – Cumulative effectiveness of the subtest suites for P05

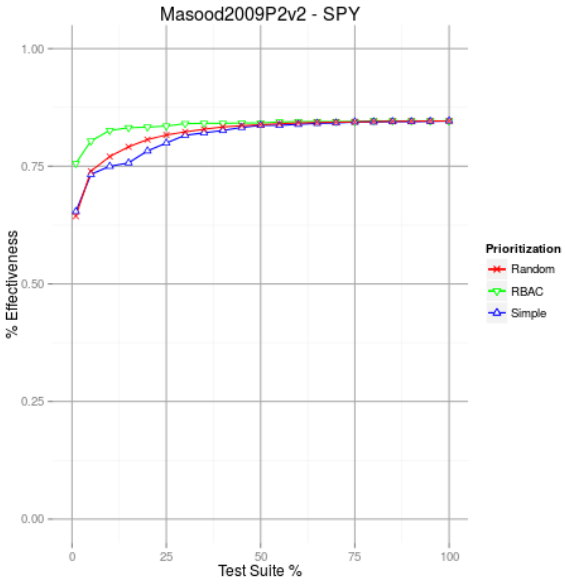
(a) P05 + W



(b) P05 + HSI



(c) P05 + SPY



Source: Research data.

4.3.2.2 *Average Percentage Faults Detected*

We also used the APFD metric to evaluate the performance of the prioritization approaches on the subtest suites. As in this experiment we considered 30 subtest suites, we calculated the APFD for the average effectiveness of the subtest suite. The APFD values computed for each test scenario are shown in Table 25. The highest APFD of each scenario are highlighted in **bold**.

The analysis of the APFD values for the subtest suites showed that the RBAC similarity

Table 24 – Cumulative effectiveness of the P05 subtest suites

P05 + W	Percent	10	20	30	40	50	60	70	80	90
	Simple	0.597	0.614	0.628	0.644	0.660	0.667	0.678	0.687	0.698
	RBAC	0.585	0.632	0.632	0.632	0.632	0.708	0.708	0.708	0.708
	Random	0.593	0.609	0.623	0.637	0.650	0.665	0.675	0.688	0.697
P05 + HSI	Percent	10	20	30	40	50	60	70	80	90
	Simple	0.610	0.630	0.65	0.670	0.698	0.714	0.732	0.753	0.767
	RBAC	0.610	0.610	0.619	0.779	0.779	0.779	0.779	0.779	0.779
	Random	0.603	0.627	0.651	0.676	0.696	0.716	0.734	0.748	0.764
P05 + SPY	Percent	10	20	30	40	50	60	70	80	90
	Simple	0.657	0.717	0.762	0.802	0.839	0.865	0.898	0.907	0.95
	RBAC	0.874	0.967	0.975	0.980	0.982	0.982	0.987	0.987	0.987
	Random	0.710	0.792	0.852	0.895	0.924	0.948	0.963	0.975	0.983

Source: Research data.

Table 25 – APFD of the subtest suites

Scenario	$APFD_{RBAC}$	$APFD_{Simple}$	$APFD_{Random}$
P03 + W	**0.973**	0.97	0.97
P04 + W	**0.646**	0.641	0.638
P05 + W	**0.811**	0.788	0.797
P03 + HSI	0.96	**0.967**	0.966
P04 + HSI	**0.706**	0.676	0.675
P05 + HSI	**0.797**	0.772	0.777
P03 + SPY	**0.974**	0.969	0.97
P04 + SPY	**0.922**	0.794	0.856
P05 + SPY	**0.819**	0.794	0.801

Source: Research data.

performed better than simple similarity and random prioritization in most of the cases. The simple similarity also did not reach an APFD higher than random prioritization. In policy P04 scenario, the RBAC prioritization presented the best results. The policy P04 consist on 11 users and two roles where each of these RBAC elements have constraints limiting the number of assignments and activation, thus its mutants have faults spread around many of the FSM transitions (self-loops). The usage of the RBAC applicability metrics enabled to improve the identification of distinct but relevant test cases.

4.4 Discussion

Previous investigations showed that similarity-based strategies can be helpful when it is necessary to select test cases from an exhaustive test suite automatically generated. In model-based testing context, similarity functions have been used as test prioritization criteria and have enabled to efficiently eliminate redundant test cases, have improved the cumulative effectiveness and the transition coverage of labelled transition systems (CARTAXO; MACHADO; NETO,

2011). At the same time, in the access control testing context, similarity metrics have been combined with applicability metrics for access control testing and empirical results have shown that these approaches can be useful on improving the effective compared to random prioritization and simple similarity based approaches (BERTOLINO *et al.*, 2015). In our investigation, we proposed a test prioritization approach which combines similarity criteria with an applicability metric specific to the RBAC testing domain. Since RBAC faults can be exhibited across many different transitions of $FSM(P)$ (MASOOD; GHAFOOR; MATHUR, 2010b), we expected that the joint usage of the RBAC applicability and simple dissimilarity degrees as test prioritization criteria could improve the effectiveness of test cases better than the isolated application of simple similarity. Our results pointed out to this same direction and the proposed RBAC similarity performed better than simple similarity and random prioritization in most of the cases considering APFD and cumulative effectiveness. In some of the test scenarios, 10% of the total number of test cases of the W, HSI and SPY complete and random sub test suites became as much effective as the whole test suite only by using our RBAC similarity criteria.

4.5 Threats to Validity

The threats to validity (WOHLIN *et al.*, 2014) identified to this investigation are discussed in this section.

Conclusion Validity: Threats to this validity relate with the ability draw correct conclusions about the relation between the treatment (e.g. test prioritization methods and policies under test) and the outcomes (e.g. test effectiveness and APFD). Fifteen complete test suites with different characteristics were prioritized using three different techniques, random prioritization, RBAC similarity and simple similarity. Due to time and resource constraints, only six of the fifteen test suites were completely prioritized. The nine other test suites were used to generate random test suites containing a subset of the total number of test cases. Although, some of the subsets did not reach 100% effectiveness we were still able to compare the prioritization techniques.

Internal Validity: Threats to internal validity are related with influences that can affect independent variables with respect to causality. They threat conclusions about a possible causal relationship between treatment and outcome, such as the effectiveness and APFD of the FSM-based testing methods. The complete test suites were prioritized using the random prioritization 30 times and the random subsets were generated 30 times and randomly prioritized 10 times in order to avoid results obtained by chance.

External Validity: It concerns with the generalization of the outcomes to other scenarios (e.g. policies or methods). To mitigate this threat, the evaluation of the prioritization approaches was performed based on fifteen test suites, thus the confidence that the investigated approaches will behave similarly on other environments can be improved.

Construct Validity: Construct validity concerns with generalizing outcomes to the concept or theory behind the experiment. Fault detection was measured using mutation analysis (e.g. simple RBAC faults) which is a common assessment approach on software testing investigations (JIA; HARMAN, 2011).

4.6 Final Remarks

The RBAC similarity was designed based on the XACML similarity and the test similarity to the LTS domain, proposed by Bertolino *et al.* (2015) and Cartaxo, Machado and Neto (2011). The approach was experimentally evaluated and the obtained results pointed out that RBAC similarity can, in most of the cases, improve the APFD and enabled to reach the maximum effectiveness with less than 50% of the test suites. Prioritizing SPY test suites with RBAC similarity resulted on better APFD values than applying the technique on HSI and W test suites. All test artifacts are available on-line[2] and can be used to replicate this experiment. In the next chapter, we discuss the conclusions obtained from the investigations performed in this project. The contributions, limitations and future work are also presented.

[2] <https://github.com/damascenodiego/rbac-bt>

CHAPTER

5

CONCLUSIONS

Investigations on model based testing of RBAC using FSMs have shown that, although very effective, these approaches can be very costly. Recent studies have also found that recent testing methods can improve the cost-effectiveness of FSM-based testing processes when random FSM models are under test. Studies on test case prioritization to the FSM-based testing domain also pointed out that similarity testing can be more effective than random prioritization approaches. In addition, similar conclusions have been found on investigations about similarity testing for XACML system, and that significant improvements can be obtained if the applicability degree of test cases to XACML policies is taken into consideration. Nevertheless, since the existing investigations have been restricted to both random FSMs and XACML domains, their conclusions cannot be generalized to other domains, such as RBAC testing.

In this sense, this master dissertation comes to fill this gap on model based testing and investigated FSM-based test generation methods and test prioritization techniques for RBAC testing. The contributions of this study are presented in section 5.1. The research limitations are presented in section 5.2. The resulting publications and future work are shown in section 5.3.

5.1 Contributions

This section revisits the main contributions of this Master's Dissertation.

- ***A comparison among FSM-based testing methods on RBAC***: The results obtained indicate that recent FSM testing methods can be more adequate even for testing RBAC systems, and not only for testing random FSM models. The test suites obtained using the SPY method presented less resets, lower total length, and longer test cases. The fault detection effectiveness also persisted on 100% in all cases. In this sense, recent FSM testing methods can replace traditional methods, such as W and HSI, without effectiveness degradation. The proposed experimental protocol, presented in section 3.1, can also be

reused to replicate this investigation with, for example, other test generation methods, RBAC policies or even as a first step for studies on later stages of software testing, such as test selection or prioritization.

- ***An investigation of test prioritization criteria on RBAC***: The RBAC similarity criteria for test prioritization was introduced in section 4.1. This test prioritization approach takes into consideration the relevance of a test case to the SUT, expressed as the RBAC applicability degree, and the dissimilarity of pairs of test cases as criteria for ordering test cases. Since a single RBAC fault can be traced to multiple elements of an FSM, there is no need for complete coverage of the FSM fault domain. The RBAC similarity checks the coverage performed on the RBAC constraints and gives higher priority to the most distinct and applicable pairs of test cases which cover more RBAC elements.

- The repository http://github.com/damascenodiego/rbac-bt contains all artifacts discussed and generated in these experiments. The complete source code of the RBAC-BT tool, all policies and test analysis data can be found in this repository. These artifacts and the experimental protocols can be used to replicate or ratify the experiments performed in this Master's Dissertation.

5.2 Research Limitations

In this investigation, we exclusively used the W, HSI and SPY test generation methods. In this sense, since the performance of test prioritization approaches depends on the characteristics of the test cases and the SUT, our results cannot be generalized to domains where other test generation approaches are taken under consideration. Moreover, none of the RBAC policies found in the systematic review presented role hierarchies, thus this investigation did not cover the hierarchical RBAC model. However, it filled a research gap of the XACML standard. The current version of the XACML standard for specifying RBAC policies only supports role hierarchies but not the static and dynamic SoD RBAC models. The RBAC-BT tool was designed to support only SSoD and DSoD relationships but it did not include activation and inheritance role hierarchies.

5.3 Resulting publications and Future work

The studies published and under development from the data obtained in this Master's investigation and future work are presented in this section.

- DAMASCENO, C. D. N.; DELAMARO, M. E.; SIMÃO, A. d. S. Uma revisão sistemática em teste de segurança baseado em modelos. In: **Anais do Workshop Brasileiro de Testes de Software Automatizados e Sistemático - CBSoft - Congresso Brasileiro de Software: Teoria e Prática**. Porto Alegre: SBC, 2014. p. 31–40.

There are also two other papers under development:

- The first paper will present and discuss the results of the experiment *comparing FSM-based testing methods on RBAC*, presented in Chapter 3. A first version of this paper was submitted to the XXX Brazilian Symposium on Software Engineering (SBES) 2016.
- The second paper will introduce the RBAC similarity approach for test prioritization and the experimental analysis, presented in Chapter 4.

As future work, it is planned the comparison of the test prioritization approaches using different test generation methods, such as pairwise testing. The work used as reference for designing the RBAC similarity used pairwise and it can be used as another baseline to compare the test prioritization methods. The extension of the RBAC-BT tool to support role hierarchies and further replications of the experiment involving more RBAC policies are also under consideration. A possible extension for this work is the evaluation of the effectiveness and APFD values of the FSM-based test generation and test prioritization approaches using different coverage criteria, such as XACML mutation analysis or source code level mutation analysis. The usage of the RBAC similarity concept for test case generation is another possible future work. The RBAC applicability values could be used as test criteria to guide the deterministic generation of test cases or even as constraints on random test generation. The test prioritization could also take advantage of these values to reduce test efforts and support search-based software testing approaches (MCMINN, 2004) as well.

BIBLIOGRAPHY

AMMANN, P.; OFFUTT, J. **Introduction to Software Testing**. [S.l.]: Cambridge University Press, 2008. ISBN 9781139468671. Cited on page 21.

ANDERSON, R. **Security Engineering: A Guide to Building Dependable Distributed Systems**. [S.l.]: Wiley, 2008. Cited on page 33.

ANDREWS, J. H.; BRIAND, L. C.; LABICHE, Y.; NAMIN, A. S. Using mutation analysis for assessing and comparing testing coverage criteria. **IEEE Transactions on Software Engineering**, v. 32, n. 8, p. 608–624, Aug 2006. ISSN 0098-5589. Cited on page 29.

ANSI. **Role Based Access Control**. [S.l.], 2004. ANSI/INCITS 359-2004. Cited 2 times on pages 34 and 55.

BERTOLINO, A.; DAOUDAGH, S.; KATEB, D. E.; HENARD, C.; TRAON, Y. L.; LONETTI, F.; MARCHETTI, E.; MOUELHI, T.; PAPADAKIS, M. Similarity testing for access control. **Information and Software Technology**, v. 58, p. 355 – 372, 2015. ISSN 0950-5849. Available: <http://www.sciencedirect.com/science/article/pii/S0950584914001578>. Cited 13 times on pages 23, 24, 41, 42, 43, 57, 59, 60, 61, 64, 65, 74, and 75.

BROY, M.; JONSSON, B.; KATOEN, J.-P.; LEUCKER, M.; PRETSCHNER, A. **Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN 3540262784. Cited on page 31.

BUGLIESI, M.; CALZAVARA, S.; FOCARDI, R.; SQUARCINA, M. Gran: Model checking grsecurity rbac policies. In: **IEEE 25th Computer Security Foundations Symposium (CSF 2012)**. [S.l.: s.n.], 2012. p. 126–138. ISSN 1940-1434. Cited on page 87.

CARTAXO, E. G.; MACHADO, P. D. L.; NETO, F. G. O. On the use of a similarity function for test case selection in the context of model-based testing. **Software Testing, Verification and Reliability**, John Wiley & Sons, Ltd., v. 21, n. 2, p. 75–100, 2011. ISSN 1099-1689. Available: <http://dx.doi.org/10.1002/stvr.413>. Cited 8 times on pages 23, 24, 41, 57, 58, 62, 74, and 75.

CHOW, T. S. Testing software design modeled by finite-state machines. **IEEE Transactions on Software Engineering**, IEEE Press, Piscataway, NJ, USA, v. 4, n. 3, p. 178–187, May 1978. ISSN 0098-5589. Cited 5 times on pages 23, 30, 31, 32, and 36.

COUTINHO, A. E. V. B.; CARTAXO, E. G.; MACHADO, P. D. d. L. Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing. **Software Quality Journal**, Springer US, p. 1–39, 2014. ISSN 0963-9314. Available: <http://dx.doi.org/10.1007/s11219-014-9265-z>. Cited on page 42.

DAMASCENO, C. D. N.; DELAMARO, M. E.; SIMÃO, A. d. S. Uma revisão sistemática em teste de segurança baseado em modelos. In: **Anais do Workshop Brasileiro de Testes de Software Automatizados e Sistemático - CBSoft - Congresso Brasileiro de Software: Teoria e Prática**. Porto Alegre: SBC, 2014. p. 31–40. Cited 2 times on pages 23 and 86.

DURY, A.; BORODAY, S.; PETRENKO, A.; LOTZ, V. Formal verification of business workflows and role based access control systems. In: **The International Conference on Emerging Security Information, Systems, and Technologies (SecureWare 2007)**. [S.l.: s.n.], 2007. p. 201–210. Cited on page 87.

ELBAUM, S.; MALISHEVSKY, A. G.; ROTHERMEL, G. Prioritizing test cases for regression testing. **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 25, n. 5, p. 102–112, Aug. 2000. ISSN 0163-5948. Available: <http://doi.acm.org/10.1145/347636.348910>. Cited on page 40.

____. Test case prioritization: A family of empirical studies. **IEEE Transactions on Software Engineering**, IEEE Press, Piscataway, NJ, USA, v. 28, n. 2, p. 159–182, Feb. 2002. ISSN 0098-5589. Available: <http://dx.doi.org/10.1109/32.988497>. Cited 3 times on pages 41, 64, and 65.

ENDO, A. T.; SIMAO, A. Evaluating test suite characteristics, cost, and effectiveness of fsm-based testing methods. **Information and Software Technology**, v. 55, n. 6, p. 1045 – 1062, 2013. ISSN 0950-5849. Cited 8 times on pages 23, 24, 33, 45, 46, 48, 52, and 54.

FABBRI, S. C. P. F.; DELAMARO, M. E.; MALDONADO, J. C.; MASIERO, P. C. Mutation analysis testing for finite state machines. In: **Software Reliability Engineering, 1994. Proceedings., 5th International Symposium on**. [S.l.: s.n.], 1994. p. 220–229. Cited on page 30.

FADHEL, A. B.; BIANCULLI, D.; BRIAND, L. A comprehensive modeling framework for role-based access control policies. **J. Syst. Softw.**, Elsevier Science Inc., New York, NY, USA, v. 107, n. C, p. 110–126, Sep. 2015. ISSN 0164-1212. Available: <http://dx.doi.org/10.1016/j.jss.2015.05.015>. Cited on page 34.

FELDERER, M.; ZECH, P.; BREU, R.; BÜCHLER, M.; PRETSCHNER, A. Model-based security testing: a taxonomy and systematic classification. **Software Testing, Verification and Reliability**, p. n/a–n/a, 2015. ISSN 1099-1689. Available: <http://dx.doi.org/10.1002/stvr.1580>. Cited 2 times on pages 23 and 59.

FERRAIOLO, D. F.; KUHN, R. D.; CHANDRAMOULI, R. **Role-Based Access Control, Second Edition**. Norwood, MA, USA: Artech House, Inc., 2007. ISBN 1596931132. Cited on page 21.

GEEPALLA, E.; BORDBAR, B.; OKANO, K. Verification of spatio-temporal role based access control using timed automata. In: **IEEE 3rd International Conference on Networked Embedded Systems for Every Application (NESEA 2012)**. [S.l.: s.n.], 2012. p. 1–6. Cited on page 87.

GILL, A. **Introduction to the Theory of Finite State Machines**. New York: McGraw-Hill, 1962. Cited on page 27.

IEEE. Ieee standard glossary of software engineering terminology. **IEEE Std 610.12-1990**, p. 1–84, Dec 1990. Cited on page 21.

JANG-JACCARD, J.; NEPAL, S. A survey of emerging threats in cybersecurity. **Journal of Computer and System Sciences**, v. 80, n. 5, p. 973 – 993, 2014. ISSN 0022-0000. Special Issue on Dependable and Secure Computing The 9th IEEE International Conference on Dependable, Autonomic and Secure Computing. Cited 2 times on pages 21 and 33.

JIA, Y.; HARMAN, M. An analysis and survey of the development of mutation testing. **Software Engineering, IEEE Transactions on**, v. 37, n. 5, p. 649–678, Sept 2011. ISSN 0098-5589. Cited 4 times on pages 23, 30, 56, and 75.

MASOOD, A.; BHATTI, R.; GHAFOOR, A.; MATHUR, A. P. Scalable and effective test generation for role-based access control systems. **IEEE Transactions on Software Engineering**, IEEE Press, Piscataway, NJ, USA, v. 35, n. 5, p. 654–668, Sep. 2009. ISSN 0098-5589. Cited 15 times on pages 23, 24, 34, 35, 36, 37, 38, 39, 45, 54, 55, 64, 86, 87, and 95.

MASOOD, A.; GHAFOOR, A.; MATHUR, A. Conformance testing of temporal role-based access control systems. **IEEE Transactions on Dependable and Secure Computing**, v. 7, n. 2, p. 144–158, April 2010. ISSN 1545-5971. Cited on page 87.

____. Fault coverage of constrained random test selection for access control: A formal analysis. **Journal of Systems and Software**, v. 83, n. 12, p. 2607 – 2617, 2010. ISSN 0164-1212. {TAIC} {PART} 2009 - Testing: Academic & Industrial Conference - Practice And Research Techniques. Cited 2 times on pages 74 and 87.

MCMINN, P. Search-based software test data generation: A survey: Research articles. **Softw. Test. Verif. Reliab.**, John Wiley and Sons Ltd., Chichester, UK, v. 14, n. 2, p. 105–156, Jun. 2004. ISSN 0960-0833. Available: <http://dx.doi.org/10.1002/stvr.v14:2>. Cited on page 79.

MONDAL, S.; SURAL, S. Security analysis of temporal-rbac using timed automata. In: **Fourth International Conference on Information Assurance and Security (ISIAS 2008)**. [S.l.: s.n.], 2008. p. 37–40. Cited on page 87.

MONDAL, S.; SURAL, S.; ATLURI, V. Towards formal security analysis of gtrbac using timed automata. In: **Proceedings of the 14th ACM Symposium on Access Control Models and Technologies**. New York, NY, USA: ACM, 2009. (SACMAT '09), p. 33–42. Cited on page 87.

____. Security analysis of gtrbac and its variants using model checking. **Computers & Security**, v. 30, n. 2-3, p. 128 – 147, 2011. ISSN 0167-4048. Special Issue on Access Control Methods and Technologies. Cited on page 87.

MOUELHI, T.; KATEB, D. E.; TRAON, Y. L. Chapter five - inroads in testing access control. In: MEMON, A. (Ed.). Elsevier, 2015, (Advances in Computers, v. 99). p. 195 – 222. Available: <http://www.sciencedirect.com/science/article/pii/S0065245815000327>. Cited on page 21.

OASIS. **eXtensible Access Control Markup Language (XACML) Version 3.0**. [S.l.], 2013. Available: <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf>. Cited 3 times on pages 23, 42, and 59.

____. **XACML v3.0 Core and Hierarchical Role Based Access Control (RBAC) Profile Version 1.0**. [S.l.], 2014. Available: <http://docs.oasis-open.org/xacml/3.0/rbac/v1.0/cs02/xacml-3.0-rbac-v1.0-cs02.pdf>. Cited on page 59.

OURIQUES, J. a. F. S. Strategies for prioritizing test cases generated through model-based testing approaches. In: **Proceedings of the 37th International Conference on Software Engineering - Volume 2**. Piscataway, NJ, USA: IEEE Press, 2015. (ICSE '15), p. 879–882. Available: <http://dl.acm.org/citation.cfm?id=2819009.2819204>. Cited on page 40.

PETRENKO, A.; BOCHMANN, G. V. Selecting test sequences for partially-specified nondeterministic finite state machines. In: LUO, G. (Ed.). **7th IFIP WG 6.1 International Workshop on Protocol Test Systems**. London, UK, UK: Chapman and Hall, Ltd., 1995. (IWPTS '94), p. 95–110. ISBN 0-412-71160-5. Available: <http://dl.acm.org/citation.cfm?id=236187.233118>. Cited 3 times on pages 23, 31, and 32.

SAMARATI, P.; VIMERCATI, S. Access control: Policies, models, and mechanisms. In: FOCARDI, R.; GORRIERI, R. (Ed.). **Foundations of Security Analysis and Design**. [S.l.]: Springer Berlin Heidelberg, 2001, (Lecture Notes in Computer Science, v. 2171). p. 137–196. ISBN 978-3-540-42896-1. Cited 3 times on pages 21, 33, and 34.

SHAFIQUE, M.; LABICHE, Y. A systematic review of state-based test tools. **International Journal on Software Tools for Technology Transfer**, Springer Berlin Heidelberg, v. 17, n. 1, p. 59–76, 2013. Cited on page 86.

SIMÃO, A.; PETRENKO, A.; YEVTUSHENKO, N. Generating reduced tests for fsms with extra states. In: NUNEZ, M.; BAKER, P.; MERAYO, M. (Ed.). **Testing of Software and Communication Systems**. Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5826). p. 129–145. ISBN 978-3-642-05030-5. Available: <http://dx.doi.org/10.1007/978-3-642-05031-2_9>. Cited 5 times on pages 23, 31, 32, 33, and 51.

SONG, B.; CHEN, S. Roles-based access control modeling and testing for web applications. In: **Third World Congress on Software Engineering (WCSE 2012)**. [S.l.: s.n.], 2012. p. 57–62. Cited on page 87.

UTTING, M.; PRETSCHNER, A.; LEGEARD, B. A taxonomy of model-based testing approaches. **Software Testing, Verification and Reliability**, John Wiley & Sons, Ltd, v. 22, n. 5, p. 297–312, 2012. ISSN 1099-1689. Cited on page 22.

WOHLIN, C.; RUNESON, P.; HST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLN, A. **Experimentation in Software Engineering**. [S.l.]: Springer Publishing Company, Incorporated, 2014. ISBN 3642432263, 9783642432262. Cited 2 times on pages 55 and 74.

YOO, S.; HARMAN, M. Regression testing minimization, selection and prioritization: A survey. **Softw. Test. Verif. Reliab.**, John Wiley and Sons Ltd., Chichester, UK, v. 22, n. 2, p. 67–120, Mar. 2012. ISSN 0960-0833. Available: <http://dx.doi.org/10.1002/stv.430>. Cited 2 times on pages 39 and 57.

APPENDIX

A

SYSTEMATIC REVIEW OF RBAC POLICIES

In this Appendix, we present the research protocol of a systematic study to identify scientific papers on RBAC testing (Section A.1), the results obtained (Section A.2), and the policies extracted (Section A.3).

A.1 Research Protocol

Objective

The primary objective of this investigation was to identify experimental studies on access control testing using transition based notations for RBAC systems.

Research Questions

The following research questions (RQ) were established for this systematic study.

RQ1: What kind of transition based notations have been used on access control testing?

RQ1.1: What are the most used test selection criteria?

RQ1.2: What kind of test generation technologies have been used?

RQ2: What policies under test have been used in these investigations?

The research question RQ1 was defined to identify what are the most used transition based notations for specifying and testing access control systems. The RQ1.1 and RQ1.2 were designed to give a picture of how transition based notations have been used on test generation for access control testing. The RQ2 was defined to identify what testing artifacts (RBAC policies) have been considered as system under test.

Search String

Based on the research questions, the search string used in this systematic study was designed in order to essentially cover three scopes: access control, transition based modelling notations, and software testing. In this sense, we have adapted Shafique and Labiche (2013) search string for investigating transition based testing tools by including terms related to *access control*. Thus, the following search string was defined: *("Access Control" OR "Control Policy" OR "Control Policies") AND ("transition system" "statechart" OR "statecharts" OR "state model" OR "state machine" OR "state machines" OR "automata") AND ("Testing" OR "Test" OR "Verification" OR "Validation")*. Masood *et al.* (2009) and Damasceno, Delamaro and Simão (2014) papers were used as parameter for defining this string. The search string was applied at IEEExplore [1], ScienceDirect [2], ACM Digital Library [3], ISI Web of Science [4], and Scopus [5] in September 25, 2015. The StArt software [6] was used as supporting tool during this systematic study.

Inclusion and Exclusion Criteria

The following inclusion and exclusion criteria were defined:

1. Only papers investigating RBAC testing will be included;
2. Only papers using transition based notations for test generation will be included;
3. Only papers performing experimental investigations will be included;
4. Only papers written in English will be included;
5. Papers not written in English will be excluded;
6. Any kind of gray literature will be excluded;

A.2 Results Obtained

At the end of the inclusion and exclusion, ten papers were selected and are listed in Table 26. Initially, the following information were extracted from each of the papers: Title, authors, year of publication, Source, and type of publication.

1 <http://ieeexplore.org/>
2 <http://www.sciencedirect.com/>
3 <http://dl.acm.org/>
4 <http://webofscience.com>
5 <http://www.scopus.com/>
6 <http://lapes.dc.ufscar.br/tools/start_tool>

Table 26 – Papers on RBAC testing

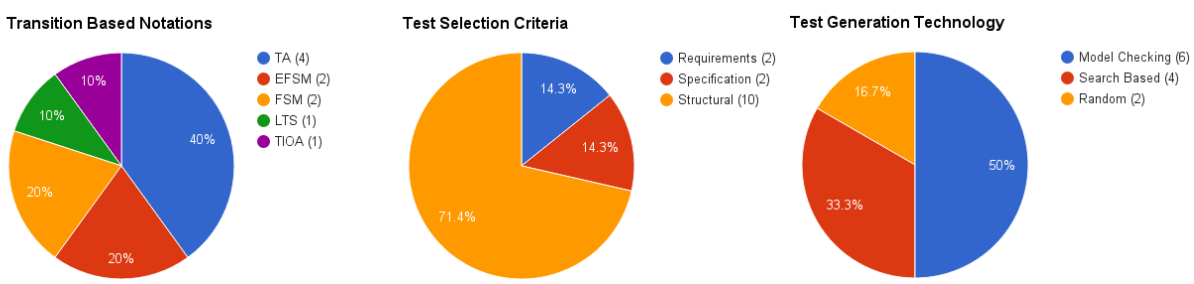
ID	Citation	Paper title	Authors	Year	Source	Type
A1	(DURY *et al.*, 2007)	Formal Verification of Business Workflows and Role Based Access Control Systems	Dury, A. and Boroday, S. and Petrenko, A. and Lotz, V.	2007	International Conference on Emerging Security Information, Systems, and Technologies (SecureWare)	Conference
A2	(SONG; CHEN, 2012)	Roles-based Access Control Modeling and Testing for Web Applications	Bo Song and Shengbo Chen	2012	Third World Congress on Software Engineering (WCSE)	Conference
A3	(MASOOD *et al.*, 2009)	Scalable and Effective Test Generation for Role-Based Access Control Systems	Masood, A. and Bhatti, R. and Ghafoor, A. and Mathur, A.P.	2009	IEEE Transactions on Software Engineering	Journal
A4	(MASOOD; GHAFOOR; MATHUR, 2010b)	Fault coverage of Constrained Random Test Selection for access control: A formal analysis	Masood, A.a and Ghafoor, A.b and Mathur, A.P.c	2010	Journal of Systems and Software	Journal
A5	(BUGLIESI *et al.*, 2012)	Gran: Model Checking Grsecurity RBAC Policies	Bugliesi, M. and Calzavara, S. and Focardi, R. and Squarcina, M.	2012	IEEE 25th Computer Security Foundations Symposium (CSF)	Symposium
A6	(MONDAL; SURAL, 2008)	Security Analysis of Temporal-RBAC Using Timed Automata	Mondal, S. and Sural, S.	2008	Fourth International Conference on Information Assurance and Security (ISIAS)	Conference
A7	(MONDAL; SURAL; ATLURI, 2009)	Towards formal security analysis of GTRBAC using timed automata	Mondal, S.a and Sural, S.a and Atluri, V.b	2009	Proceedings of ACM Symposium on Access Control Models and Technologies (SACMAT)	Symposium
A8	(MONDAL; SURAL; ATLURI, 2011)	Security analysis of GTRBAC and its variants using model checking	Mondal, S.a and Sural, S.a and Atluri, V.b	2011	Computers & Security	Journal
A9	(GEEPALLA; BORDBAR; OKANO, 2012)	Verification of Spatio-Temporal Role Based Access Control using Timed Automata	Geepalla, E. and Bordbar, B. and Okano, K.	2012	IEEE 3rd International Conference on Networked Embedded Systems for Every Application (NESEA)	Conference
A10	(MASOOD; GHAFOOR; MATHUR, 2010a)	Conformance Testing of Temporal Role-Based Access Control Systems	Masood, A. and Ghafoor, A. and Mathur, A.P.	2010	IEEE Transactions on Dependable and Secure Computing	Journal

Moreover, the following information were also extracted from each paper in order to answer the research questions: Modelling notation used, Test selection criteria, Test generation technology, and policy under test.

All ten papers used testing methods used structural coverage as selection criteria. From the selected papers, four used Timed Automata (TA) as modelling notation, one used Temporized Input Output Automata (TIOA), one other used Labelled Transition System (LTS), two used Extended Finite State Machines (EFSM), and two used Finite State Machines. The papers A3 and A4 also used specification and requirements based selection criteria. The *model checking* was the most used technology, where six papers used it. Search based approaches was used on four papers and only two used random generation. Figure 21a presents a chart showing the quantity of studies using each of the identified approaches, Figure 21b shows the total amount of studies using each of the selection criteria, and Figure 21c presents a chart with the proportion of usage of each test generation technology.

Figure 21 – Notations, selection, and generation criteria used on RBAC testing

(a) Transition based notations (b) Test selection criteria (c) Test generation technology



The information extracted from each paper are presented in Table 27 and the policies under test are presented in the next section.

Table 27 – Papers on RBAC testing - Information Extracted

ID	Notation	Selection Criteria	Technology	Policy under test	Source code
A1	EFSM	Structural	Model Checking	Procure to Stock	12
A2	EFSM	Structural	Search based	Experience points and Roles	5
A3	FSM	Structural, Requirements, Specification	Search based, Random	Hospital	7, 9
A4	FSM	Structural, Requirements, Specification	Search based, Random	2-Role, 1-User, 2-Permission	11
A5	LTS	Structural	Model Checking	n/a	-
A6	TA	Structural	Model Checking	Full & Part Time Employee	5
A7	TA	Structural	Model Checking	4-Role, 16-User, 4-Permission	-
A8	TA	Structural	Model Checking	3-Role, 12-User, 5-Permission	15
A9	TA	Structural	Model Checking	SECURE Banking	-
A10	TIOA	Structural	Search based	Senior & Trainee Doctor	14

FSM: Finite State Machine | EFSM: Extended FSM | LTS: Labeled Transition System
TA: Timed Automata | TIOA: Timed Input-Output Automata
n/a: Not available / - : Not extracted

A.3 Policies Extracted

The policies under test were also extracted from each of the papers and textually described. The adapted versions of the policies are also presented below.

Source code 5: RBAC policy - ExperiencePoints

```
1 users (3):  {u1,u2,u3}
2 roles (4):  {Admin,Bronze,Silver,Gold}
3 Su (2): {(u1,2),(u2,2)}
4 Du (0): {}
5 Sr (0): {}
6 Dr (0): {}
7 UR (3): {(u1,Admin),(u1,Silver),(u3,Gold)}
8 SSoD (0): {}
9 DSoD (1): {({Silver,Bronze,Gold},1)}
```

Source code 6: RBAC policy - ExperiencePointsv2

```
1 users (2):  {u1,u2}
2 roles (4):  {Admin,Bronze,Silver,Gold}
3 Su (0): {}
4 Du (0): {}
5 Sr (1): {(Admin,1)}
6 Dr (0): {}
7 UR (0): {}
8 SSoD (2): {({Silver,Bronze,Admin,Gold},2),({Silver,Bronze,Gold},1)}
9 DSoD (1): {({Silver,Bronze,Admin,Gold},1)}
```

Source code 7: RBAC policy - Masood2009P1

```
1 users (5):  {U1,U2,U3,U4,U5}
2 roles (4):  {R1,R2,R3,R4}
3 Su (0): {}
4 Du (5): {(U1,2),(U2,2),(U3,2),(U4,2),(U5,1)}
5 Sr (0): {}
6 Dr (4): {(R1,3),(R2,1),(R3,3),(R4,2)}
7 UR (9): {(U1,R1),(U4,R1),(U2,R2),(U5,R2),(U1,R3),(U2,R3),(U4,R3),(U4,
      R4),(U5,R4)}
8 SSoD (1): {({R1,R2},1)}
9 DSoD (1): {({R3,R2},1)}
```

Source code 8: RBAC policy - Masood2009P1v2

```
1 users (3):  {U1,U2,U4}
2 roles (4):  {R1,R2,R3,R4}
3 Su (0): {}
4 Du (3): {(U1,2),(U2,2),(U4,2)}
5 Sr (0): {}
6 Dr (4): {(R1,3),(R2,1),(R3,3),(R4,2)}
7 UR (3): {(U1,R1),(U2,R1),(U4,R1)}
8 SSoD (3): {({R1,R2},1),({R3,R4,R1},1),({R3,R2},1)}
9 DSoD (0): {}
```

Source code 9: RBAC policy - Masood2009P2

```
1 users (4):  {U1,U2,U3,U4}
2 roles (6):  {R1,R2,R3,R4,R5,R6}
3 Su (0): {}
4 Du (2): {(U1,2),(U2,2)}
5 Sr (0): {}
6 Dr (6): {(R1,1),(R2,1),(R3,1),(R4,1),(R5,1),(R6,1)}
7 UR (8): {(U1,R1),(U1,R3),(U2,R3),(U2,R4),(U1,R5),(U2,R5),(U1,R6),(U2,
      R6)}
8 SSoD (2): {({R3,R1,R2},2),({R4,R5},1)}
9 DSoD (2): {({R1,R2},1),({R3,R4,R2},2)}
```

Source code 10: RBAC policy - Masood2009P2v2

```
1 users (2):  {U1,U2}
2 roles (5):  {R1,R2,R3,R4,R5}
3 Su (1): {(U2,1)}
4 Du (1): {(U1,2)}
5 Sr (0): {}
```

```
6 Dr (5): {(R1,1),(R2,1),(R3,1),(R4,1),(R5,1)}
7 UR (4): {(U1,R1),(U1,R2),(U2,R2),(U1,R4)}
8 SSoD (2): {({R3,R1,R2},2),({R4,R5},1)}
9 DSoD (2): {({R1,R2},1),({R3,R4,R2},2)}
```

Source code 11: RBAC policy - Masood2010Example1

```
1 users (2):  {u1,u2}
2 roles (1):  {r1}
3 Su (2): {(u1,1),(u2,1)}
4 Du (2): {(u1,1),(u2,1)}
5 Sr (1): {(r1,2)}
6 Dr (1): {(r1,1)}
7 UR (2): {(u1,r1),(u2,r1)}
8 SSoD (0): {}
9 DSoD (0): {}
```

Source code 12: RBAC policy - ProcureToStock

```
1 users (4):  {Alice,Carol,Bob,Employee}
2 roles (5):  {Role1,Role2,Role3,Role4,Role5}
3 Su (0): {}
4 Du (0): {}
5 Sr (0): {}
6 Dr (0): {}
7 UR (5): {(Carol,Role1),(Alice,Role2),(Carol,Role2),(Carol,Role5),(
    Employee,Role5)}
8 SSoD (0): {}
9 DSoD (0): {}
```

Source code 13: RBAC policy - ProcureToStockV2

```
1 users (3):  {Alice,Carol,Bob}
2 roles (5):  {Role1,Role2,Role3,Role4,Role5}
3 Su (2): {(Bob,1),(Carol,1)}
4 Du (0): {}
5 Sr (1): {(Role5,1)}
6 Dr (0): {}
7 UR (0): {}
8 SSoD (1): {({Role4,Role3,Role2},1)}
9 DSoD (0): {}
```

Source code 14: RBAC policy - SeniorTraineeDoctor

```
users (2):  {Bob,Alice}
roles (2):  {SeniorDoctor,TraineeDoctor}
Su (2): {(Bob,2),(Alice,1)}
Du (2): {(Bob,2),(Alice,1)}
Sr (2): {(SeniorDoctor,1),(TraineeDoctor,2)}
Dr (2): {(SeniorDoctor,1),(TraineeDoctor,2)}
UR (2): {(Bob,SeniorDoctor),(Alice,TraineeDoctor)}
SSoD (1): {({SeniorDoctor,TraineeDoctor},1)}
DSoD (0): {}
```

Source code 15: RBAC policy - user11roles2

```
users (12): {U0,U1,U2,U3,U4,U5,U6,U7,U8,U9,U10,U11}
roles (3):  {R0,R1,R2}
Su (2): {(U1,2),(U0,2)}
Du (0): {}
Sr (2): {(R0,9),(R1,10)}
Dr (3): {(R2,3),(R1,9),(R0,7)}
UR (4): {(U1,R1),(U0,R2),(U1,R2),(U2,R2)}
SSoD (0): {}
DSoD (0): {}
```

Source code 16: RBAC policy - user11roles2_v2

```
users (11): {U1,U2,U3,U4,U5,U6,U7,U8,U9,U10,U11}
roles (2):  {R1,R2}
Su (11):  {(U1,1),(U2,1),(U3,1),(U4,1),(U5,1),(U6,1),(U7,1),(U8,1),(
    U9,1),(U10,1),(U11,1)}
Du (0): {}
Sr (2): {(R1,1),(R2,1)}
Dr (0): {}
UR (0): {}
SSoD (1): {({R1,R2},2)}
DSoD (0): {}
```

APPENDIX

B

RBAC-BT SOFTWARE

The RBAC-BT is a java software designed to support the execution of experiments on FSM-based testing of RBAC systems. It has been developed in the context of the master degree project of *Carlos Diego Nascimento Damasceno* at the Institute of Mathematical and Computer Sciences (ICMC), University of São Paulo (USP) (2014-2016). This project investigated FSM-based test generation methods and test prioritization on RBAC. All artifacts used in this work can be found at <https://github.com/damascenodiego/rbac-bt>. In this document the main aspects of the RBAC-BT v1.0 software are described. First, in section B.1, the structure of the RBAC-BT repository is described followed by the main features and the commands for using the RBAC-BT tool, in section B.2.

B.1 RBAC-BT Repository

The RBAC-BT (v1.0) repository is organized in five main directories (Table 28). The *rbac-bt/* directory contains an eclipse project with the source code of the RBAC-BT software. All policies used as SUT are available in the *policies_example/* directory. The *fragmentTestSuite/* directory contains a java software designed to fragment test suites for analysing test prioritization approaches. The *experiments/msc_dissertation/* directory contains the data of experiments performed in this master dissertation. The *doc/* directory contains the documentation of the RBAC-BT project.

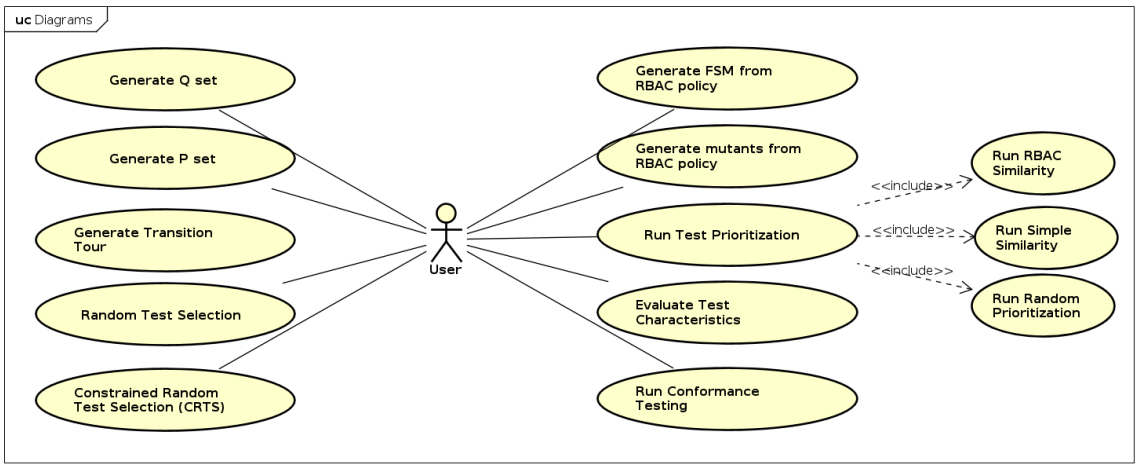
Table 28 – Organization of the RBAC-BT repository

Repository name	Description
rbac-bt/	Contains the source code of the RBAC-BT software.
doc/	Contains the documentation of the RBAC-BT project
policies_example/	Contains the seven original RBAC policies and the five adapted versions.
fragmentTestSuite/	Contains a software designed to generate test suite fragments
experiments/msc_dissertation/	Contains the data obtained with the experiments performed in this work

B.2 RBAC-BT Main Features

The RBAC-BT software has been implemented to support the four steps of software testing: test generation, test selection, test prioritization and test assessment. Figure 22 shows the use case diagram of all features supported by the RBAC-BT. The RBAC-BT tool supports: Generation of FSM from RBAC policies; Generation of mutants from RBAC policies; Test prioritization; Execution of conformance testing; and Evaluation of test characteristics. The RBAC-BT software also supports state cover method, transition cover method, transition tour method, CRTS method, and random test selection but these features will not be discussed as they were not used in the experiments.

Figure 22 – RBAC-BT use case diagram



Generate FSM from RBAC policy

Given an RBAC policy P, the RBAC-BT tool is able to generate an $FSM(P)$ describing the access control decisions which an RBAC mechanism should enforce based on P. The conversion of RBAC policies to FSM models consists on a Breadth First Search (BFS) algorithm seeking for all the reachable states that the mutable elements of P grant. To generate an $FSM(P)$ using the RBAC-BT tool run the following command:

```
1  $ java -jar rbac-bt.jar <options> -r2f <rbac_filename>
```

In $< options >$ the user can set the format to save the generated the FSMs. By default the *-fsm* parameter is considered. It saves the FSMs in XML format. The *-kk* parameter enables to save the FSM using the kiss file format. In $< rbac\_filename >$ the user will set the path to the RBAC policy described using the XML format. After run the command given above, the FSM will be generated and saved with the same filename but different extension. The extension will depend on the $< options >$ parameter.

Generate mutants from RBAC policy

The RBAC-BT tool also supports mutation analysis of RBAC policies. The tool uses the fault model proposed by Masood *et al.* (2009) to generate mutants of RBAC policies. The following mutation operators are supported: (i) *Replacement of users and/or roles from UR relationships*; (ii) *Replacement of users from SSoD and DSoD sets*; (iii) *Increment of static and dynamic cardinality constraints of users and roles*; and (iv) *Decrement of static and dynamic cardinality constraints of users and roles*.

To generate the mutants of an RBAC policy run the following command:

```
1  $ java -jar rbac-bt.jar -rmut <rbac_filename>
```

After run the given command, a set of folders containing the mutants generated from each mutation operator will be created.

Run Test Prioritization

The RBAC-BT tool supports three different test prioritization algorithms (Simple Dissimilarity, RBAC Similarity, and Random Prioritization). To execute test prioritization using the RBAC-BT tool run the following command:

```
1  $ java -jar rbac-bt.jar -prtz <t> -rbac <rbac> -mode <prtz>
```

In $< t >$ the user defines the test suite to be prioritized given an RBAC policy $<$ *rbac* $>$. The *-mode* parameter is optional and by default it uses Simple Dissimilarity. Other prioritization techniques can be used setting the following values in $<$ *prtz* $>$: Simple similarity-Based Prioritization (*cartax*), RBAC similarity-Based Prioritization (*damasc*), and Random Prioritization (*random*).

Run Conformance Testing

The RBAC-BT tool is able to perform conformance testing of an RBAC policy P against a set of RBAC mutants P' given a test suite. In conformance testing, the RBAC policy P and all the P' mutants are compared based on the outputs obtained given a test suite. If any divergence between P and P' outputs is detected, the P' mutant is removed from the pool of alive mutants. To run conformance testing execute the following command:

```
1  $ java -jar rbac-bt.jar -ct <rbac> -mutants <mut> -test <t>
```

With this command the RBAC-BT tool loads an RBAC policy ($<$ *rbac* $>$), a set of RBAC mutants listed in a text file ($<$ *mut* $>$), and a test suite ($< t >$). The program display the results using the standard output and prints the name of the policy tested followed by the effectiveness percentual and the list of alive mutants.

The text file listing the mutants ($< mut >$) must have the path to each RBAC mutant policy file one per line. The test suite ($< t >$) must be a text file where each line have a sequences of three numbers. Each sequence of three numbers refers to an input of the input domain of the RBAC policy ($< rbac >$). For example, 000 refers to the first input, 001 to the second, and so on.

Evaluate Test Characteristics

The RBAC-BT tool can summarize the characteristics of test suites, such as number of resets, test suite length, and test case length, given an RBAC policy.

All this information is print using standard output. To display test characteristics run the following command:

```
1  $ java -jar rbac-bt.jar -testCharact -rbac <rbac> -test <t>
```

The RBAC policy $< rbac >$ is loaded with the test suite $< t >$ and all the information listed before are printed using the standard output.