Testing non-testable programs using association rules

Antonia Bertolino
ISTI – CNR
Pisa, Italy
antonia.bertolino@isti.cnr.it

Emilio Cruciani
University of Salzburg
Salzburg, Austria
emilio.cruciani@sbg.ac.at

Breno Miranda
Federal University of Pernambuco
Recife, Brazil
bafm@cin.ufpe.br

Roberto Verdecchia
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
r.verdecchia@vu.nl

ABSTRACT

We propose a novel scalable approach for testing non-testable programs denoted as ARMED testing. The approach leverages efficient Association Rules Mining algorithms to determine relevant implication relations among features and actions observed while the system is in operation. These relations are used as the specification of positive and negative tests, allowing for identifying plausible or suspicious behaviors: for those cases when oracles are inherently unknownable, such as in social testing, ARMED testing introduces the novel concept of testing for plausibility. To illustrate the approach we walk-through an application example.

KEYWORDS

testing, non-testable systems, association rules, plausibility testing

ACM Reference Format:
Antonia Bertolino, Emilio Cruciani, Breno Miranda, and Roberto Verdecchia. 2022. Testing non-testable programs using association rules. In *Proceedings of AST '22: 3rd ACM/IEEE International Conference on Automation of Software Test (AST '22).* ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION

Given a program P to be tested, a test case t for P can be described as a pair $(x, exp(x))$, where x is a test input and $exp(x)$ is the expected output of P for the given input x. Let us assume that by feeding the program P with input x, we get an output $P(x) = o$; we say that test case t passes if $o = exp(x)$, i.e., if the actual output o matches the expected output for x, otherwise we say that t fails. The function $exp(x)$ is referred to as the *test oracle*, which is a mechanism able to decide whether the actual output is equal (or "sufficiently close" for partial oracles) to the expected one.

Already in 1982, Weyuker [20] drew software engineers attention to the problem of *non-testable* programs, i.e., programs for which a test oracle cannot be determined with a "reasonable" effort or does not even exist. This problem is exacerbated in modern huge and dynamic software intensive systems, such as autonomous reactive systems, cyber-physical systems, and social networks, that ask not only for more scalable and adaptive test approaches, but even for a radical rethinking of what software testing means. One eminent example is WW, the cyber-cyber simulation of the Facebook platforms [4]: at ICSE 2021 Facebook launched a Call-for-Proposal of new ideas to address the challenges of testing agent-based user interaction simulation, including novel approaches to "social testing": with this term they refer to a new testing level for social media at which properties emerging from more users interacting over the platform are validated. As elegantly noted by Ahlgren et al. with reference to such simulation systems, they constitute the "epitome" of non-testable programs, because a test oracle is not unknown but "inherently *unknowable*" [3].

Our ongoing work answers that Call-for-Proposal by introducing a novel concept of testing. When testing highly non-deterministic and large-scale systems for which an oracle is unknowable, the traditional meaning of a test is lost, as we cannot decide whether a test passes or fails. Therefore, upon observing a test execution $P(x) = o$, we will decree whether the observed output o is *plausible* or otherwise *suspicious* by comparing it with a prediction $\overline{exp}(x)$ of the test oracle, which is unknowable. Such a prediction $\overline{exp}(x)$ is based on how the system has behaved so far and is used to establish the "plausibility" of the output o. More precisely, we propose to leverage Association Rules Mining (ARM) algorithms [1] for this purpose and introduce below the novel approach of ARM-basED (ARMED, for short) testing.

As better described in the following section, an Association Rule (AR) expresses an implication such as $A \Rightarrow B$, i.e., if we observe A, then it is likely that B will also be observed. The very idea of ARMED testing is simple: we mine relevant ARs by observing features and actions of a system while in operation; then a mined AR can be used as a test case specification to test the follow-up behavior of the same system. The left part of the AR is taken as the test input and the right part (the implication) as the test oracle. In this way ARMED testing supports automated mining not only of the test oracle, but also of the test input. Clearly, such a test cannot decide whether the observed test output is "correct", but can only predicate whether what we observe is credible after having monitored the system for a while: for this reason we denote our approach as *testing for plausibility*.

Interestingly, ARs can postulate not only that an element/output is positively associated with a set of elements (or, for us, a test input), but also that an element/output is not likely to be observed given a test input. Thus plausible testing can cover not only "positive" tests, i.e., we expect to see o, but also "negative" (or robustness) tests: after an input x, output o should not occur.

It is important to note that, for ensuring that the concept of ARMED testing can find application to practical systems, we need to allow the concept of plausibility to evolve, i.e., the ARs used for plausibility testing are not mined once and for all, but have to be periodically updated according to some policy and depending on the domain.

2 BACKGROUND

Let $I = \{i_1, \ldots, i_n\}$ be a set of n items and let $D = \{d_1, \ldots, d_m\}$ be a database of m observations such that each $d_j \subseteq I$. An AR is an implication of the form $A \Rightarrow B$, where the *itemsets* $A, B \subseteq I$ and $A \cap B = \emptyset$; A is called the *antecedent* and B the *consequent* of the rule. ARM algorithms learn from D a set of rules that can tell with some confidence if, having observed A, it is likely that also B is observed. However, the number of possible ARs is exponential in n and usually most of them are not significant in practical scenarios. The typical way to mine interesting ARs efficiently comprises two steps: 1) mining frequent itemsets, i.e., finding all subsets of I that appear frequently in D; 2) generating interesting ARs, i.e., using the frequent itemsets to generate significant rules.

2.1 Mining frequent itemsets

Given a database of observations D as input and a frequency threshold $\sigma \in [0, 1]$, mining frequent itemsets from D means finding all itemsets (i.e., nonempty subsets of I), as well as their frequency, that appear in at least a fraction σ of D. The task requires to look for the frequency of all possible itemsets (2^n-1 in the worst case), which has an enormous computational cost even for small databases.

In practice the problem can be solved by efficient algorithms that exploit the down-ward closure property of frequency, i.e., all subsets of a frequent set are also frequent, as in *Apriori* [2], or that exploit advanced data structures or algorithms, as in *FP-Growth* [10] or *Eclat* [22]. Nevertheless, when dealing with big data the cost of such a task could still be prohibitive. For this scenario there exist also scalable random randomized algorithms such as *Toivonen's* [18], or algorithms that only need a limited number of passes over the database such as *SON* [17], overall requiring lower memory and lending themselves for parallel computing environments.

An extension of the standard problem discussed so far is the mining frequent itemsets in *evolving scenarios*, which is of interest for applying ARM algorithms to our ARMED testing concept. In particular, we could imagine an evolving database (observations are added to the database over time), a database of evolving observations (items are added to observations of the database over time), or a combination thereof. While some clever adaptations of the previously discussed algorithms have been provided for the first scenario, to the best of our knowledge the others have not been studied in the data mining literature and it is not clear if something smarter than trivial adaptations can be performed.

2.2 Generate association rules

Once the frequent itemsets of D are computed, it is possible to generate association rules by simply considering all their binary partitions, i.e., for each itemset S we generate all the rules of the form $A \Rightarrow B$, where $A \subset S$, $A \neq \emptyset$, and $B = S \setminus A$. However, we still need to measure the significance of the generated rules in order to only consider the interesting ones.

Originally defined in [1], the trivial interest measure is *support*:

$$\text{supp}(A \Rightarrow B) := \Pr(A \cap B), \qquad (1)$$

namely the probability of observing A and B together in D, measured as the fraction of data-entries containing both A and B. Generating ARs from frequent itemsets with frequency threshold $\sigma \in [0, 1]$ guarantees that each rule $A \Rightarrow B$ has $\text{supp}(A \Rightarrow B) \geq \sigma$.

A measure defined together with support in [1] is *confidence*:

$$\text{conf}(A \Rightarrow B) := \Pr(B \mid A), \qquad (2)$$

namely the conditional probability of observing B given that A has already been observed. Confidence indicates how often the rule has been observed to be true. A rule $A \Rightarrow B$ is interesting if $\text{conf}(A \Rightarrow B) \geq \gamma$ for a threshold $\gamma \in [0, 1]$.

Another commonly-used measure is *leverage*, defined in [15]:

$$\text{levg}(A \Rightarrow B) := \Pr(A \cap B) - (\Pr(A) \cdot \Pr(B)), \qquad (3)$$

namely the difference between the probability of observing A and B together and what would be expected if A and B were independent. If $\text{levg}(A \Rightarrow B) = 0$, then A and B are independent; if $\text{levg}(A \Rightarrow B) > 0$, then A and B are positively correlated; if $\text{levg}(A \Rightarrow B) < 0$, then A and B are negatively correlated. The absolute value of leverage indicates the degree of correlation between A and B, while its sign if the correlation is direct or inverse. A rule $A \Rightarrow B$ is interesting if $|\text{levg}(A \Rightarrow B)| \geq \lambda$ for a threshold $\lambda \in [0, 1]$.

3 ARMED TESTING

As previously discussed, the idea behind ARMED testing is simple and elegantly fits several specific scenarios. In every scenario the approach proceeds at a high level in the same fashion. An overview of the ARMED testing process is depicted in Figure 1.

Given three parameters $\sigma, \gamma, \lambda \in [0, 1]$, the generic workflow of ARMED testing follows two main steps. In the *preparation phase*, the approach generates a (plausibility) test suite by simply observing the considered software system and mining ARs from the observations that have been made:

(1) By observing features and actions of the system in operation, we build a database of observations D, each one modeled as a set of items. An accurate modeling is crucial for the generation of relevant rules and must be customized according to the specific scenario by focusing on the aspects of interest; however, the ARMED testing approach is independent of the modeling choices and only requires observations to be modeled as discrete sets of items. This is depicted in Figure 1, Step P1.
(2) We mine from the database D the set F of frequent itemsets, with frequency threshold σ, using the most suitable algorithm according to the scale of the problem. This is depicted in Figure 1, Step P2.
(3) Using F we generate a set R of ARs guaranteeing that each rule $(A \Rightarrow B) \in R$ has $\text{supp}(A \Rightarrow B) \geq \sigma$, $\text{conf}(A \Rightarrow B) \geq \gamma$, and $|\text{levg}(A \Rightarrow B)| \geq \lambda$ (respectively Equations (1), (2), and (3)). This is depicted in Figure 1, Step P3.

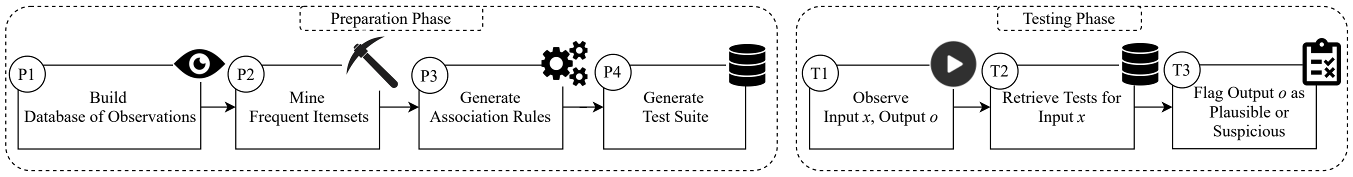
Preparation Phase
P1
Build Database of Observations
P2
Mine Frequent Itemsets
P3
Generate Association Rules
P4
Generate Test Suite
Testing Phase
T1
Observe Input x, Output o
T2
Retrieve Tests for Input x
T3
Flag Output o as Plausible or Suspicious

Figure 1: ARMED Testing approach overview.

(4) We generate a test suite T by mapping each rule $(A \Rightarrow B) \in R$ to a positive test $(x : A; \overline{exp}(x) : B)$ if $\text{levg}(A \Rightarrow B) > 0$, or to a negative test $(x : A; \neg\overline{exp}(x) : B)$ if $\text{levg}(A \Rightarrow B) < 0$; we denote by $\overline{exp}(x)$ the *expected output* of positive tests and by $\neg\overline{exp}(x)$ the *unexpected output* of negative tests, i.e., predictions of the test oracle. This is depicted in Figure 1, Step P4.

The preparation phase can be adapted to evolving scenarios, for example by updating the database D whenever the system changes (e.g., a new observation is made or an existing observation is updated) or by considering only the most recent history under a sliding window model of observations.

In the *testing phase*, the approach tests the system for which significant rules have been generated.

(1) We observe an input $x = A$ as well as its output o. This is depicted in Figure 1, Step T1.
(2) We "get armed" and retrieve the set of tests $T(A) \subseteq T$, i.e., all positive and negative tests generated in the preparation phase that have $(x : A)$. This is depicted in Figure 1, Step T2.
(3) We test the program: for each positive test $(x : A; \overline{exp}(x) : B) \in T(A)$ the output o is flagged *plausible* if and only if $B \subseteq o$, i.e., the expected output $\overline{exp}(x)$ is observed in the actual output, or *suspicious* otherwise; for each negative test $(x : A; \neg\overline{exp}(x) : B) \in T(A)$ the output o is flagged *plausible* if and only if $B \nsubseteq o$, i.e., the unexpected output $\neg\overline{exp}(x)$ is not observed in the actual output o, or *suspicious* otherwise. This is depicted in Figure 1, Step T3.

In the case of multiple tests for the same input/output pair, multiple flags can be aggregated into a single flag, e.g., via a voting strategy where the most frequent flag wins or via a more conservative strategy where a suspicious flag is enough to flag the output suspicious.

4 A WALK-THROUGH EXAMPLE

We anticipate that ARMED testing can find useful variant applications in several scenarios involving non-testable programs. Herein we illustrate a possible application of our ARMED testing approach assuming the specific scenario that the system under test is the Facebook WW simulator mentioned in Section 1. Among other purposes, ARMED testing could be applied to validate the behavior of BOTs that are used in WW to simulate real individuals.

In particular, suppose we want to test if the interactions between the BOTs inhabiting the Facebook digital twin platform are realistic (w.r.t. the personas they are modeled from). First we learn from the real system how typical users' interaction patterns are correlated with users' features taken into account to create personas. Then we test BOTs by checking if their interaction patterns are expected or not, by comparing them with those of the personas they are modeled from.

In more detail, following the workflow described in Figure 1, we start by observing the real system, namely the real Facebook platform, in order to build a database D of observations[1]. In this specific scenario, an observation consists of modeling the users as sets of features. For the purpose of this example, we use the `MyPersonality` database[2] to illustrate a possible modeling via aggregated features such as *personality traits* and *network measures*[3].

`MyPersonality` in fact aggregates data regarding personality traits of a set of Facebook users and a set of their relative social network measures. More in detail, regarding personality traits, the database includes five different traits, namely *openness to experience* (*OPN*), *conscientiousness* (*CON*), *agreeableness* (*AGR*), *extroversion* (*EXT*), and *neuroticism* (*NEU*) [16]. The data regarding user traits was gathered by involving a set of Facebook users to a psychological research that consisted in filling in a personality questionnaire. In addition to personality traits, each user is mapped to a set of social network metrics. The metrics included in the database are: *network size* (*SIZ*), *betweenness centrality* (*BET*), *density* (*DEN*), *brokerage* (*BRK*), and *transitivity* (*TRN*) [19].

Concerning the features of `MyPersonality`, we model the values of each user as a set of "items". Specifically, each user is mapped to the presence/absence of a certain metric if the metric value of the user is above/below a threshold (e.g., median value). By considering the data included in `MyPersonality`, we could for instance model a user u as $u = \{EXT, AGR, SIZ, BET\}$ if u is extrovert, agreeable, has a large network size and a high betweenness centrality.

Once the database D is built, we mine frequent itemsets, i.e., subsets of features of users that are common in the database, for example setting $\sigma = 0.05$, i.e., observed in at least 5% of users. Then we generate relevant ARs by setting, for example, a confidence threshold $\gamma = 0.5$ and a leverage threshold $\lambda = 0.1$. In this specific application we model users/personas features as a set of personality traits and users/bots interactions as a set of network metrics. For this reason, we restrict the set of generated rules R to having the antecedent $A \subseteq \{EXT, NEU, AGR, CON, OPN\}$ and the consequent $B \subseteq \{SIZ, BET, DEN, BRK, TRN\}$. Finally, from the set of rules R, we generate a set of tests T where the inputs are sets of personality traits and the expected outputs are network measures. An example of rules could be $\{EXT, AGR\} \Rightarrow \{SIZ\}$ or $\{AGR, OPN\} \Rightarrow \{SIZ, BET\}$, i.e., we expect a BOT modeled from

[1]Such observations must be obviously carried out in respect of privacy regulations, such as GDPR law.
[2]https://www.psychometrics.cam.ac.uk/productsservices/mypersonality. Accessed on 09/10/2021.
[3]Note that this is just a possible modeling of users, driven by the `MyPersonality` database, but that a different and more comprehensive modeling could be adopted.

an extrovert and agreeable persona to have a large network size, or a BOT modeled from an agreeable and open to experience persona to have a large network size and a high betweenness centrality.

In order to test a BOT b, we also model b using the same features considered for the users. Therefore, some of the features of b will be personality traits (call this set A_b), while other will be network measures (call this set B_b). We then look in the test suite for $T(A_b)$, i.e., the set of generated tests that have input $x = A_b$, and finally test the BOT b: for each positive test $(x : A_b; \overline{exp}(x) : B)$, we flag the behavior of BOT b *plausible* if the expected behavior B is observed, namely if $B \subseteq B_b$, or *suspicious* otherwise; for each negative test $(x : A_b; \neg\overline{exp}(x) : B)$, we flag the behavior of BOT b *plausible* if the unexpected behavior B is not observed, namely if $B \nsubseteq B_b$, or *suspicious* otherwise. As described in Section 3, flags of BOT b are then aggregated according to some strategies, e.g., if the majority of the flags are *suspicious* then b does not behave as the persona it is modeled from would in the majority of the tested interactions.

5 RELATED WORK

Conceived in the early 90's for mining interesting relations in large data bases, e.g., in market basket analyses, ARs have already found several applications in software engineering, e.g., for ranking clones [13], for revealing false test alarms [11], even for statically detecting software bugs such as, e.g., in [6, 12]. However, ours is the first proposal of leveraging ARM techniques as the basis for formulating a new software testing concept. The only similar work we are aware of is an early short paper by Zheng et al. [23], who proposed to use ARs for validating the outcome of Web Search Engines and detecting suspicious outcomes. Indeed, Web Search Engines are again an example of systems whose test oracle is unknowable, and the idea formulated in that work could be seen as an application of our ARMED testing approach. However, that work only presented one specific application example of using ARs for testing search engine results and did not explicitly recognize the novel broader concept of *testing for plausibility* behind the idea. Moreover, they did not consider negative tests and did not discuss the evolvability and scalability of the approach, as we do.

There exists a large literature on test oracles: a 2015 survey [5] found 694 papers over the period 1978-2012 dealing with the test oracle problem, whereas a 2018 mapping study [14] identified until mid 2014 a set of 137 primary studies focusing on testing non-testable systems. In the latter, solution are classified under five "umbrella" techniques: N-version Testing, Metamorphic Testing, Assertions, Machine Learning, and Statistical Hypothesis Testing.

Among these, Metamorphic Testing (MT) [7] has by far received the highest attention with regard to testing non-testable programs. MT is in fact also the solution currently pursued to test the Facebook WW simulator [3]. MT does not need to know the expected outputs, but relies on (metamorphic) relations that must hold between the outputs produced by some "source" test cases and their "follow up" ones. MT does not really solve the oracle problem though, rather it shifts it from determining the expected output to discovering meaningful Metamorphic Relations (MRs), for which "*the current state of the art is only, at best, partially automated*" [3]. In fact, for Facebook WW testing, some effective MRs have been manually identified; however the approach requires the support of domain experts and appears *ad hoc* and non scalable.

MT and ARMED testing could be seen as complementary approaches: in our intuition, ARMED testing can fit the need for testing large scale systems when some continuity in the exposed behavior is expected. In such conditions, the advantage over MT is evident: ARs can be derived automatically and systematically, in contrast with the difficulty of manually deriving metamorphic relations. In WW, for example, humans accessing the platform are simulated by BOTs: at Facebook scale it is simply impossible to think to build a faithful simulator that can act as a test oracle for each individual BOT. However, we can plausibly expect that humans sharing similar traits will behave similarly when exposed to similar stimuli. Hence we can test whether the behavior of a BOT is plausible when referring to the humans it simulates. On the other hand, there can be systems for which homogeneity of behavior cannot be assumed and even deciding what is plausible can be difficult. Examples could be testing of scientific software [8], or of machine learning classifiers [21]. In both cases we cannot easily predict what is a plausible output, but some MRs have been settled leveraging the inherent mathematics of the domain.

Finally, also specification mining approaches and, more in particular, dynamic invariant detection from execution traces, such as Daikon [9], aim at identifying properties that hold for a program under test, similarly to what we do. However such approaches, although scalable, have been conceived for software systems testing and it is yet unclear if they can be adapted to the more abstract dimension of social testing addressed by ARMED testing. Although a more detailed comparison is needed, we foresee a semantic difference between extracting an invariant property that should locally hold for all executions of every BOT instance and finding a rule that applies to a broad context, also involving human simulations.

6 CONCLUSIONS AND FUTURE WORK

With this ongoing work we introduced a novel scalable approach for testing non-testable programs denoted as ARMED testing. Our approach leverages efficient association rules mining algorithms to determine relevant implication relations among features and actions observed while the system is in operation. As part of our future work we plan to instantiate the concept of ARMED testing in several testing scenarios. In addition to the case of testing the realism of BOTs described in Section 4, we foresee several other possible applications. For instance, ARMED testing could allow for detecting abnormal user or community behaviors. We could leverage ARs for hinting at abnormal users ("bad actors") in two ways: if ARs of known abnormal behaviors are available, they can be compared against other users' behavior; otherwise, when an unexpected behavior is observed, the output is reported to the tester as a symptom worthy of attention. Moreover, ARMED testing could also help to early detect deviations for ultra-large scale systems, looking at the overall trend of observed outcomes. We also plan to investigate different application scenarios of ARMED testing by means of empirical experimentation in tight collaboration with an industrial partner, in order to evaluate the feasibility, effectiveness, and scalability of the approach.

Acknowledgement. This work is supported by a Facebook 2021 Research Award on "Agent-based user interaction simulation to find and fix integrity and privacy issues".

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules between Sets of Items in Large Databases. *SIGMOD Rec.* 22, 2 (June 1993), 207–216. https://doi.org/10.1145/170036.170072

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. 1994. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, Vol. 1215. Citeseer, 487–499.

[3] John Ahlgren, Maria Eugenia Berezin, Kinga Bojarczuk, Elena Dulskyte, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Maria Lomeli, Erik Meijer, et al. 2021. Testing Web Enabled Simulation at Scale Using Metamorphic Testing. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 140–149. https://doi.org/10.1109/ICSE-SEIP52600.2021.00023

[4] John Ahlgren, Kinga Bojarczuk, Sophia Drossopoulou, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Maria Lomeli, Simon M. M. Lucas, Erik Meijer, Steve Omohundro, Rubmary Rojas, Silvia Sapora, and Norm Zhou. 2021. Facebook's Cyber–Cyber and Cyber–Physical Digital Twins. In *Evaluation and Assessment in Software Engineering* (Trondheim, Norway) *(EASE 2021)*. Association for Computing Machinery, New York, NY, USA, 1–9. https://doi.org/10.1145/3463274.3463275

[5] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Trans. Softw. Eng.* 41, 5 (May 2015), 507–525. https://doi.org/10.1109/TSE.2014.2372785

[6] Pan Bian et al. 2018. Nar-miner: Discovering negative association rules from code for bug detection. In *ESEC/FSE*. 411–422.

[7] Tsong Yueh Chen et al. 2018. Metamorphic Testing: A Review of Challenges and Opportunities. *ACM Comput. Surv.* 51, 1, Article 4 (Jan. 2018), 27 pages. https://doi.org/10.1145/3143561

[8] Junhua Ding, Dongmei Zhang, and Xin-Hua Hu. 2016. An application of metamorphic testing for testing scientific software. In *Proceedings of the 1st International Workshop on Metamorphic Testing*. 37–43.

[9] Michael D Ernst, Jeff H Perkins, Philip J Guo, Stephen McCamant, Carlos Pacheco, Matthew S Tschantz, and Chen Xiao. 2007. The Daikon system for dynamic detection of likely invariants. *Science of computer programming* 69, 1-3 (2007), 35–45.

[10] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM sigmod record* 29, 2 (2000), 1–12.

[11] Kim Herzig and Nachiappan Nagappan. 2015. Empirically detecting false test alarms using association rules. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 39–48.

[12] Benjamin Livshits and Thomas Zimmermann. 2005. Dynamine: finding common error patterns by mining software revision histories. *ACM SIGSOFT Software Engineering Notes* 30, 5 (2005), 296–305.

[13] Manishankar Mandal, Chanchal K Roy, and Kevin A Schneider. 2014. Automatic ranking of clones for refactoring through mining association rules. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, 114–123.

[14] Krishna Patel and Robert M. Hierons. 2018. A Mapping Study on Testing Non-Testable Systems. *Software Quality Journal* 26, 4 (Dec. 2018), 1373–1413. https://doi.org/10.1007/s11219-017-9392-4

[15] Gregory Piatetsky-Shapiro. 1991. Discovery, analysis, and presentation of strong rules. *Knowledge discovery in databases* (1991), 229–238.

[16] Sebastiaan Rothmann and Elize P Coetzer. 2003. The big five personality dimensions and job performance. *SA Journal of Industrial Psychology* 29, 1 (2003), 68–74.

[17] Ashok Savasere et al. 1995. *An efficient algorithm for mining association rules in large databases*. Technical Report. Georgia Institute of Technology.

[18] Hannu Toivonen. 1996. Sampling Large Databases for Association Rules. In *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB '96)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 134–145.

[19] Stanley Wasserman and Katherine Faust. 1994. *Social Network Analysis: Methods and Applications*. Cambridge University Press. https://doi.org/10.1017/CBO9780511815478

[20] Elaine J Weyuker. 1982. On testing non-testable programs. *Comput. J.* 25, 4 (1982), 465–470.

[21] Xiaoyuan Xie, Joshua WK Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. 2011. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software* 84, 4 (2011), 544–558.

[22] Mohammed Javeed Zaki. 2000. Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering* 12, 3 (2000), 372–390.

[23] Wujie Zheng, Hao Ma, Michael R. Lyu, Tao Xie, and Irwin King. 2011. Mining Test Oracles of Web Search Engines. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE '11)*. IEEE Computer Society, USA, 408–411. https://doi.org/10.1109/ASE.2011.6100085