Damasceno *et al. Journal of Software Engineering Research and Development* (2018) 6:1
DOI 10.1186/s40411-017-0045-x

Journal of Software Engineering Research and Development

RESEARCH Open Access

Similarity testing for role-based access control systems

Carlos Diego N. Damasceno[1,2*], Paulo C. Masiero[1,2] and Adenilso Simao[1,2]

*Correspondence: damascenodiego@usp.br
[1]Institute of Mathematics and Computer Science, University of Sao Paulo (ICMC-USP), Trabalhador Sao-carlense Avenue, 400, 13566-590 Sao Carlos-SP, Brazil
[2]Software Engineering Laboratory – LabES, Trabalhador Sao-carlense Avenue, 400, Room 6-208, 13566-590 Sao Carlos-SP, Brazil

Abstract

Context: Access control systems demand rigorous verification and validation approaches, otherwise, they can end up with security breaches. Finite state machines based testing has been successfully applied to RBAC systems and enabled to obtain effective test cases, but very expensive. To deal with the cost of these test suites, test prioritization techniques can be applied to improve fault detection along test execution. Recent studies have shown that similarity functions can be very efficient at prioritizing test cases. This technique is named *similarity testing* and assumes the hypothesis that resembling test cases tend to have similar fault detection capabilities. Thus, there is no gain from similar test cases, and fault detection ratio can be improved if test diversity increases.

Objective: In this paper, we propose a similarity testing approach for RBAC systems named *RBAC similarity* and compare to simple dissimilarity and random prioritization. RBAC similarity combines the dissimilarity degree of pairs of test cases with their relevance to the RBAC policy under test to maximize test diversity and the coverage of its constraints.

Method: Five RBAC policies and fifteen test suites were prioritized using each of the three test prioritization techniques and compared using the Average Percentage Faults Detected metric.

Results: Our results showed that the combination of the dissimilarity degree to the relevance of a test case to RBAC policies in the *RBAC similarity* can be more effective than random prioritization and simple dissimilarity, by itself, in most of the cases.

Conclusion: The RBAC similarity criterion is suitable as a test prioritization criteria for test suites generated from finite state machine models specifying RBAC systems.

Keywords: Finite state machines, Role-Based Access Control (RBAC), Test prioritization, Similarity testing

1 Introduction

Access control is one of the major pillars of software security. It is responsible for ensuring that only intended users can access data and only the required permissions to accomplish a task is guaranteed (Ferraiolo et al. 2007). In this context, the Role-Based Access Control (RBAC) model has been established as one of the most significant access control paradigms. In RBAC, users receive privileges through role assignments and activate them during sessions (ANSI 2004). Despite its simplicity, mistakes can occur during development and lead to faults, or either security breaches. Therefore, software verification and validation becomes necessary.

Finite State Machine (FSM) has been widely used for model-based testing (MBT) of reactive systems (Broy et al. 2005). Previous investigations using random FSMs have shown that recent test generation methods (e.g., SPY (Simão et al. 2009)), compared to traditional methods (e.g., W (Chow 1978) and HSI (Petrenko and Bochmann 1995)), tend to rely on fewer and longer test cases, reducing the overall test cost without impacting test effectiveness (Endo and Simao 2013). In the RBAC domain, although very effective and less costly, recent test generation methods still tend to output large amounts of test cases (Damasceno et al. 2016). Thus, there is a need for additional steps during software testing, such as test prioritization (Mouelhi et al. 2015).

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 and Harman 2012). A test prioritization criterion that has recently shown very promising results is *similarity testing* (Cartaxo et al. 2011; Bertolino et al. 2015). In similarity testing, we assume that resembling test cases tend to cover identical parts of an SUT, have equivalent fault detection capabilities, and no additional gain can be expected if executed simultaneously. This concept has been investigated under MBT (Cartaxo et al. 2011), access control testing (Bertolino et al. 2015) and software product line (SPL) testing (Henard et al. 2014) domains, but it has never been applied to RBAC. Moreover, since the fault detection effectiveness of test criteria are strongly related to its ability to represent faults of specific domains (Felderer et al. 2015), similarity testing may not be necessarily effective on RBAC domain.

In this paper, we investigate similarity testing for RBAC systems. A similarity testing criterion named *RBAC similarity* is introduced and compared to random prioritization and simple dissimilarity criteria using Average Percentage Faults Detected (APFD) metric, five RBAC policies, and three FSM-based testing methods. Our results show that RBAC similarity makes test prioritization more suitable to the specificities of the RBAC model and achieve higher APFD values compared to simple dissimilarity and random prioritization, in most of the cases.

This paper is organized as follows: Section 2 shows the theoretical background related to our investigation. Sections 2.1 to 2.3 give a brief introduction to FSM-Based Testing. The RBAC model and an FSM-based testing approach for RBAC systems are introduced in Sections 2.4 and 2.5. The test case prioritization problem and similarity testing are discussed in Section 2.6. Section 3 details our proposed similarity testing criteria named *RBAC similarity*. Section 4 depicts the experiment we performed to compare *RBAC similarity* to simple dissimilarity and random prioritization techniques. The results obtained from our experiments are analyzed and discussed. The threats to validity and final remarks are presented in Sections 6 and 7, respectively.

2 Background

This section introduces the background behind our similarity testing approach for RBAC systems. First, we present the concept of FSM-based testing and three test generation methods (i.e., W, HSI, and SPY) which were considered in this study. Second, the RBAC model and an FSM-based testing approach for RBAC systems are described. At last, the test case prioritization problem and the specificities of the similarity testing are detailed.

2.1 Finite state machine based testing

A Finite State Machine (FSM) is a hypothetical machine composed of 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 set of input symbols, O is the set of output symbols, $D \subseteq S \times I$ is the specification domain, $\delta : D \rightarrow S$ is the transition function, and $\lambda : D \rightarrow O$ is the output function. An FSM always has a single current (origin) state $s_i \in S$ which changes to destination (tail) state $s_j \in S$ by applying an input $x \in I$ where $s_j = \delta(s_i, x)$, and returns an output $y = \lambda(s_i, x)$. An input x is defined for s if in state s there is a transition consuming input x (i.e. $(s, x) \in D$). Such transition is said *defined*. An FSM is *complete* if all inputs are defined for all states, otherwise it is *partial*. Figure 1 depicts an example of a complete FSM with three states $\{q_0, q_1, q_2\}$.

A sequence $\alpha = x_1 x_2 ... x_n \in I$ is defined for state $s \in S$, if there are states $s_1, s_2, ..., s_{n+1}$ such that $s = s_1$ and $\delta(s_i, x_i) = s_{i+1}$, for all $1 \leq i \leq n$. 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 input sequence ω. An empty sequence is denoted by ϵ and a sequence α is a proper prefix of β, denoted by $\alpha < \beta$, if $\beta = \alpha\omega$ for a given $\omega \neq \epsilon$. 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*.

The transition and output functions can be lifted to input sequences as usual; for the empty sequence ϵ, we have that $\delta(s, \epsilon) = s$ and $\lambda(s, \epsilon) = \epsilon$. For a sequence αx defined for state s, we have that $\delta(s, \alpha x) = \delta(\delta(s, \alpha), x)$ and $\lambda(s, \alpha x) = \lambda(s, \alpha)\lambda(\delta(s, \alpha), x)$. A sequence $\alpha = x_1 x_2 ... x_n \in I$ is a transfer sequence from s to s_{n+1} if $\delta(s, \alpha) = s_{n+1}$, thus s_{n+1} is reachable from s. If every state of an FSM is reachable from s_0 then it is *initially connected* and if every state is reachable from all states, it is *strongly connected*.

The symbol $\Omega(s)$ denotes all input sequences defined for a state s and Ω_M abbreviates $\Omega(s_0)$, which refers to all defined input sequences for an FSM M. 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 $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$. In addition, if γ is able to distinguish every pair of states of an FSM, it is a *distinguishing sequence*. Considering the FSM presented in Fig. 1, 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 FSMs $M_S = (S, s_0, I, O, D, \delta, \lambda)$ and $M_I = (S', s'_0, I, O', D', \delta', \lambda')$ are equivalent if their initial states are. Two states s_i, s_j are equivalent if $\forall\ \alpha \in \Omega(s_i) \cap \Omega(s_j),\ \lambda(s_i, \alpha) =$

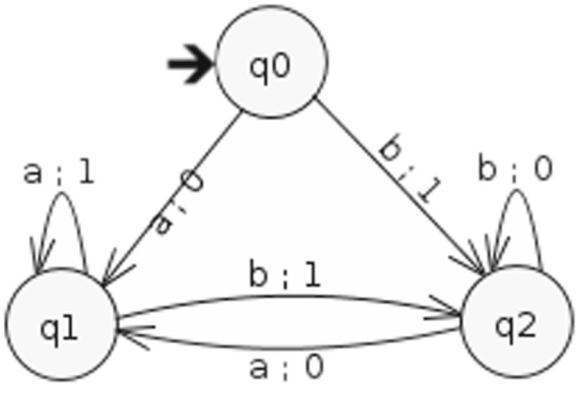

Fig. 1 Example of complete FSM

$\lambda'(s_j, \alpha)$. An FSM M may have a *reset operation*, denoted by r, which takes to s_0 regardless the current state. An input sequence $\alpha \in \Omega_M$ starting with a reset symbol r is a *test case* of M. A test suite T consists of a finite set of test cases of M, such that there are no $\alpha, \beta \in T$ where $\alpha < \beta$. Prefixes $\alpha < \beta$ are excluded from test suite since the execution of β implies the execution of α. The length of a test case α is represented by $|\alpha|$ and describes the cost of executing α plus the reset operation. The number of test cases of one test suite T also describes the number of resets of T which is depicted as $|T|$.

2.2 Mutation analysis in FSM-based testing

In FSM-based testing, given a specification M, the symbol $\Im(M)$ denotes the set of all deterministic FSMs, variants of M, with the same inputs of M for which all sequences in Ω_M are defined. The set $\Im(M)$ is called *fault domain* for M and these variants of M are named *mutants* and can be obtained either manually or by automatically performing simple syntactic changes using *mutation operators* (Andrews et al. 2006). Given $m \geq 1$, 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 is a test case $t \in T$ that distinguish M from N. The following mutation operators are often used on FSM-based testing (Chow 1978): *change initial state* (CIS), which changes the s_0 of an FSM to s_k, such that $s_0 \neq s_k$; *change output* (CO), which modifies the output of a transition (s, x), using a different function $\Lambda(s, x)$ instead of $\lambda(s, x)$; *change tail state* (CTS), which modifies the destination state of a transition (s, x), using a different function $\Delta(s, x)$ instead of $\delta(s, x)$; and *add extra state* (AES), which inserts a new state such that mutant N is equivalent to M. Figure 2 shows examples of mutants of the FSM shown in Fig. 1 using CIS, CO, CTS, and AES operators. Changes are marked with an asterisk (*).

If the output of a mutant is different from the original FSM, for any test case, the mutant is distinguished (or *killed*) and the seeded fault denoted by the mutant is detected.

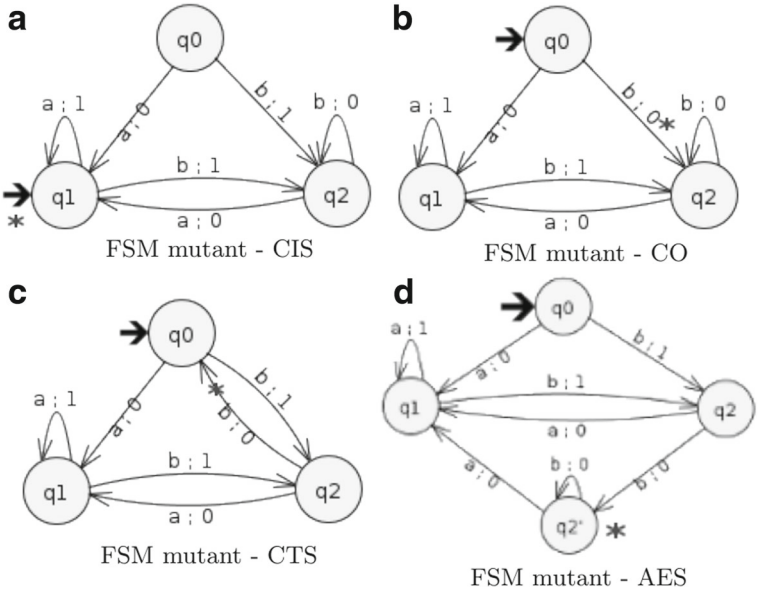

Fig. 2 Examples of FSM Mutants. **a** FSM mutant - CIS. **b** FSM mutant - CO. **c** FSM mutant - CTS. **d** FSM mutant - AES

Moreover, some mutants can be syntactically different but functionally equivalent to the original model. These are called *equivalent mutants*. The process of analyzing if test cases trigger failures and kill mutants is called *mutation analysis* and is often used in software testing research (Jia and Harman 2011; Fabbri et al. 1994).

The main outcome of the mutation analysis is the *mutation score*, which indicates the effectiveness of a test suite. Given a test suite 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 $\#em$ denotes 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 a given domain $\Im_m(M)$ and can detect all faults in any FSM implementation with at most m states. Thus, it scores 1.0, by definition.

2.3 FSM-based testing methods

FSM-based testing relies on FSM models to derive test cases and evaluate if the behavior of an SUT conforms to its specification (Utting et al. 2012). To check this behavioral conformance, two basic sets of sequences are often used: the state cover (Q) and transition cover (P) sets (Broy et al. 2005).

A set of input sequences is a *state cover set* of M if for each state $s_i \in S$ there exists an $\alpha \in Q$ such that $\delta(s_0, \alpha) = s_i$ and $\epsilon \in Q$ to reach the initial state. A set of input sequences P is named *transition cover set* of M if for each transition $(s, x) \in D$ there are sequences $\alpha, \alpha x \in P$, such that $\delta(s_0, \alpha) = s$, and $\epsilon \in P$ to reach the initial state. The transition cover set of an FSM is obtained by generating the *testing tree* of this FSM (Broy et al. 2005). The state and transition cover sets of the FSM depicted in Fig. 1 are respectively $Q = \{\epsilon, a, b\}$ and $P = \{\epsilon, a, aa, ba, b, ab, bb\}$. After obtaining state and transition coverage, FSM-based testing methods require some pre-defined sets to identify the reached parts of an FSM. These are the *characterization set* and *separating families*.

A characterization set (W set) contains at least one input sequence which distinguishes each pair of states of an FSM. Formally, it means that for all pairs of states $s_i, s_j \in S, i \neq j$, $\exists \alpha \in W$ such that $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$.

A separating family, or harmonized state identifiers, is a set of sequences 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 has a common prefix α such that $\alpha \in \Omega(s_i) \cap \Omega(s_j)$ and $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. In the worst case, the separating family is the W set itself.

The characterization set of the FSM model shown in Fig. 1 is $W = \{a, b\}$, and the separating family of states q_0, q_1, q_2 are respectively $H_0 = \{a, b\}$, $H_1 = \{a\}$, and $H_2 = \{b\}$. These sets are building blocks for most traditional and recent testing methods, such as W (Chow 1978; Vasilevskii 1973), HSI (Petrenko and Bochmann 1995), and SPY (Simão et al. 2009).

2.3.1 W method

The W method is the most classic FSM-based test generation algorithm (Chow 1978; Vasilevskii 1973). It uses the P set, to traverse all transitions, concatenated to the W set, for state identification. Moreover, it can also detect an estimated number of extra states using a traversal set $\bigcup_{i=0}^{m-n}(I^i)$, such that $(m - n)$ is the number of extra states and I^i

contains all sequences of length i combining symbols of I. Thus, by concatenating P, the traversal set, and W, the W method can detect $(m - n)$ extra states (e.g., AES mutants). Assuming the FSM in Fig. 1, no extra states ($m = n$) or proper prefixes, W method can generate $T_W = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$, and $|T_W| = 8$.

2.3.2 HSI method

The Harmonized State Identifiers (HSI) method (Petrenko and Bochmann 1995) uses state identifiers H_i to distinguish each state $s_i \in S$ of an FSM model. The HSI test suite is obtained by concatenating the transition cover set P with H_i, such that $\delta(s_0, \alpha) = s_i, s_i \in S$ and $\alpha \in P$. The HSI method can be applied to complete and partial FSMs. Assuming the FSM in Fig. 1, no extra states or proper prefixes, HSI method can generate $T_{HSI} = \{aaa, aba, abb, baa, bba, bbb\}$, and $|T_{HSI}| = 6$, which is 75% the size of T_W.

2.3.3 SPY method

The SPY method (Simão et al. 2009) is a recent test generation method able to generate m-complete test suites *on-the-fly*. First, the state cover set Q is concatenated to the state identifiers H_i. Afterwards, differently from traditional methods, such as W and HSI, the traversal set is distributed over the set containing Q concatenated with H_i based on sufficient conditions (Simão et al. 2009). Thus, by avoiding testing tree branching, test suite length and the number of resets can be reduced.

Experimental studies have indicated that SPY can generate test suites on average 40% shorter than traditional methods (Simão et al. 2009). Moreover, it can achieve higher fault detection effectiveness even if the number of extra states is underestimated (Endo and Simao 2013). Assuming the FSM in Fig. 1, no extra states or proper prefixes, SPY method can generate $T_{SPY} = \{aaaba, abbb, baa, bba\}$, and $|T_{SPY}| = 4$, which is 50% the size of T_W.

2.4 Role-based access control

Access Control (AC) is one of the most important security mechanisms (Jang-Jaccard and Nepal 2014). Essentially, it ensures that only allowed users have access to protected system resources based on a set of rules, named *security policies*, that specify authorizations and access restrictions (Samarati and de Vimercati 2001). In this context, the Role-Based Access Control (RBAC) model has been established as one of the most significant access control paradigms (Ferraiolo et al. 2007). It uses the concept of *grouping privileges* to reduce the complexity of security management tasks (Samarati and de Vimercati 2001).

In RBAC, *roles* describe organizational figures (e.g., functions or jobs) which own a set of responsibilities (e.g., *permissions*). Roles can be assigned or revocated to *users* via *role assignments* and performed under *sessions* through *role activations*. *Role hierarchies* can be specified as inheritance relationships between senior and junior roles (e.g., sales director inherits permissions from sales manager). Thus, the mapping between security policies and the organizational structure can be more natural. These elements compose the ANSI RBAC model (ANSI 2004) which can also be extended to groups of administrative roles and permissions (Ben Fadhel et al. 2015). In Fig. 3, the ANSI RBAC and, within dashed lines, the Administrative RBAC models are depicted.

Masood et al. (2009) define an RBAC policy 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:

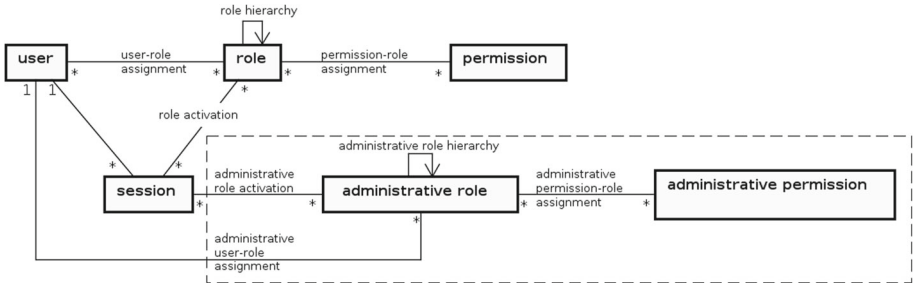

Fig. 3 ANSI RBAC and administrative RBAC

- U and R are 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 the role activation and inheritance hierarchies relationships;
- $I = \{AS, DS, AC, DC, AP, DP\}$ is the finite set of types of RBAC requests which respectively stand for user-role assignments (AS), deassignments (DS), activations (AC) and deactivations (DC); and permission-role activations (AC) and deactivations (DC);
- $S_u, D_u : U \rightarrow \mathbb{Z}^+$ are static and dynamic cardinality constraints on users;
- $S_r, D_r : R \rightarrow \mathbb{Z}^+$ are static and dynamic cardinality constraints on roles;
- $SSoD, DSoD \subseteq 2^R$ are the Static and Dynamic Separation of Duty (SoD) sets, respectively;
- $S_s : SSoD \rightarrow \mathbb{Z}^+$ specifies the cardinality of SSoD sets;
- $D_s : DSoD \rightarrow \mathbb{Z}^+$ specifies the cardinality of DSoD sets.

Role inheritance hierarchy is a role-to-role relationship (e.g., $r_j \leq_I r_s$) that enable users assigned to a senior role (r_s) to have access to all permissions of junior roles (r_j). Role activation is a variant of role hierarchy (e.g., $r_j \leq_A r_s$) which enable users assigned to a senior role (r_s) to activate junior roles (r_j) without being directly assigned to that junior role (Masood et al. 2009). Cardinality constraints specify a bound on the cardinality of user-role assignment and role activation relationships (Ben Fadhel et al. 2015). Static cardinality constraints (S_u and S_r) bound user-role assignments and dynamic cardinality constraints (D_u and D_r) limit user-role activations (i.e., role activations) and they can be specified from a user (S_u and D_u, respectively) and role (S_r and D_r, respectively) perspectives. Separation of Duty (SoD) constraints define static and dynamic ($SSoD$ and $DSoD$, respectively) mutual exclusion relationships among roles based on a positive integer number $n \geq 2$ to avoid the simultaneous assignments or activations of conflicting roles (ANSI 2004) (e.g., given $SSoD = \{\text{staff, accountant, director}\}$ and $n = 2$, $S_{SSoD} = 2$ defines that no user can be assigned to more than two roles of $SSoD_i$ set). Listing 1 shows an example of RBAC policy with two users (line 1), one role (line 2), and two permissions (line 3).

```
1 U = {u1,u2} /* users */
2 R = {r1} /* roles */
3 Pr = {pr1,pr2} /* permissions */
4 UR = {(u1,r1)} /* user-role assignments */
5 PR = {(r1,pr1), (r1,pr2)} /* permission-role assignments */
6 S_u(u1) = S_u(u2) = 1 /* static user cardinality constraints */
7 D_u(u1) = D_u(u2) = 1 /* dynamic user cardinality constraints */
8 S_r(r1) = 2 /* static role cardinality constraints */
9 D_r(r1) = 1 /* dynamic role cardinality constraints */
```

Listing 1 Example of RBAC policy

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 and activate at most one role (line 6-7). Role $r1$ can be assigned to at most two users (line 8); however, it can be activated by one user per time (line 9).

2.5 FSM-based testing of RBAC systems

Masood et al. (2009) propose an approach based on FSMs to specify and test the behavior of RBAC systems. Given an RBAC policy P, an $FSM(P)$ consists of a complete FSM modeling all access control decisions that an RBAC mechanism must enforce. Formally, an $FSM(P)$ is 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 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 where $I_P = \{(rq, up, r)\}$ for all $rq \in I$, $u \in \{U \cup Pr\}$ and $r \in R\}$;
- O is the output domain formed by *granted* and *denied*;
- $D = S_P \times I_P$ is the specification domain;
- $\delta_P : D \rightarrow S_P$ is the state transition function; and
- $\lambda_P : D \rightarrow O$ is the output function.

Each state $s \in S_P$ is labeled using a sequence of pairs of bits containing one pair for each combination of *user-role* and *permission-role*. A pair *user-role* can be assigned (10), activated (11) or not assigned (00); and a pair *permission-role* can be assigned (10) or not assigned (00). The maximum number of states of $FSM(P)$ is bounded to $3^{|U|\times|R|}$ and the number of reachable states depends on the constraints of P. The set of input symbols I_P contains all combinations of users, roles, permissions and types of RBAC requests which can be applied to P. Formally, it means that $I_P = \{(rq, up, r)\}\ \forall\ rq \in I$, $up \in \{U \cup Pr\}$ and $r \in R$.

Transitions of $FSM(P)$ denote access control decisions on destination states ($s_j \in S_P$) and output symbols (*granted* or *denied*) given the specification domain, that is complete (Masood et al. 2009) and composed by pairs of an origin state ($s_i \in S_P$), and an input symbol $(rq, up, r) \in I_P$, and the constraints of P. Given the constraints of P, an origin state s_i and an input symbol (rq, up, r), a destination state $s_j = \delta_P(s_i, (rq, up, r))$ is defined by flipping the bits of s_i label related to an user (or permission) up and role r, if the constraints of P allow such request. This procedure denotes how the state transition function δ_P operates.

Regarding the output function λ_P, a *denied* symbol is returned to inputs (requests) which do not change the state of P, such as *user-role* assignments already performed or requests denied due to some cardinality constraint. Thus, *denied* is only returned on self-loops. Transitions with different origin and destination states always return *granted*. The

generation of an $FSM(P)$ can be iteratively performed by evaluating all defined inputs of state s_0 given the constraints of P ($\Omega_{FSM(P)}$).

Figure 4 shows the $FSM(P)$ of the RBAC policy presented in Listing 1. Self-loop transitions, corresponding to requests returning *denied*, and transitions related to permissions are not shown to keep the figure uncluttered. The initial state 1000 depicts line 4 of Listing 1 where $u1$ is assigned to $r1$. From state 1000 all defined inputs are applied once to reach states 1100, 1010 and 0000 where respectively user $u1$ activates $r1$, $u1$ and $u2$ are assigned to $r1$, and none is assigned to $r1$. This procedure is iteratively repeated over all reached states until no new state is obtained. At the end, the resulting $FSM(P)$ has a total of eight states due to $Dr(r1) = 1$ which makes state 1111 unreachable, but not $9 = 3^{|U|\times|R|}$, which is the maximum number of states.

2.5.1 Test generation from FSM(P)

Given an RBAC system implementing a policy P, FSM-based testing can verify if the behavior of such system conforms to P using its respective $FSM(P)$ and some test generation method, such as $\mathbb{W}$ or transition cover (Masood et al. 2009).

Let $\mathcal{R}$ denote the set of all RBAC policies. Given a policy $P \in \mathcal{R}$, the set $\mathcal{R}$ can be partitioned into two subsets of policies: Equivalent (*conforming*) to P ($\mathcal{R}^P_{conf}$); and *Faulty* policies ($\mathcal{R}^P_{fault}$). Since $\mathcal{R}$ is infinitely large, Masood et al. (2009) proposed a mutation analysis technique to measure the effectiveness of a test suite as its ability to detect if an RBAC system behaves as some faulty policy $P' \in \mathcal{R}^P_{fault}$.

The RBAC mutation analysis restricts $\mathcal{R}^P_{fault}$ to be finite by only considering policies mutants $P' = (U, R, Pr, UR', PR', \leq'_A, \leq'_I, I, S'_u, D'_u, S'_r, D'_r, SSoD', DSoD', S'_s, D'_s)$ generated by making simple changes to policy $P = (U, R, Pr, UR, PR, \leq_A, \leq_I, I, S_u, D_u, S_r, D_r, SSoD, DSoD, S_s, D_s)$. Note that all mutants share the same set of users (U), roles (R), permissions (Pr) and inputs (I) of the original policy P. The set $\mathcal{R}^P_{fault}$ of faulty policies is generated by making changes using two kinds of operators: *mutation operators* and *element modification operators*.

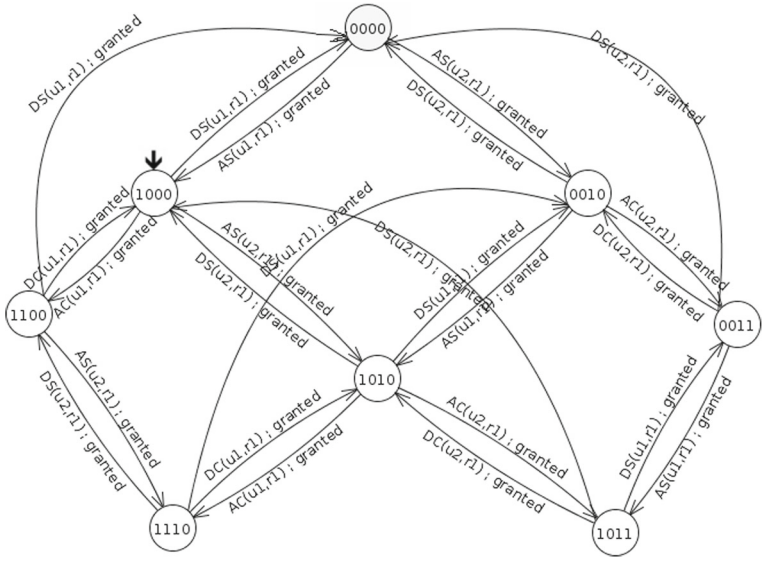

Fig. 4 Example of $FSM(P)$ specifying an RBAC policy

The *mutation operators* generate RBAC mutants by adding, modifying and removing elements from *UR*, *PR*, $\leq_A$, $\leq_I$, *SSoD*, and *DSoD* sets (e.g. add role to *SSoD* set). The *element modification operators* mutate policies by incrementing or decrementing the cardinality constraints S_u, D_u, S_r, D_r, S_s, and D_s. Each of these RBAC faults has corresponding faults on the FSM domain (Chow 1978), and FSM-based testing methods are also able to detect them (Masood et al. 2009). Figure 5 illustrates a part of one testing tree generated from four test cases and the $FSM(P)$ in Fig. 4.

By executing this test suite, an RBAC mutant generated from the policy shown in Listing 1 by applying the element modification operator to increment $Dr(r1) = 1$ to $Dr(r1) = 2$ can be detected. The FSM of this variant has state 1111 as reachable and, since test case $t3$ covers the transition $1110 - AC(u2, r1) \rightarrow 1110$, it can detect this fault.

2.6 Test case prioritization

Although very effective, FSM-based testing of RBAC systems tends to generate a large number of test cases regardless the methods used (Damasceno et al. 2016). Thus, development processes of RBAC systems with time and resources constraints may demand improvements on test execution. To cope with this issue, different techniques have been proposed to improve cost-effectiveness of test suites, such as *Test Suite Minimization*, also called test suite reduction, where redundant test cases are permanently removed; and *Test Case Selection*, which selects test cases based on changed parts of a System Under Test (SUT) (Yoo and Harman 2012). These techniques reduce time effort, but they may not work effectively, since they may also omit important test cases able to detect certain faults (Ouriques 2015).

Test Case Prioritization improves test execution without filtering out any test case. It aims at identifying an efficient test execution ordering so that maximum benefits can be obtained, even if test execution is prematurely halted at some arbitrary point (Ouriques 2015). To that, it uses a function f which quantitatively describes the quality of

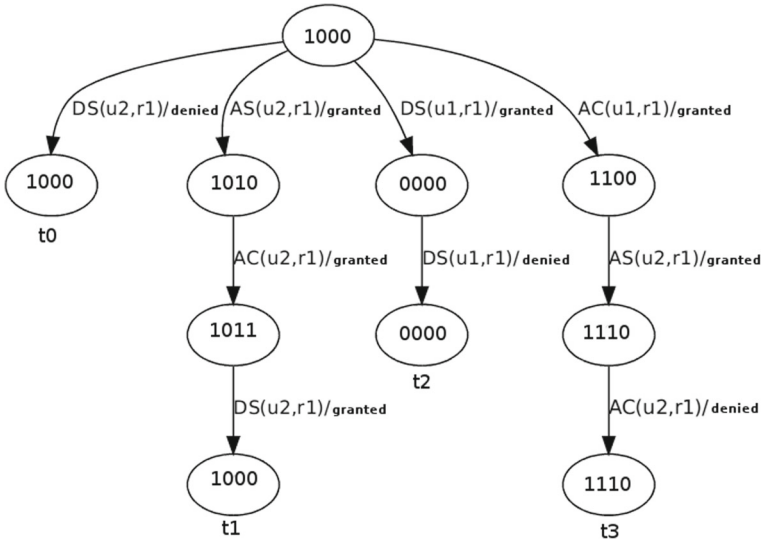

Fig. 5 Testing Tree of an $FSM(P)$

an ordering as *test criteria* (e.g., test effectiveness, code coverage). To illustrate test prioritization, consider an hypothetical SUT with 10 faults and five test cases A, B, C, D, E, as shown in Table 1.

In this example, all faults can be detected by running test cases C and E, since they respectively have 70% and 30% of fault-detection effectiveness. Test case A, on the other hand, can detect only 20% of the faults so it can negatively affect fault detection along test execution if placed at the beginning of a test suite. Thus, it is possible to speed up fault detection during test cases execution by placing C and E at the beginning of the test suite.

After test prioritization, the quality of a ordering can be measured using the Average Percentage Faults Detected (APFD) metric. The APFD is a metric commonly used in test prioritization research (Elbaum et al. 2002), and it is defined as follows:

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

In Eq. 1, the parameter n describes the total number of test cases, l defines the number of faults under consideration and F_i specifies the number of faults detected by a test case i. The APFD value depicts the detection of faults (i.e., test effectiveness) along with test execution given test cases ordering. This value ranges from 0 to 1 and the greater the APFD is, the better is test cases ordering. Table 2 shows the APFD for three prioritized test suites, $T1$, $T2$ and $T3$ obtained from test cases in Table 1. In this example, the APFD points that $T3$ performs better than $T2$ and $T1$.

2.7 Similarity testing

Similarity testing is a promising test case prioritization approach that uses similarity functions to calculate the degree of similarity between pairs of tests and define test ordering (Cartaxo et al. 2011; Bertolino et al. 2015; Coutinho et al. 2014). It is an all-to-all comparison problem (Zhang et al. 2017) and, as most test prioritization algorithms (Elbaum et al. 2002), it has complexity $O(n^2)$. It assumes that resembling test cases are redundant in a sense they cover the same features of an SUT and tend to have equivalent fault detection capabilities (Bertolino et al. 2015).

To run similarity testing, a *similarity matrix* describing the resemblance between all pairs of test cases of a test suite T must be calculated with a similarity function d_x. The similarity matrix SM of a test suite T with n test cases is a matrix where each element

Table 1 Example of test cases with fault-detection capability, taken from Elbaum et al. (2000)

	Fault revealed by test case									
Test case	1	2	3	4	5	6	7	8	9	10
A	•				•					
B	•				•	•	•			
C	•	•	•	•	•	•	•			
D					•					
E								•	•	•

Table 2 APFD value for the test cases example

Test suite id	Test cases ordering	APFD
*T*1	*A*, *B*, *C*, *D*, *E*	0.5
*T*2	*E*, *D*, *C*, *B*, *A*	0.64
*T*3	*C*, *E*, *B*, *A*, *D*	0.84

$SM_{ij} = d_x(t_i, t_j)$ describes the similarity degree between two test cases t_i and t_j, such that $1 \leq i < j \leq n$. In Eq. 2 an illustrative example of similarity matrix is presented.

$$SM = \begin{array}{c} \\ t_1 \\ t_2 \\ \vdots \\ t_{n-1} \\ t_n \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & \cdots & t_{n-1} & t_n \end{array} \\ \begin{bmatrix} 0 & d_x(t_1,t_2) & \cdots & d_x(t_1,t_{n-1}) & d_x(t_1,t_n) \\ 0 & 0 & & & d_x(t_2,t_n) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ & & & 0 & d_x(t_{n-1},t_n) \\ & 0 & \cdots & 0 & 0 \end{bmatrix} \end{array} \tag{2}$$

After calculating the similarity matrix, test ordering is defined based on similarity degrees (Cartaxo et al. 2011; Bertolino et al. 2015; Henard et al. 2014; Coutinho et al. 2014). According to Elbaum et al. (2002), the ordering process can use *total* or *additional* information. Test prioritization based on *total information* uses only pairwise similarity for ordering test cases, whereas *additional information* includes the similarity of previously executed test cases to improve ordering (i.e., the most distinct test case compared to all previous).

Cartaxo et al. (2011) showed that similarity testing can be more effective than random prioritization when applied to test sequences automatically generated from Labelled Transition Systems (LTS) (Cartaxo et al. 2011). In their study, the similarity degree (d_{sd}) between two test cases was calculated as the *number of identical transitions* (*nit*) divided by the *average test case length*. The average length was used to avoid small (large) similarity degrees due to similar short (long) test sequences. An extensive investigation on similarity testing for LTS is found in (Coutinho et al. 2014).

Bertolino et al. (2015) also investigated the application of similarity testing on XACML systems. XACML is an XML-based declarative notation for specifying access control policies and evaluating access requests (OASIS 2013). Essentially, they proposed a test prioritization approach named *XACML similarity* (d_{xs}) which considers three values for test prioritization: (i) a simple similarity (d_{ss}), which describes how much resembling are two test cases (t_i, t_j) based on their lexical distance; (ii) an applicability degree (*AppValue*), which points the percentage of parts of an XACML policy affected by a test case; and (iii) a priority value (*PriorityValue*) which gives weight to pairs of test cases based on their applicability degree. Although investigations have shown that simple similarity d_{ss} is comparable to random prioritization, XACML similarity enabled significant improvements compared to simple similarity and random prioritization.

It should be noticed that the XACML standard can be used to specify and implement RBAC policies (OASIS 2014). However, its current version (OASIS 2014) does not support the specification of SSoD and DSoD constraints. Moreover, since the effectiveness of test criteria is strongly related to its ability to represent specific domain faults (Felderer et al. 2015), there is no guarantee that similarity testing can be as effective on RBAC as they were on XACML and LTS.

3 Similarity testing for RBAC systems

In this section, we introduce our similarity testing approach specific to RBAC systems, named RBAC similarity. The RBAC similarity consists of a similarity testing approach based on Cartaxo et al. (2011) and Bertolino et al. (2015) approaches and suitable for FSM-based testing of RBAC systems. A prioritization algorithm used to perform ordering test cases based on similarity criteria is also discussed.

3.1 RBAC similarity

In XACML similarity, applicability is the relation between an access request and an XACML policy which quantitatively describes the impact of this request (i.e., test case) to the rules of the policy (Bertolino et al. 2015). In our work we extend the concept of XACML applicability to the RBAC domain and propose the *RBAC similarity*, a similarity testing approach specific to RBAC systems.

Essentially, the RBAC similarity (d_{rs}) takes an RBAC policy P and a test suite T generated from an $FSM(P)$ and evaluates the degree of resemblance between all pairs of test cases $t_i, t_j \in T$. To that, it uses a dissimilarity function and the applicability of this pair of test cases to the policy P under test. Given this information, a test case prioritization algorithm performs test ordering from the most distinct and relevant tests to the less diverse and suitable ones. To support similarity testing for RBAC, we proposed the concept of *RBAC applicability* which quantitatively describes the relevance of a test case to one RBAC policy. The dissimilarity function and the RBAC applicability are detailed in the following sections.

3.1.1 Simple dissimilarity:

The simple dissimilarity between test cases is measured based on the number of distinct transitions (ndt). Given two test cases t_i and t_j, the degree of simple dissimilarity (d_{sd}) is calculated as presented in Eq. 3.

$$d_{sd}(t_i, t_j) = \frac{ndt(t_i, t_j)}{avg(length(t_i) + length(t_j))} \tag{3}$$

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 their 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. Listing 2 shows an example of four test cases and their respective transitions and states covered given the $FSM(P)$ previously shown in Fig. 4. The number of distinct transitions, the average length and the simple dissimilarity d_{sd} for each pair of test cases are shown in Table 3.

Table 3 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	2.5	2.0

```
1 1000 − DS(u2, r1)/denied → 1000  //  t0
2 1000 − AS(u2, r1)/granted → 1010 − AC(u2, r1)/granted → 1011 − DS(u2, r1)/granted → 1000  //  t1
3 1000 − DS(u1, r1)/granted → 0000 − DS(u1, r1)/denied → 1000  //  t2
4 1000 − AC(u1, r1)/granted → 1100 − AS(u2, r1)/granted → 1110 − AC(u2, r1)/denied → 1110  //  t3
```

Listing 2 Test cases example

3.1.2 RBAC applicability:

The idea of the RBAC applicability is to quantitatively describe the relevance of a test case to one RBAC policy under test. An RBAC constraint is *applicable to* a test case if there is a match between the users, roles, or permissions of any input of this test case and the attributes of the constraint. For example, if an RBAC policy contains a static cardinality constraint $S_u(u1) = 1$, this constraint must regulate (i.e., *apply* some regulation to) all test cases with user $u1$ as test input (e.g., $AS(u1, r2)$). This idea enables to measure how much a test case t may impact a given policy P, without considering dynamic (behavioral) aspects of the RBAC model (e.g., $FSM(P)$ states/transitions). Thus, it describes the *structural* or *static coverage* of a test case t over one policy P.

However, since RBAC is essentially a reactive system, a behavioral view of a test case is also necessary. In order to satisfy this requirement, we also propose the concept of *behavioral* or *dynamic coverage*. An RBAC constraint of a policy P *reacts to* a test case when this constraint is applicable to any input symbol and it influences on (enforces) the access control decision. As example, the test case $t3$, shown in Fig. 5, depicts a scenario of an RBAC policy containing a dynamic cardinality constraint $D_r(r1) = 1$ and two users $u1$ and $u2$ attempting to activate $r1$. This constraint is applicable (and reacts) to the last input requesting the second role activation of $r1$, and enforces a *denied* response. This information is associated with many transitions of the $FSM(P)$ and used as requirements-based coverage criteria (Utting et al. 2012). Thus, by quantifying the number of RBAC constraints reacting to the inputs of a test case, the dynamic coverage of a policy P can be measured and support test prioritization.

Based on the concepts of static and dynamic coverage, we proposed the RBAC Applicability Degree (AD), which is an array of four values defined as shown in Eq. 4.

$$AD_{P(t)} = \begin{bmatrix} pad_{P(t)} & asad_{P(t)} & acad_{P(t)} & prad_{P(t)} \end{bmatrix} \tag{4}$$

The RBAC Applicability Degree (AD) of a test case t to a given a policy P consists of four values:

- **Policy Applicability Degree** ($pad_{P(t)}$), which shows the ratio of test inputs applicable to any RBAC constraint over the test case length;
- **Assignment Applicability Degree** ($asad_{P(t)}$), which shows the number of RBAC constraints related to assignment faults reacting to t;
- **Activation Applicability Degree** ($acad_{P(t)}$), which shows the number of RBAC constraints related to activation faults reacting to t; and
- **Permission Applicability Degree** ($prad_{P(t)}$), which shows the number of RBAC constraints related to permission faults reacting to t.

The $pad_{P(t)}$ measures how much applicable one test case t is to a given policy based on all RBAC constraints applicable to t. The $asad_{P(t)}$ gives a quantitative information about how many RBAC constraints related to assignment faults (i.e., UR, S_u, S_r, $SSoD$, and S_s) react to t. The $acad_{P(t)}$ gives a quantitative information about how many RBAC

constraints related to activation faults (i.e., $\leq_A$, D_u, D_r, *DSoD*, and D_s) react to t. Finally, the $prad_{P(t)}$ gives a quantitative information about how many RBAC constraints related to permission faults (i.e., *PR*, $\leq_I$) react to t.

Based on the values of *AD*, the *RBAC Applicability Degree* ($RA_{P(t)}$) is calculated. The $RA_{P(t)}$ value is a single quantitative attribute which summarizes the relevance of a single test case t to one policy P by summing the four applicability degrees.

$$RA_{P(t)} = pad_{P(t)} + asad_{P(t)} + acad_{P(t)} + prad_{P(t)} \tag{5}$$

However, since test similarity is calculated for pairs of test cases, we also defined the *RBAC Applicability Value* (*AppValue*) which sums the applicability degrees of test cases (Eq. 6).

$$AppValue(P, t_i, t_j) = RA_{P(t_i)} + RA_{P(t_j)} \tag{6}$$

A priority value (*PriorityValue*) is calculated to weight the pairwise relevance of two test cases. This *PriorityValue* is a constant number α, β, γ, or δ defined based on the $pad_{P(t_i)}$ and $pad_{P(t_j)}$ values. These α, β, γ, and δ constants are defined by the user, such that $\alpha > \beta > \gamma > \delta$. The α is given for pairs of test cases where all test inputs are applicable, and δ is given if none of test inputs are applicable to the constraints of the RBAC policy P. The values 3, 2, 1 and 0 are suggested by Bertolino et al. (2015). Equation 7 shows the formula which derivates the *PriorityValue*

$$PriorityValue(P, t_i, t_j) = \begin{cases} \alpha \text{ if } (pad_{P(t_i)} = pad_{P(t_j)} = 1) \\ \beta \text{ if } (pad_{P(t_i)} \; XOR \; pad_{P(t_j)}) \\ \gamma \text{ if } (0 < pad_{P(t_i)}, pad_{P(t_j)} < 1) \\ \delta \text{ otherwise} \end{cases} \tag{7}$$

The *RBAC Similarity* (d_{rs}) of a pair of test cases consists of the sum of the d_{sd}, *AppValue* and *PriorityValue* values, if $d_{sd}(t_i, t_j) \neq 0$, as shown in Eq. 8. The RBAC similarity was designed based on Bertolino et al. (2015) approach for similarity testing for XACML policies.

$$d_{rs}(P, t_i, t_j) = \begin{cases} 0 & \text{if } d_{sd}(t_i, t_j) = 0 \\ d_{sd}(t_i, t_j) + & \\ AppValue(P, t_i, t_j) + & \\ PriorityValue(P, t_i, t_j) & \text{otherwise} \end{cases} \tag{8}$$

As an example, the applicability degrees of each test case presented in Listing 2, given the RBAC policy in Listing 1, are presented in Table 4.

As shown in Table 4, all test inputs of t_3 are applicable to at least one RBAC constraint and test case t_3 has the greatest RBAC applicability degree. Test case t_2 has the second greatest value, followed by t_1 and t_0 with the same applicability degree. Afterwards, the simple dissimilarity, RBAC application value, and priority value are calculated for all pairs

Table 4 RBAC applicability degree of each test case

Test case 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

of test cases. All these values are joined in the RBAC similarity (d_{rs}) that is calculated for each pair of test cases, as presented in Table 5.

3.2 Test prioritization algorithm

Given the similarity of all pairs of test cases, a test prioritization algorithm has to be used for scheduling test cases execution. The pseudocode of the test prioritization algorithm used in this study is presented in Algorithm 1. Essentially, the test prioritization algorithm iterates a similarity matrix calculated using a similarity function d_x, from the most distinct pairs of test cases to the less dissimilar ones of a test suite S. Given each pairwise similarity, the longest test case is included in the list of prioritized test cases. Otherwise, the shortest is included, if not previously included. This process is performed until all test cases of S are included in L, which stands for the prioritized test suite.

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** $nextMax_{dec}(d_x[\,a][\,b])$
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

Using the RBAC similarity and the test suite shown in Listing 2, the similarity matrix shown in Eq. 9 is obtained.

$$SM = \begin{array}{c} \\ t_0 \\ t_1 \\ t_2 \\ t_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_0 & t_1 & t_2 & t_3 \end{array} \\ \begin{bmatrix} 0 & 4.55 & 5.55 & 7.77 \\ 0 & 0 & 5.55 & 7.77 \\ 0 & 0 & 0 & 8.77 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{9}$$

Using Algorithm 1, the first most dissimilar pair of test cases (t_2, t_3) is selected and the longest test case t_3 is added to L. Afterwards, test case t_0 is included since it is the

Table 5 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

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. The prioritization ends with test case t_2, from pair (t_0, t_2), scheduled at the end of the test execution. Listing 3 shows the L resulting test suite prioritized according to RBAC similarity.

```
1 1000 − AC(u1, r1)/granted → 1100 − AS(u2, r1)/granted → 1110 − AC(u2, r1)/denied → 1110 // t3
2 1000 − DS(u2, r1)/denied → 1000 // t0
3 1000 − AS(u2, r1)/granted → 1010 − AC(u2, r1)/granted → 1011 − DS(u2, r1)/granted → 1000 // t1
4 1000 − DS(u1, r1)/granted → 0000 − DS(u1, r1)/denied → 1000 // t2
```

Listing 3 Test cases example - RBAC similarity

4 Experimental evaluation

According to Damasceno et al. (2016), a larger number of test cases tends to be generated regardless the FSM-based testing methods for RBAC systems. Thus, the higher the number of states and transitions of $FSM(P)$ increase, the greater the test suites are concerning the number of resets, total test suite length, and average test case length. Thus, additional steps become necessary to make software testing more cost-effective.

We proposed RBAC similarity to fill this research gap and designed an experiment to evaluate the cumulative effectiveness and the APFD of the RBAC similarity and compare to simple dissimilarity and random prioritization using test suites generated from FSM-based testing methods on RBAC systems. An schematic overview of this experiment is presented in Fig. 6.

Fifteen test suites were taken from a previous study (Damasceno et al. 2016) where test characteristics (i.e., number of resets, test suite length, and avg. test case length) and effectiveness were analyzed based on the $FSM(P)$ characteristics (i.e., numbers of states, and transitions). These test suites were generated from five RBAC policies specified as $FSM(P)$ models using the RBAC-BT software (Damasceno et al. 2016) and implementations of the W (Chow 1978), HSI (Petrenko and Bochmann 1995), and SPY

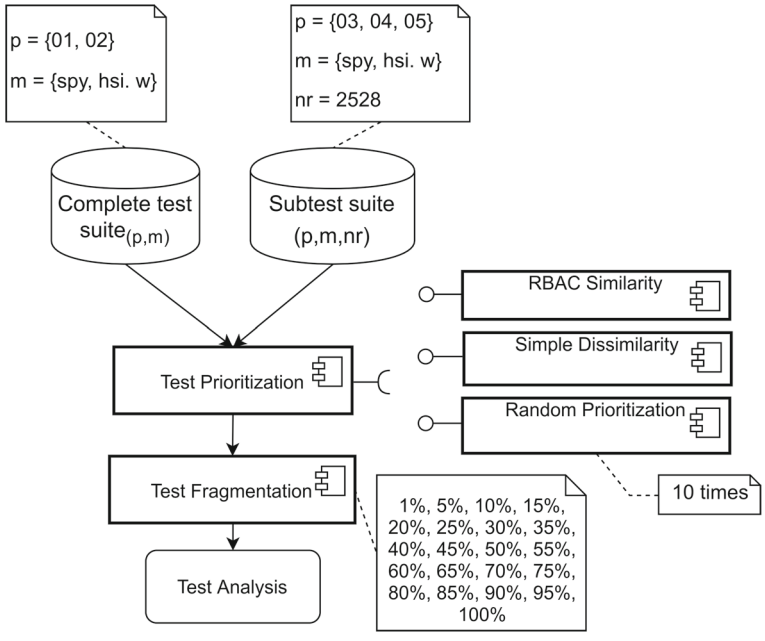

Fig. 6 Comparison of test prioritization techniques - schematic overview

(Simão et al. 2009) methods. Table 6 shows a summary of the five RBAC policies and the total number of RBAC mutants.

The RBAC-BT[1] is an FSM-based testing tool designed by Damasceno et al. (2016) to support FSM-based testing of RBAC systems and the automatic generation of $FSM(P)$ models and RBAC mutants. RBAC-BT was extended to support test prioritization using RBAC similarity and simple dissimilarity. Due to the high number of pairwise comparisons required to perform test prioritization, a time constraint of 24 hours for each test prioritization procedure was defined. Procedures with a duration above this limit were canceled and random subsets of the complete test suites, named as *subtest suite*, were taken for prioritization.

On preliminary experiments, the prioritization of the test suites of policies P03, P04, P05 took more than 24 hours.

Thus, subtest suites of the aforementioned policies containing 2528 test cases were randomly generated 30 times. The number 2528 was taken from the largest complete test suite with test prioritization duration below the 24 hours threshold, the W test suite of policy P02. Table 7 shows the characteristics of the $FSM(P)$ models and their respective complete test suites.

The six complete test suites were prioritized using each test prioritization, and the cumulative effectiveness of these test suites was measured in twenty-one parts. Afterwards, the cumulative effectiveness was used to calculate the APFD of each scenario. The APFD value was calculated using Eq. 1, F_i as the number of faults detected by one test fragment i and l as the number of RBAC mutants. Random prioritization was performed 10 times to the 30 random subtest suites of P03, P04 and P05.

Using the R statistical package, we calculated mean APFD with confidence interval (CI) of 95% to all test scenarios and performed the nonparametric Wilcoxon matched-pairs signed ranks test to verify if the RBAC similarity reached different APFDs compared to simple dissimilarity and random prioritization with a confidence interval of 95%. As the alternative hypothesis, we considered that RBAC similarity performed better (i.e., greater mean cumulative effectiveness) than the other criteria.

To complement hypothesis tests, we analyzed the effect size by computing unstandardized (i.e., median and mean differences) and standardized measures (i.e., Cohen's d Hedges g (Kampenes et al. 2007) and Vargha-Delaney's $\hat{A}_{12}$ (Arcuri and Briand 2011)) using R and the `effsize` package (Torchiano 2017).

4.1 Analysis of the complete test suites

In this section, we discuss the results of the experiments comparing RBAC similarity, simple and random prioritization based on complete test suites. The mean cumulative effectiveness for P01 and P02 are respectively shown in Tables 8 and 9, and Figs. 7 and 8

Table 6 RBAC policies characteristics

ID	RBAC policy name	U	R	S_u	D_u	S_r	D_r	$SSoD$	$DSoD$	Mutants
P01	01_Masood2010Example1	2	1	•	•	•	•			9
P02	02_SeniorTraineeDoctor	2	2	•	•	•	•	•		17
P03	03_ExperiencePointsv2	2	4			•		•	•	11
P04	04_users11roles2v2	11	2	•		•		•		28
P05	05_Masood2009P2v2	2	5	•	•		•	•	•	48

Table 7 *FSM*(*P*) and test characteristics

	FSM characteristics		Test suite length			Number of resets			Avg. test case length		
ID	States	Transitions	W	HSI	SPY	W	HSI	SPY	W	HSI	SPY
P01	8	64	1240	753	542	285	176	93	3.350	3.278	4.827
P02	21	336 n	14704	8238	5841	2528	1408	751	4.816	4.850	6.777
P03	203	6496	776074	333550	213799	119586	51451	24001	5.489	5.482	7.907
P04	485	42680	13125662	6085633	2392981	2236388	993492	138766	4.869	5.125	16.24
P05	857	34280	7086325	2970528	1735818	835600	353836	159463	7.480	7.395	9.885

Table 8 Cumulative effectiveness of the P01 complete test suites

	Percent	1	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
P01 + W	Simple	0.00	0.50	0.50	0.75	0.75	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RBAC	0.75	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Random	0.57	0.91	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P01 + HSI	Simple	0.00	0.00	0.50	0.50	0.50	0.50	0.50	0.50	0.75	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RBAC	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Random	0.26	0.78	0.87	0.93	0.97	0.98	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P01 + SPY	Simple	0.0	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	RBAC	0.00	0.75	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Random	0.033	0.750	0.917	0.942	0.958	0.975	0.992	0.992	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table 9 Cumulative effectiveness of the P02 complete test suites

	Percent	1	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
P02 + W	Simple	0.29	0.43	0.71	0.71	0.71	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RBAC	0.86	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Random	0.79	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P02 + HSI	Simple	0.14	0.57	0.57	0.71	0.71	0.71	0.71	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86	1.00	1.00	1.00	1.00	1.00
	RBAC	0.43	0.71	0.71	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Random	0.73	0.95	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P02 + SPY	Simple	0.57	0.57	0.71	0.71	0.71	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86	1.00	1.00	1.00	1.00	1.00	1.00
	RBAC	0.86	0.86	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Random	0.71	0.94	0.97	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

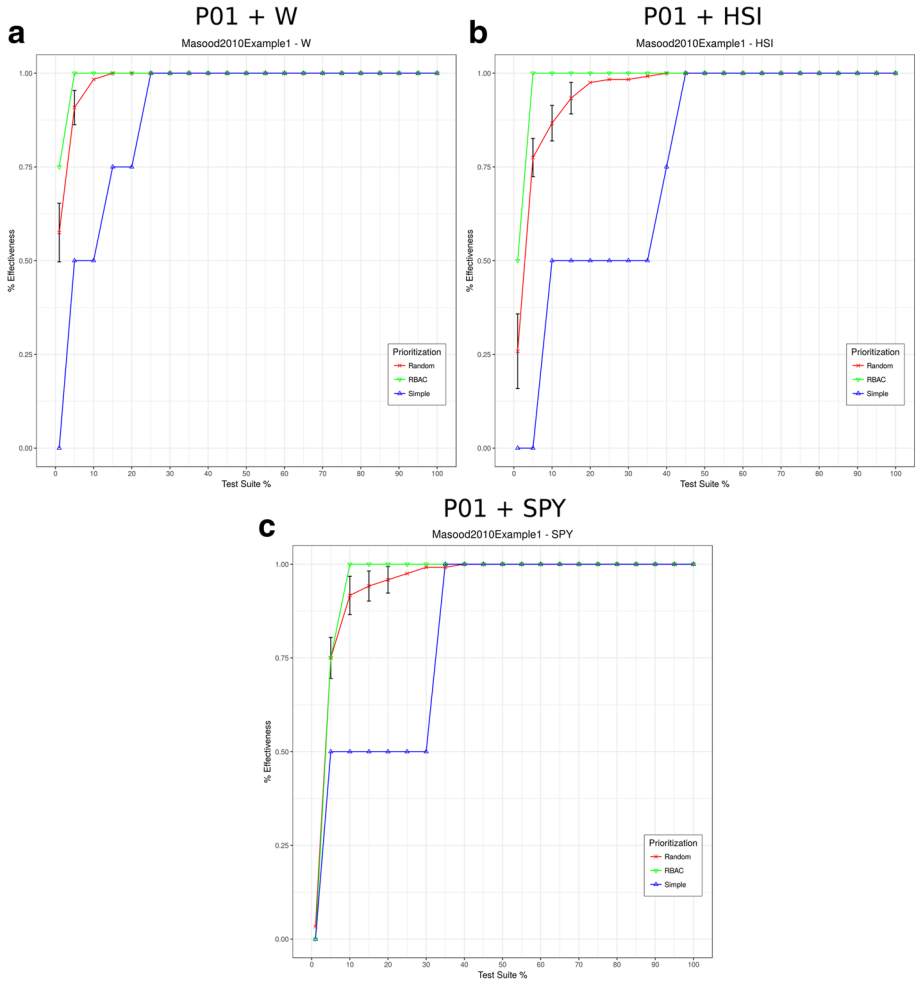

Fig. 7 Cumulative effectiveness for P01 with error bars (CI=95%). **a** P01 + W. **b** P01 + HSI. **c** P01 + SPY

with error bars calculated with a confidence interval of 95%. At the end of this section, we also show the mean APFD and the results of the Wilcoxon matched-pairs signed ranks test.

In most of the cases, there was no statistically significant difference between the prioritization algorithms in the *P01* and *P02* scenarios. The *P01 + HSI* scenario was the only exception where RBAC similarity reached an APFD higher than simple dissimilarity and random prioritization. In the five remaining scenarios, RBAC similarity performed without significant difference compared to at least one of the methods. The mean APFD for each scenario are shown in Table 10 with their respective confidence intervals of 95% subscripted.

Table 11 shows the results of the Wilcoxon matched-pairs signed ranks test using a confidence interval of 95% to the mean cumulative effectiveness. In this case, we compared RBAC similarity to simple and random prioritization and random prioritization to simple dissimilarity. Significant results are highlighted in **bold**.

Table 11 corroborates to the finding of Fig. 7 and Table 10 where RBAC similarity had a statistically significant difference compared to the other criteria in *P01 + HSI* scenario;

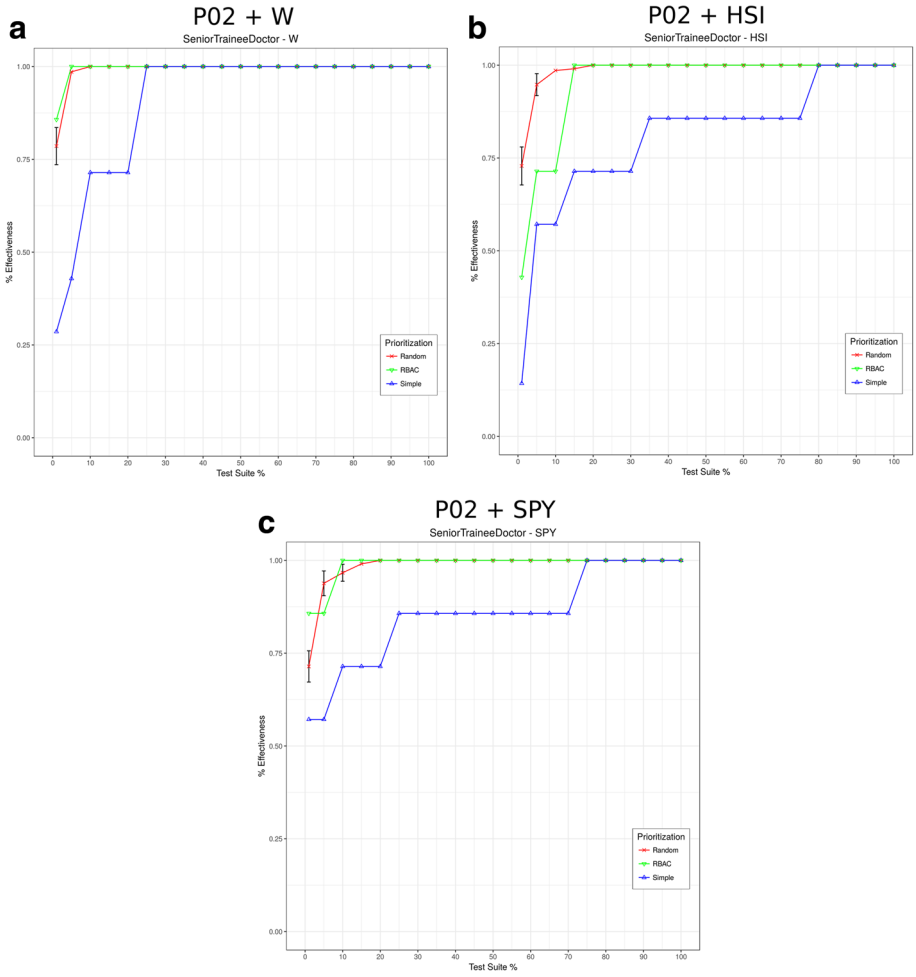

Fig. 8 Cumulative effectiveness for P02 with error bars (CI=95%). **a** P02 + W. **b** P02 + HSI. **c** P02 + SPY

and random prioritization reached significantly different APFDs compared to simple dissimilarity in the all scenarios.

4.2 Analysis of the subtest suites

Since test prioritization for P03, P04 and P05 was too expensive, we considered 30 random subtest suites with 2528 test cases. Random prioritization was run 10 times for each of the 30 subtest suites.

Table 10 Mean APFD of the complete test suites with confidence interval of 95%

Scenario	RBAC	Simple	Random
P01 + W	0.964	0.857	$0.951_{\pm 0.00518}$
P01 + HSI	**0.952**	0.726	$0.917_{\pm 0.00933}$
P01 + SPY	0.917	0.786	$0.908_{\pm 0.00814}$
P02 + W	0.969	0.874	$0.965_{\pm 0.00254}$
P02 + HSI	0.922	0.779	$0.960_{\pm 0.00369}$
P02 + SPY	0.963	0.827	$0.958_{\pm 0.00382}$

Statistically significant, or the highest values given the confidence interval of 95% are captured in bold

Table 11 Wilcoxon matched-pairs signed ranks test (CI=95%) for P01 and P02

	P01			P02		
Hypothesis	W	HSI	SPY	W	HSI	SPY
RBAC > Simple	**0.0284**	**0.0030**	**0.0131**	**0.0267**	**0.0001**	**0.0002**
RBAC > Random	0.0907	**0.0071**	0.0538	0.1855	0.9498	0.2919
Random > Simple	**0.0290**	**0.0045**	**0.0104**	**0.0272**	**0.0002**	**0.0002**

Statistically significant, or the highest values given the confidence interval of 95% are captured in bold

The mean cumulative effectiveness of P03, P04, and P05 are respectively presented in Tables 12, 13, and 14. Figs. 9, 10, and 11 show the mean cumulative effectiveness with error bars calculated using a confidence interval of 95%.

In the P03 test scenarios, the first 5 to 10% of the W, HSI, and SPY subtest suites (i.e., a subset of 125 to 250 test cases) became sufficient to reach the maximum effectiveness. All test prioritization approaches presented similar results and no statistical significance was found between RBAC and the other approaches. In scenarios like this, test minimization techniques may be more cost-effective than test prioritization due to its $O(n^2)$ complexity.

In the P04 scenario, the benefits of RBAC similarity started to become more visible and statistically significant, as shown in Fig. 10 and Table 13. There was one exception where no significant difference was obtained. In the *P04 + W* scenario, the W method generated an extremely large test suite and, to enable test prioritization, we selected random subtest suites containing 2528 test cases. This random selection may have reduced test diversity. In the other scenarios, *P04 + HSI* and *P04 + SPY*, we found that the cumulative effectiveness of the RBAC similarity had a statistically significant difference compared to the other methods.

The mean cumulative effectiveness for the P05 test scenarios are presented in Fig. 11 and Table 14. In the P05 scenario, RBAC similarity, simple dissimilarity, and random prioritization clearly had statistically different cumulative effectivenesses. Respectively, 65% of the W and HSI, and 80% of the SPY subtest suites prioritized using RBAC similarity became capable of reaching the highest effectivenesses. RBAC similarity presented a significantly greater cumulative effectiveness compared to random prioritization and simple dissimilarity.

To the P03, P04 and P05, we also calculated the mean APFD based on the cumulative effectiveness of all runs of the 30 random subtest suites. The mean APFD of each test scenario with confidence interval of 95% is shown in Table 15. The highest APFD values are highlighted in **bold**.

In P03 scenario, the fault distribution along the $FSM(P03)$ may have benefited fault detection and all methods performed similarly. In P04 scenario, there was only one case where RBAC similarity did not work well and no statistically significant difference was found (i.e., *P04 + W*). Regarding simple dissimilarity, it did not reach an APFD higher than random prioritization. At last, in all P05 scenarios, we found statistically significant differences between RBAC, simple and random prioritization. Table 16 shows the results of the Wilcoxon matched-pairs signed ranks test in the test scenarios of policies P03, P04 and P05. Significant results are highlighted in **bold**.

The analysis of the mean APFD and the confidence intervals of the subtest suites indicated that RBAC similarity performed better than simple dissimilarity and random prioritization in some scenarios. In addition to assessing whether an algorithm performs

Table 12 Cumulative effectiveness of the P03 subtest suites

	Percent	1	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
P03 + W	Simple	0.87	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RBAC	0.95	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Random	0.88	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P03 + HSI	Simple	0.82	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RBAC	0.82	0.85	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Random	0.83	0.97	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P03 + SPY	Simple	0.86	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RBAC	0.97	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Random	0.89	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 13 Cumulative effectiveness of the P04 subtest suites

	Percent	1	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
P04 + W	Simple	0.58	0.58	0.60	0.61	0.61	0.62	0.63	0.64	0.64	0.65	0.66	0.66	0.67	0.68	0.68	0.68	0.69	0.69	0.70	0.70	0.71
	RBAC	0.58	0.59	0.59	0.59	0.63	0.63	0.63	0.63	0.63	0.63	0.63	0.64	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.71
	Random	0.57	0.59	0.59	0.60	0.61	0.62	0.62	0.63	0.64	0.64	0.65	0.66	0.67	0.67	0.68	0.68	0.69	0.69	0.70	0.70	0.71
P04 + HSI	Simple	0.58	0.59	0.61	0.62	0.63	0.65	0.65	0.66	0.67	0.68	0.70	0.71	0.71	0.72	0.73	0.74	0.75	0.76	0.77	0.77	0.78
	RBAC	0.58	0.58	0.61	0.61	0.61	0.61	0.62	0.76	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.78
	Random	0.57	0.59	0.60	0.62	0.63	0.64	0.65	0.66	0.68	0.69	0.70	0.71	0.72	0.73	0.73	0.74	0.75	0.76	0.76	0.77	0.78
P04 + SPY	Simple	0.58	0.62	0.66	0.69	0.72	0.74	0.76	0.79	0.80	0.82	0.84	0.85	0.87	0.88	0.90	0.90	0.91	0.93	0.95	0.98	0.99
	RBAC	0.63	0.72	0.87	0.93	0.97	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
	Random	0.59	0.65	0.71	0.75	0.79	0.82	0.85	0.87	0.90	0.91	0.92	0.94	0.95	0.96	0.96	0.97	0.98	0.98	0.98	0.99	0.99

Table 14 Cumulative effectiveness of the P05 subtest suites

	Percent	1	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
P05 + W	Simple	0.62	0.72	0.75	0.76	0.77	0.79	0.81	0.82	0.82	0.82	0.83	0.83	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.85	0.85
	RBAC	0.69	0.77	0.80	0.81	0.83	0.83	0.83	0.83	0.84	0.84	0.84	0.84	0.84	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85
	Random	0.62	0.73	0.76	0.79	0.80	0.81	0.82	0.83	0.83	0.83	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.85	0.85	0.85
P05 + HSI	Simple	0.60	0.71	0.73	0.74	0.75	0.76	0.77	0.78	0.79	0.80	0.81	0.82	0.83	0.83	0.83	0.83	0.84	0.84	0.84	0.84	0.84
	RBAC	0.66	0.75	0.77	0.79	0.80	0.81	0.81	0.82	0.83	0.83	0.83	0.83	0.83	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84
	Random	0.61	0.71	0.74	0.75	0.77	0.78	0.79	0.80	0.80	0.81	0.81	0.82	0.82	0.83	0.83	0.83	0.83	0.84	0.84	0.84	0.84
P05 + SPY	Simple	0.65	0.73	0.75	0.76	0.78	0.80	0.82	0.82	0.83	0.83	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.85	0.85
	RBAC	0.76	0.80	0.83	0.83	0.83	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.85	0.85	0.85	0.85	0.85
	Random	0.64	0.74	0.77	0.79	0.81	0.82	0.82	0.83	0.83	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.85	0.85	0.85	0.85	0.85

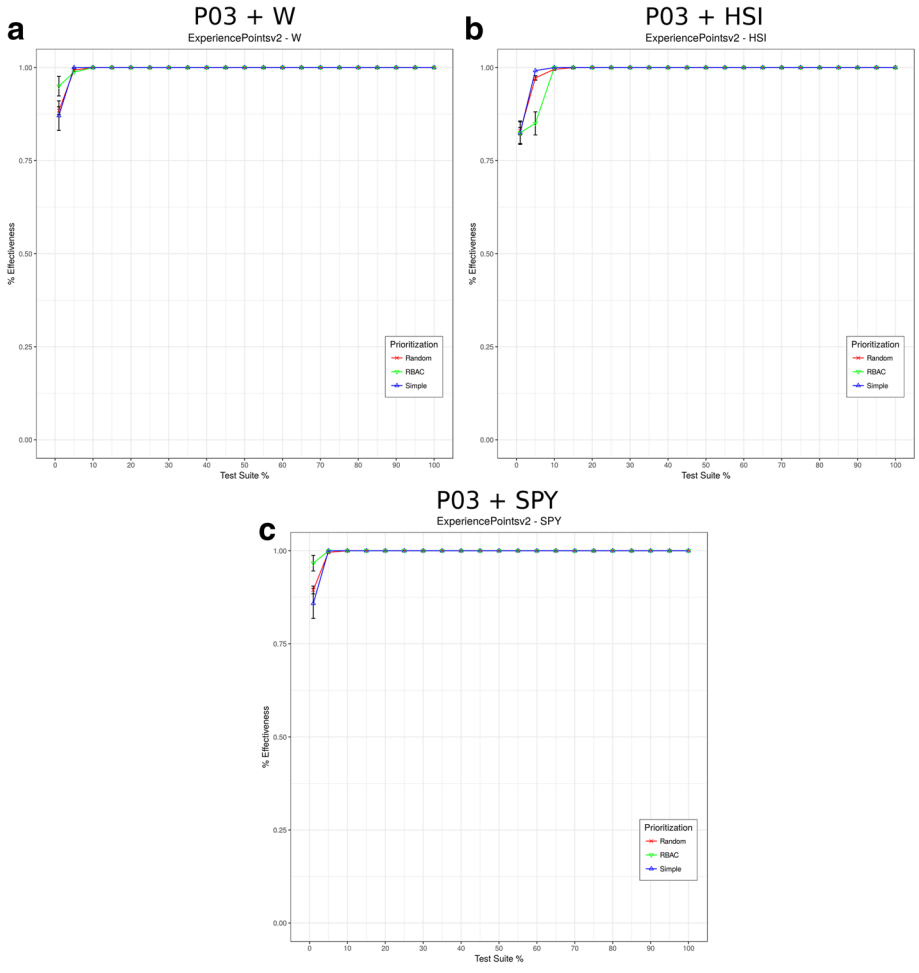

Fig. 9 Cumulative effectiveness for P03 with error bars (CI=95%). **a** P03 + W. **b** P03 + HSI. **c** P03 + SPY

statistically better than another, it is crucial to measure the magnitude of such improvement. To analyze such aspect, effect size measures are required (Kampenes et al. 2007; Arcuri and Briand 2011; Wohlin et al. 2012).

4.2.1 Effect size to subtest suites

Effect size measures allow for quantifying the difference (i.e., magnitude of the improvement) between two groups (Wohlin et al. 2012). Kampenes et al. (2007) found that only 29% of software engineering experiments report some effect size measure. Thus, to improve our analysis, we also evaluated the effect that one test prioritization method had on the APFD compared with the other methods.

There are two main classes of effect size: (i) unstandardized, which are dependent from the unit of measurement; and (ii) standardized, which are independent from the evaluation criteria measurement units. For each pair of different prioritization method, we computed five different measures: two unstandardized (i) mean and (ii) median differences; and three standardized (iii) Cohen's d (Cohen 1977), (iv) Hedges' g (Hedges 1981), and (v) Vargha-Delaney's $\hat{A}_{12}$ (Vargha and Delaney 2000).

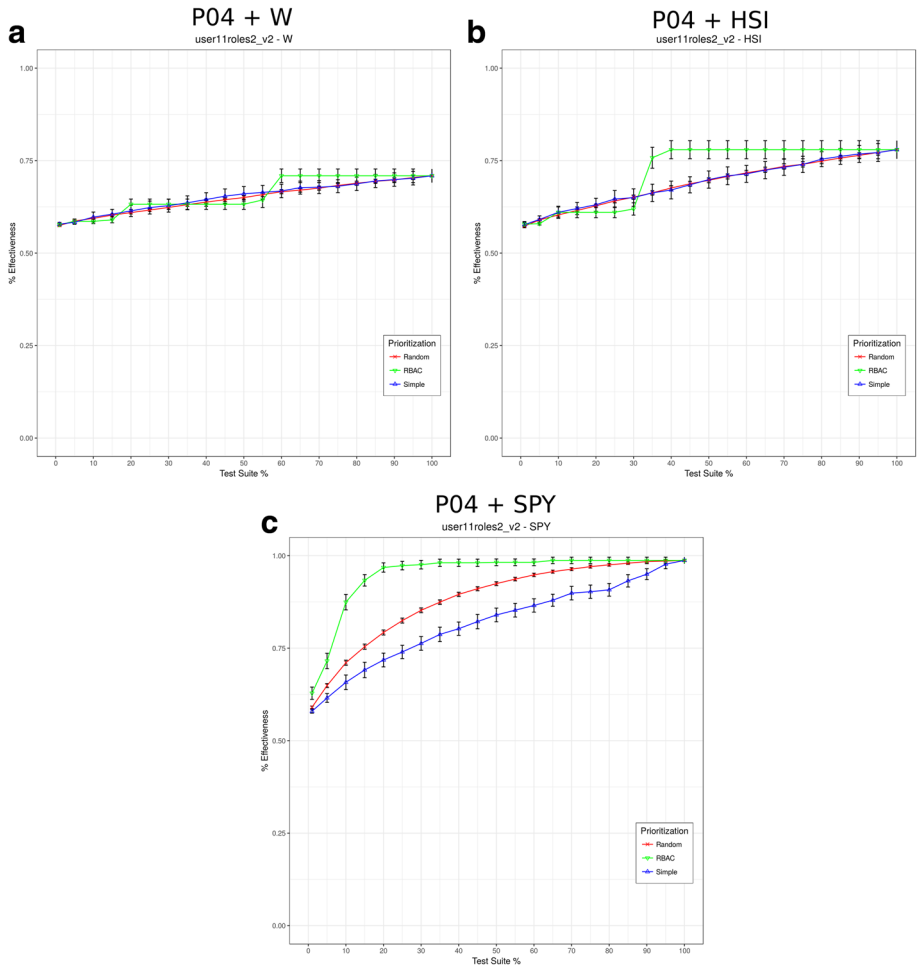

Fig. 10 Cumulative effectiveness for P04 with error bars (CI=95%). **a** P04 + W. **b** P04 + HSI. **c** P04 + SPY

Mean and median differences, Cohen's *d*, and Hedges' *g* are presented as often referred metrics in the software engineering literature (Kampenes et al. 2007). Cohen's *d*, and Hedges' *g* are computed based on the mean difference and an estimate of population standard deviation σ_{pop} and compared using standard conventions (Cohen 1992).

Vargha-Delaney (VD) $\hat{A}_{12}$ is an effect size measure based on stochastic superiority that denotes the probability of a method outperform another (Vargha and Delaney 2000). If both methods are equivalent then $\hat{A}_{12} = 0.5$. An effect size $\hat{A}_{12} > 0.5$ means that the treatment method has higher probability of achieving a better performance than the control method, otherwise vice-versa. Vargha-Delaney's $\hat{A}_{12}$ is recommended by Arcuri and Briand (2011) as a simple and intuitive measure of effect size for assessing randomized algorithms in software engineering research. Table 17 shows the pairwise comparison of the three test prioritization methods. The metrics presented can also be used in future research (e.g., meta-analysis (Kampenes et al. 2007)).

We did not compute the effect size to *P01* and *P02* due to the deterministic nature of RBAC and simple prioritizations and its consequent $\sigma_{pop} = 0$. The analysis of effect size corroborated to the mean APFDs and Wilcoxon matched-pairs signed ranks tests and RBAC similarity had good results in *P04+HSI*, *P04+SPY*, and all *P05* scenarios.

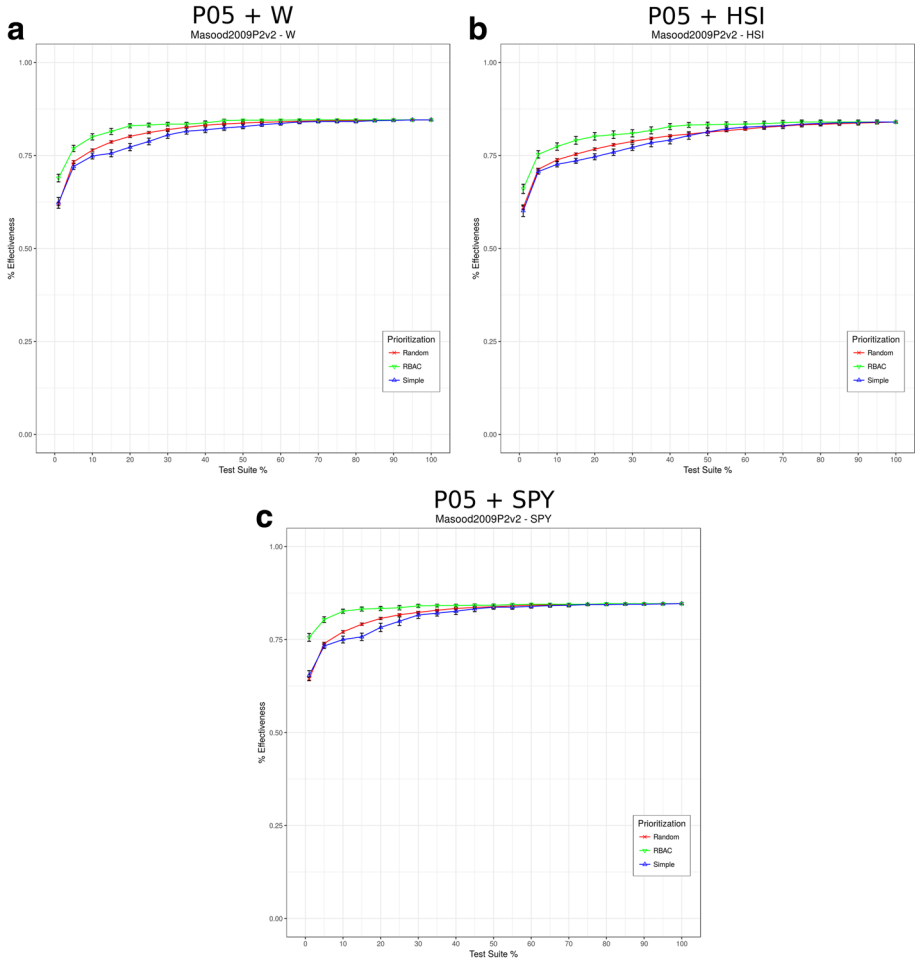

Fig. 11 Cumulative effectiveness for P05 with error bars (CI=95%). **a** P05 + W. **b** P05 + HSI. **c** P05 + SPY

We found differences of medium magnitude between RBAC compared with simple and random prioritizations in *P04+HSI*; and large magnitude in *P04+SPY* and all *P05* scenarios. There was only one case (i.e., *P03+HSI*) where RBAC prioritization did not outrun the other methods. In the other scenarios, we found negligible to medium differences between the techniques. Thus, the following order was observed, from the method with the lowest to the highest APFDs, $Simple \prec Random \prec RBAC$.

Table 15 Mean APFD of the subtest suites with confidence interval of 95%

Scenario	$APFD_{RBAC}$	$APFD_{Simple}$	$APFD_{Random}$
P03 + W	$0.9732_{\pm 0.0016}$	$0.9700_{\pm 0.0019}$	$0.9704_{\pm 0.0005}$
P03 + HSI	$0.9607_{\pm 0.0027}$	$0.9675_{\pm 0.0016}$	$0.9665_{\pm 0.0007}$
P03 + SPY	$0.9746_{\pm 0.0010}$	$0.9694_{\pm 0.0019}$	$0.9710_{\pm 0.0005}$
P04 + W	$0.6466_{\pm 0.0110}$	$0.6417_{\pm 0.0129}$	$0.6385_{\pm 0.0036}$
P04 + HSI	$\mathbf{0.7062}_{\pm 0.0164}$	$0.6770_{\pm 0.0166}$	$0.6757_{\pm 0.0047}$
P04 + SPY	$\mathbf{0.9222}_{\pm 0.0073}$	$0.7943_{\pm 0.0110}$	$0.8561_{\pm 0.0033}$
P05 + W	$\mathbf{0.8113}_{\pm 0.0025}$	$0.7888_{\pm 0.0032}$	$0.7977_{\pm 0.0011}$
P05 + HSI	$\mathbf{0.7973}_{\pm 0.0055}$	$0.7729_{\pm 0.0052}$	$0.7779_{\pm 0.0019}$
P05 + SPY	$\mathbf{0.8192}_{\pm 0.0027}$	$0.7948_{\pm 0.0039}$	$0.8012_{\pm 0.0012}$

Statistically significant, or the highest values given the confidence interval of 95% are captured in bold

Table 16 Wilcoxon matched-pairs signed ranks test (CI=95%) for P03,P04 and P05

	P03			P04			P05		
Hypothesis	W	HSI	SPY	W	HSI	SPY	W	HSI	SPY
RBAC > Simple	0.500000	0.977250	0.50	0.147876	**0.007448**	**0.000048**	**0.000071**	**0.000071**	**0.000048**
RBAC > Random	0.500000	0.708059	0.185547	**0.035099**	**0.004014**	**0.000048**	**0.000048**	**0.000048**	**0.000048**
Random > Simple	0.500000	0.899379	0.500000	0.999147	0.948464	**0.000048**	**0.000182**	**0.038077**	**0.000477**

Statistically significant, or the highest values given the confidence interval of 95% are captured in bold

Table 17 Pairwise comparison among the methods with respect to the APFD

		P03			P04			P05		
Alt. Hypothesis	Info	W	HSI	SPY	W	HSI	SPY	W	HSI	SPY
RBAC > Random	Mean diff	0,0029	-0,0058	0,0036	0,0081	0,0305	0,0662	0,0137	0,0195	0,018
	Median diff	0,006	-0,006	0,006	0,011	0,0174	0,0696	0,0119	0,0181	0,0192
	Cohen's d	0,6508	-0,8075	1,2479	0,2722	0,6991	3,2109	1,942	1,3114	2,3887
	Hedges' g	0,6493	-0,8056	1,245	0,2716	0,6975	3,2036	1,9376	1,3084	2,3832
	VD's $\hat{A}_{12}$	0,3311	0,733	0,2809	0,4164	0,3073	0,0283	0,1202	0,1794	0,0739
RBAC > Simple	Mean diff	0,0032	-0,0067	0,0052	0,0049	0,0292	0,1279	0,0226	0,0244	0,0244
	Median diff	0,006	0	0,006	0,0055	0,0211	0,1346	0,0238	0,0197	0,0264
	Cohen's d	0,6726	-1,1269	1,2625	0,1521	0,6615	5,0993	2,9639	1,7059	2,7021
	Hedges' g	0,6639	-1,1122	1,2461	0,1501	0,6529	5,033	2,9254	1,6837	2,667
	VD's $\hat{A}_{12}$	0,6772	0,2267	0,79	0,5689	0,6933	1	0,9778	0,9067	0,9783
Random > Simple	Mean diff	0,0003	-0,001	0,0015	-0,0032	-0,0013	0,0617	0,0089	0,0049	0,0064
	Median diff	0	0,006	0	-0,0055	0,0037	0,065	0,0119	0,0016	0,0072
	Cohen's d	0,0662	-0,1698	0,3239	-0,0997	-0,0314	2,1523	0,9026	0,3052	0,6204
	Hedges' g	0,0661	-0,1694	0,3232	-0,0995	-0,0313	2,1474	0,9005	0,3045	0,619
	VD's $\hat{A}_{12}$	0,5181	0,4761	0,5825	0,4752	0,5146	0,9277	0,7554	0,5868	0,677

5 Discussion

Recently, Cartaxo et al. (2011) and Bertolino et al. (2015) showed that similarity functions can be helpful when it is necessary to prioritize exhaustive test suites automatically generated for LTS models and XACML policies, respectively. In our previous study (Damasceno et al. 2016), we found that, no matter what FSM-based testing methods are applied to RBAC systems, when the number of users and roles increase, larger test suites tend to be generated. Thus, specific domain test criteria are required to optimize FSM-based testing for RBAC systems. To this end, there are three main approaches: (i) Test minimization, (ii) Test selection, and (iii) Test prioritization.

Unlike (i) test minimization and (ii) test selection, that may compromise fault detection capability; (iii) test prioritization aims at finding an order of execution to an entire test suite (i.e., without filtering out any test case) based on some test criteria (Yoo and Harman 2012). In this paper, we investigated the test prioritization for RBAC systems, and we proposed the RBAC similarity.

5.1 RBAC similarity compared to the other criteria

Our results showed that RBAC similarity performed better than simple dissimilarity and random prioritization in some of the scenarios, especially those with large $FSM(P)$ models. To policies *P01* and *P02*, we did not find statistically significant differences between the test prioritization criteria in most of the scenarios. The only exception was to *P01* + *HSI*, where a statistically significant difference between RBAC similarity and the other criteria was found. The HSI method reduces test dimensions by using harmonized state identifiers instead of the characterization set (Petrenko and Bochmann 1995). In this scenario, the characteristics of the HSI may have affected test diversity and, as a result, benefited RBAC similarity.

Due to the large number of test cases generated from policies *P03*, *P04*, and *P05*, prioritizing the complete test suites became infeasible. To overcome this issue, we opted to apply test prioritization on random subtest suites.

To policy *P03*, all test prioritization approaches increased the cumulative effectiveness to the maximum value yet at the first 5 to 10% and we did not find statistically significant differences between them. Thus, the fault distribution along the $FSM(P03)$ model benefited fault detection and test prioritization. In scenarios like this, test minimization may be more suitable than test prioritization, which has an $O(n^2)$ cost. However, as we highlighted earlier, there is a risk of reducing the capability of test suites detecting faults out of the RBAC domain.

The benefits of the RBAC similarity became more evident in P04 and P05 scenarios, the largest $FSM(P)$ models. In policy *P04*, we found a statistically significant difference between RBAC similarity to the subtest suites generated from HSI and SPY. The only exception was the *P04* + *W* scenario where the random selection of subtest suites may have compromised test diversity.

In the *P05* scenario, the RBAC similarity outperformed both test prioritization criteria with statistically significant differences. The analysis of the mean APFD values and effect size corroborate to the mean cumulative effectivenesses depicted in Figs. 9 to 11.

5.2 Random prioritization *vs.* simple dissimilarity

Our results showed a statistically significant difference between random prioritization and simple dissimilarity. In ten out of 15 scenarios, random prioritization presented APFD significantly different and higher than simple dissimilarity. RBAC faults can be exhibited across many different transitions of $FSM(P)$ (Masood et al. 2010). Thus, test diversity may not imply on higher APFD.

5.3 Practical feasibility

We found that RBAC similarity may not be feasible to large complete test suites, as seen in scenarios *P03*, *P04* and *P05*. The $O(n^2)$ complexity is an inherent characteristic of most test prioritization approaches (Elbaum et al. 2002), especially similarity testing, that is also an all-to-all comparison problem (Zhang et al. 2017). However, RBAC similarity can still be improved through (i) test minimization and/or (ii) parallel programming.

The RBAC applicability can be used in test minimization as requirements coverage criteria to find test cases relevant to the constraints (i.e., requirements) of RBAC policies. Afterwards, RBAC similarity can be applied as we proposed. Thus, a significant test cost reduction can be achieved, but at the risk of reducing the fault-detection capability (Yoo and Harman 2012).

Recent studies have proposed parallel algorithms to efficiently calculate similarity matrices for mathematical modelling of heterogeneous hardware (Rawald et al. 2015) and ontology mapping (Gîză-Belciug and Pentiuc 2015). However, to the best of our knowledge, they have never been investigated for similarity testing. RBAC similarity as a test minimization criterion and parallel algorithms to calculate similarity matrices for test prioritization could boost up similarity testing but this is out of the scope of this study and left as future work.

6 Threats to validity

Conclusion Validity: Threats to conclusion validity relate with the ability draw correct conclusions about the relation between the treatment and the outcomes. To mitigate this, we used the Wilcoxon matched-pairs signed ranks test to verify if the RBAC similarity reached different APFDs compared to simple dissimilarity and random prioritization with a confidence interval of 95%. We also computed the mean APFD with a confidence interval of 95% and five effect size measures to quantify the difference between the methods. The statistical analysis were performed using the R statistical package and the `effsize` package (Torchiano 2017). The R scripts, input and output statistical data are included in the RBAC-BT repository.

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. To mitigate this threat, random tasks (i.e., subtest suite generation and random prioritization) were repeatedly performed to avoid results obtained by chance. Most of artifacts used in this work were reused from the lab package of our previous study (Damasceno et al. 2016).

Construct Validity: Construct validity concerns with generalizing outcomes to the concept or theory behind the experiment. We used first-order mutants from the RBAC fault domain (Masood et al. 2009) to simulate simple faults and evaluate the effectiveness of each prioritization criteria. Mutation analysis is a common assessment approach

of software testing investigations (Jia and Harman 2011). Other RBAC fault models could be used in this experiment, such as malicious faults (Masood et al. 2009) and probabilistic models of fault coverage (Masood et al. 2010). These fault models could be used to analyse RBAC similarity testing from a perspective of faults of different nature, but they were left as future work. Moreover, despite the relatively low number of faults, the RBAC fault model is still representative to functional faults of RBAC systems (Masood et al. 2009).

External Validity: It concerns with the generalization of the outcomes to other scenarios. To mitigate this threat, we included test suites generated from three different test generation methods and RBAC policies with different characteristics.

7 Conclusions

Essentially, the RBAC model reduces the complexity of security management routines by grouping privileges through roles which can be assigned to users and activated in sessions. Access control testing is one important activity during the development of RBAC systems since implementation mistakes may lead to security breaches. In this context, previous studies have shown that FSM-based testing can be effective at detecting RBAC faults, but very expensive. Thus, additional steps become necessary to make RBAC testing more feasible and less costly.

Test case prioritization comes as a solution to this problem and it aims at finding an ordering for test cases execution to maximize some test criteria. Similarity testing is a variant of test case prioritization which has been investigated under the XACML and LTS domains and enabled to find better orders for test cases execution. In this paper we introduce a test prioritization technique named *RBAC similarity* which uses the dissimilarity between pairs of test cases and their pairwise applicability to the RBAC policy under test (i.e., the relevance of these test cases to the RBAC constraints) as test prioritization criteria.

Our *RBAC similarity* approach was experimentally evaluated and compared with simple dissimilarity and random prioritization as baselines. The obtained results pointed out that RBAC similarity improved the mean cumulative effectiveness and the APFD and enable to reach the maximum effectiveness of the test suites at a faster rate with significant difference in most of the cases. In some scenarios, prioritizing HSI and SPY test suites with RBAC similarity resulted on better APFD values than applying the technique to W test suites. The characteristics of the test cases generated from HSI and SPY favoured the similarity testing algorithms while random selection applied to complete test suites generated from W negatively impacted test prioritization using similarity functions. Moreover, random prioritization also outperformed simple dissimilarity in most of the cases. We analyze our data using Wilcoxon matched-pairs signed ranks test and error bars with CI=95%, and five effect size metrics (i.e., mean and median differences, Cohen's *d*, Hedges' *g* and Vargha-Delaney's $\hat{A}_{12}$) and found statistically significant in some scenarios.

All test artifacts (i.e., RBAC-BT tool, test suites, test results, RBAC policies, and statistical data) are available online [2] and can be used to replicate, verify and validate this experiment. As future work, we want to investigate alternative algorithms for ordering test cases, such as algorithms using total information for *test prioritization*, other fault models, such as simulated malicious faults and probabilistic fault models. We also intend to investigate the usage of RBAC similarity as a requirements coverage criterion for test minimization and as a fitness function in search-based software testing (McMinn 2004).

Endnotes

[1] https://github.com/damascenodiego/rbac-bt/

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

Acknowledgements
We acknowledge the help from all the LabES's members (Software Engineering Laboratory) at the University of Sao Paulo (USP) for their valuable comments. We also thank the reviewers for all valuable comments and suggestions to this study.

Funding
Carlos Diego Nascimento Damasceno's research project was supported by the National Council for Scientific and Technological Development (CNPq), process number 132249/2014-6.

Authors' contributions
CDND designed and conducted the experiment, adapted the RBAC-BT tool and analyzed the results. PCM and AS supported the validation of the experiment protocol and analysis of results. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests

Publisher's Note

Received: 2 March 2017 Accepted: 20 December 2017
Published online: 17 January 2018

References

Andrews JH, Briand LC, Labiche Y, Namin AS (2006) Using mutation analysis for assessing and comparing testing coverage criteria. IEEE Trans Softw Eng. 32(8):608–624. doi:10.1109/TSE.2006.83

ANSI (2004) Role based access control. Technical report, American National Standards Institute, Inc. ANSI/INCITS 359-2004

Arcuri A, Briand L (2011) A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the 33rd International Conference on Software Engineering. ICSE '11. ACM, New York, NY, USA. pp 1–10. doi: 10.1145/1985793.1985795 http://doi.acm.org/10.1145/1985793.1985795

Ben Fadhel A, Bianculli D, Briand L (2015) A comprehensive modeling framework for role-based access control policies. J Syst Softw. 107(C):110–126. doi:10.1016/j.jss.2015.05.015

Bertolino A, Daoudagh S, Kateb DE, Henard C, Traon YL, Lonetti F, Marchetti E, Mouelhi T, Papadakis M (2015) Similarity testing for access control. Inf Softw Technol. 58:355–372. doi:10.1016/j.infsof.2014.07.003

Broy M, Jonsson B, Katoen JP, Leucker M, Pretschner A (2005) Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science). Springer, Secaucus, NJ, USA

Cartaxo EG, Machado PDL, Neto FGO (2011) On the use of a similarity function for test case selection in the context of model-based testing. Softw Test Verif Reliab. 21(2):75–100. doi:10.1002/stvr.413

Chow TS (1978) Testing software design modeled by finite-state machines. IEEE Trans Softw Eng. 4(3):178–187. doi:10.1109/TSE.1978.231496

Cohen J (1977) Statistical Power Analysis for the Behavioral Sciences. Revised edn.. Academic Press, New York. doi:10.1016/B978-0-12-179060-8.50001-3. https://www.sciencedirect.com/science/article/pii/B9780121790608500013

Cohen J (1992) A power primer. Psychol Bull. 112(1):155–159. doi:10.1037/0033-2909.112.1.155

Coutinho AEVB, Cartaxo EG, Machado PDdL (2014) Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing. Softw Qual J.1–39. doi:10.1007/s11219-014-9265-z

Damasceno CDN, Masiero PC, Simao A (2016) Evaluating test characteristics and effectiveness of fsm-based testing methods on rbac systems. In: Proceedings of the 30th Brazilian Symposium on Software Engineering. SBES '16. ACM, New York, NY, USA. pp 83–92. doi:10.1145/2973839.2973849. http://doi.acm.org/10.1145/2973839.2973849

Elbaum S, Malishevsky AG, Rothermel G (2000) Prioritizing test cases for regression testing. SIGSOFT Softw Eng Notes. 25(5):102–112. doi:10.1145/347636.348910

Elbaum S, Malishevsky AG, Rothermel G (2002) Test case prioritization: A family of empirical studies. IEEE Trans Softw Eng. 28(2):159–182. doi:10.1109/32.988497

Endo AT, Simao A (2013) Evaluating test suite characteristics, cost, and effectiveness of fsm-based testing methods. Inf Softw Technol. 55(6):1045–1062. doi:10.1016/j.infsof.2013.01.001

Fabbri SCPF, Delamaro ME, Maldonado JC, Masiero PC (1994) Mutation analysis testing for finite state machines. In: Software Reliability Engineering, 1994. Proceedings., 5th International Symposium On. pp 220–229. doi:10.1109/ISSRE.1994.341378

Felderer M, Zech P, Breu R, Büchler M, Pretschner A (2015) Model-based security testing: a taxonomy and systematic classification. Softw Test Verif Reliab. doi:10.1002/stvr.1580

Ferraiolo DF, Kuhn RD, Chandramouli R (2007) Role-Based Access Control. 2nd edn. Artech House, Inc., Norwood, MA, USA

Gill A (1962) Introduction to the Theory of Finite State Machines. McGraw-Hill, New York

Gîză-Belciug F, Pentiuc SG (2015) Parallelization of similarity matrix calculus in ontology mapping systems. In: 2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER). pp 50–55. doi:10.1109/RoEduNet.2015.7311827

Hedges LV (1981) Distribution theory for glass's estimator of effect size and related estimators. J Educ Stat. 6(2):107–128. doi:10.3102/10769986006002107. https://doi.org/10.3102/10769986006002107

Henard C, Papadakis M, Perrouin G, Klein J, Heymans P, Traon YL (2014) Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. IEEE Trans Softw Eng. 40(7):650–670. doi:10.1109/TSE.2014.2327020. arXiv:1211.5451v1
Jang-Jaccard J, Nepal S (2014) A survey of emerging threats in cybersecurity. J Comput Syst Sci 80(5):973–993. doi:10.1016/j.jcss.2014.02.005 Special Issue on Dependable and Secure Computing
Jia Y, Harman M (2011) An analysis and survey of the development of mutation testing. Softw Eng IEEE Trans. 37(5):649–678. doi:10.1109/TSE.2010.62
Kampenes VB, Dyb T, Hannay JE, Sjberg DIK (2007) A systematic review of effect size in software engineering experiments. Inf Softw Technol. 49(11):1073–1086. doi:10.1016/j.infsof.2007.02.015
Masood A, Bhatti R, Ghafoor A, Mathur AP (2009) Scalable and effective test generation for role-based access control systems. IEEE Trans Softw Eng. 35(5):654–668. doi:10.1109/TSE.2009.35
Masood A, Ghafoor A, Mathur AP (2010) Fault coverage of constrained random test selection for access control: A formal analysis. J Syst Softw. 83(12):2607–2617. TAIC PART 2009 - Testing: Academic & Industrial Conference - Practice And Research Techniques
McMinn P (2004) Search-based software test data generation: A survey: Research articles. Softw Test Verif Reliab. 14(2):105–156. doi:10.1002/stvr.v14:2
Mouelhi T, Kateb DE, Traon YL (2015) Chapter five - inroads in testing access control, Advances in Computers, vol. 99. Elsevier. doi:10.1016/bs.adcom.2015.04.003 http://www.sciencedirect.com/science/article/pii/S0065245815000327
OASIS (2013) eXtensible Access Control Markup Language (XACML) Version 3.0. Technical report, Organization for the Advancement of Structured Information Standards (OASIS). http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf
OASIS (2014) XACML v3.0 Core and Hierarchical Role Based Access Control (RBAC) Profile Version 1.0. http://docs.oasis-open.org/xacml/3.0/rbac/v1.0/cs02/xacml-3.0-rbac-v1.0-cs02.pdf
Ouriques JaFS (2015) Strategies for prioritizing test cases generated through model-based testing approaches. In: Proceedings of the 37th International Conference on Software Engineering - Volume 2. ICSE '15. IEEE Press, Piscataway, NJ, USA. pp 879–882. http://dl.acm.org/citation.cfm?id=2819009.2819204
Petrenko A, Bochmann GV (1995) 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. IWPTS '94. Chapman and Hall, Ltd., London, UK. pp 95–110. http://dl.acm.org/citation.cfm?id=236187.233118
Rawald T, Sips M, Marwan N, Leser U (2015) Massively parallel analysis of similarity matrices on heterogeneous hardware. In: Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), Brussels, Belgium, March 27th, 2015. CEUR-WS, Brussels. pp 56–62
Samarati P, de Vimercati SC (2001) Access Control: Policies, Models, and Mechanisms(Focardi R, Gorrieri R, eds.). Springer, Berlin, Heidelberg. http://dx.doi.org/10.1007/3-540-45608-2_3
Simão A, Petrenko A, Yevtushenko N (2009) Generating Reduced Tests for FSMs with Extra States. In: Núñez M, Baker P, Merayo MG (eds). Springer, Berlin, Heidelberg. pp 129–145. doi:10.1007/978-3-642-05031-2_9. http://dx.doi.org/10.1007/978-3-642-05031-2_9
Torchiano M (2017) Effsize: Efficient Effect Size Computation (v. 0.7.1). CRAN package repository. https://cran.r-project.org/web/packages/effsize/effsize.pdf. CRAN package repository. [Online; accessed 20-November-2017]
Utting M, Pretschner A, Legeard B (2012) A taxonomy of model-based testing approaches. Softw Test Verif Reliab. 22(5):297–312. doi:10.1002/stvr.456
Vargha A, Delaney HD (2000) A critique and improvement of the cl common language effect size statistics of mcgraw and wong. J Educ Behav Stat. 25(2):101–132. doi:10.3102/10769986025002101. http://arxiv.org/abs/https://doi.org/10.3102/10769986025002101
Vasilevskii MP (1973) Failure diagnosis of automata. Cybernetics 9(4):653–665. doi:10.1007/BF01068590
Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Measurement. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-29044-2_3
Yoo S, Harman M (2012) Regression testing minimization, selection and prioritization: A survey. Softw Test Verif Reliab. 22(2):67–120. doi:10.1002/stv.430
Zhang YF, Tian YC, Kelly W, Fidge C (2017) Scalable and efficient data distribution for distributed computing of all-to-all comparison problems. Futur Gener Comput Syst. 67:152–162