UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Using Metamorphic Testing to Identify Authentication Vulnerabilities in Android Mobile Applications

Misael Costa Júnior

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP
SÃO CARLOS

Misael Costa Júnior

Using Metamorphic Testing to Identify Authentication Vulnerabilities in Android Mobile Applications

Thesis submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Márcio Eduardo Delamaro

USP – São Carlos
August 2024

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

C95u

Costa Júnior, Misael
Using Metamorphic Testing to Identify Authentication Vulnerabilities in Android Mobile Applications / Misael Costa Júnior; orientador Márcio Eduardo Delamaro. -- São Carlos, 2024.
123 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2024.

1. CIÊNCIA DA COMPUTAÇÃO. 2. ENGENHARIA DE SOFTWARE. 3. PROCESSO DE SOFTWARE. 4. QUALIDADE DE SOFTWARE. 5. SEGURANÇA DE SOFTWARE. I. Eduardo Delamaro, Márcio , orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Misael Costa Júnior

Usando o Teste Metamórfico para Identificar Vulnerabilidades de Autenticação em Aplicativos Android

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

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Márcio Eduardo Delamaro

USP – São Carlos
Agosto de 2024

This research is especially dedicated to my parents, Luiza Maria and Misael Costa,
and also, to my beautiful sisters, Dayanne Lises and Danielly Tamires.
This dedication is extended to my beloved family (Miriane, Zeus, Floquinha, and Lince),
who have always supported me during this journey.

ACKNOWLEDGEMENTS

Firstly, I express my gratitude to God for providing me with opportunities for growth and safeguarding me throughout this journey.

My special acknowledgements to my parents, Luiza Maria and Misael Costa, for their invaluable advice shared during this expedition and their sacrifices, dedication, and selflessness of their time and personal projects so that I could reach my objectives and dreams. To my sisters, Dayanne Lises and Danielly Tamires, who, despite challenging circumstances, always displayed patience and understanding. Finally, to my family (Miriane, Zeus, Floquinha, and Lince), for their unwavering support and understanding have been fundamental.

My gratitude to my advisor, Prof. Dr. Márcio Delamaro (ICMC-USP), for the invaluable opportunities and trust throughout this journey, which commenced in 2015, with my Master's degree. His guidance, support, and significant contributions have played a crucial role in the completion of this work. I appreciate not only his professional example, but also his patience and ethical approach, which have been a constant source of inspiration to me.

My heartfelt appreciation to Prof. Dr. Domenico Amalfitano, from the University of Naples Federico II, in Italy; his support has been essential in this journey and I am truly grateful for his invaluable assistance and guidance.

To my friend Mauricio Guimarães, who honored me with his friendship and shared his delightful experiences of Amor Fati, through which I have learned to accept and embrace everything that happens in life, including challenging and painful experiences, instead of resisting or resenting them. I extend my gratitude to my friends who accompanied me during this long journey: Alfredo Guilherme, André Pessoa, Claudinei Junior, Diógenes Dias, Gustavo Prudencio, João Choma, Joelson Santos, Jorge Cutigi, José Filomen, Lina Garcés, Ricardo Vilela, and Stevão Andrade. Their friendship was essential and one of the great achievements I have gained.

To the professors of the State University of Piauí (UESPI), who provided me with opportunities and made me believe in my potential. My special thanks to Prof. Dr. Bringel Filho, an exemplary professional and friend, who I greatly esteem and admire.

This PhD work was partially financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

“He is called a free spirit who thinks differently from what, on the basis of his origin, environment, his class and profession, or on the basis of the dominant views of the age, would have been expected of him”

(Friedrich Nietzsche, Human, All Too Human: A Book for Free Spirits)

RESUMO

JÚNIOR, M. C. **Usando o Teste Metamórfico para Identificar Vulnerabilidades de Autenticação em Aplicativos Android**. 2024. 123 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

O amplo uso de aplicativos móveis, abrangendo atividades desde operações bancárias até tarefas de escritório, intensificou a demanda por atividades de garantia de qualidade. No entanto, testes de aplicativos móveis apresentam desafios distintos, como restrições de energia (ou seja, Desempenho), adaptação de interface (ou seja, Usabilidade) e privacidade de dados do usuário (ou seja, Segurança) — exemplos de Requisitos Não Funcionais (RNFs). Segurança, um dos RNFs mais críticos, é crucial para sistemas de software, especialmente em aplicativos móveis. A existência de falhas de segurança (ou seja, vulnerabilidades) representam um risco substancial, podendo resultar em acesso não autorizado ou ataques maliciosos. Testes de segurança tradicionais são frequentemente dispendiosos e complexos, complicados ainda mais pelo "problema do oráculo". Em resposta, o Teste Metamórfico (TM) surgiu como uma abordagem estratégica para enfrentar esses desafios. Utilizando Relacionamentos Metamórficos (RMs) derivados do *System Under Testing (SUT)*, o TM avalia falhas no sistema. Estudos recentes exploraram a eficácia do TM em revelar falhas relacionadas a RNFs, incluindo Desempenho e Segurança, em domínios como sistemas *Web* e aplicativos móveis. Esta pesquisa de doutorado introduz uma técnica inovadora de TM direcionada a cinco vulnerabilidades relatadas pela OWASP em aplicativos móveis Android, que afetaram principalmente os métodos de autenticação por nome de usuário e senha. A técnica utiliza cinco RMs para avaliar a presença dessas vulnerabilidades, complementada por um Ambiente de Teste de Vulnerabilidade Metamórfica que automatiza o processo de teste. Este ambiente simplifica a geração e execução de casos de teste *source* de *follow-up*. Em um experimento abrangente com 163 aplicativos Android comerciais, a técnica proposta identificou 159 vulnerabilidades, sendo que 108 aplicativos revelaram pelo menos uma vulnerabilidade. Dentre os métodos usados para validar as vulnerabilidades encontradas, foram contatadas 37 empresas para relatar os problemas em seus aplicativos. Nove delas responderam diretamente para validar as vulnerabilidades, e três solicitaram consultas online para corrigi-las. Notou-se que, embora 26 empresas não tenham respondido, lançaram uma nova versão do app sem as vulnerabilidades relatadas. Surpreendentemente, descobriu-se que a qualidade percebida pelo usuário não está necessariamente relacionada à ausência de vulnerabilidades. Mesmo aplicativos bem avaliados podem conter falhas de segurança.

Palavras-chave: Teste de Segurança, Teste Metamórfico, Teste de Vulnerabilidade, Teste de Aplicativos Móveis, Teste Beaseado em GUI.

ABSTRACT

JÚNIOR, M. C. **Using Metamorphic Testing to Identify Authentication Vulnerabilities in Android Mobile Applications**. 2024. 123 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

The widespread use of mobile apps, spanning activities from banking to office tasks, has intensified demands for quality assurance activities. Nevertheless, mobile apps testing faces unique challenges, such as power constraints (i.e., Performance), interface adaptation (i.e., Usability), and user data privacy (i.e., Security) — examples of Non-Functional Requirements (NFRs). Security, one of the most critical NFRs, is pivotal for software systems, especially in mobile apps. The existence of security faults (i.e., vulnerabilities) poses a substantial risk, potentially resulting in unauthorized access or malicious attacks. Traditional security testing is often costly and intricate and further hampered by the "oracle problem." In response, Metamorphic Testing (MT) has emerged as a strategic approach to address those challenges. Adopting Metamorphic Relationships (MRs) derived from the Application Under Testing (AUT), MT assesses faults in applications. Recent studies have explored MT's effectiveness in uncovering NFR-related faults, including those in performance and security, across domains such as Web systems and mobile apps. This PhD thesis introduces an innovative MT technique targeting six vulnerabilities reported by OWASP in Android mobile apps, which have affected mainly username and password authentication methods. The technique employs five MRs to evaluate the existence of such vulnerabilities, complemented by a Metamorphic Vulnerability Testing Environment automating the testing process. The environment streamlines both generation and execution of source and follow-up test cases. In an extensive experiment with 163 commercial Android applications, the technique identified 159 vulnerabilities, with 108 apps revealing at least one of them. Towards confirming the vulnerabilities identified, 37 companies were contacted for reporting those in their apps, of which nine directly responded to ratifying them, with three even requesting online consultations for addressing the issues. Although not responding to the reports, 26 companies released new versions of their apps, addressing the reported vulnerabilities. The experiments also revealed a surprising finding: contrarily to expectations, the user-perceived quality does not necessarily correlate with the absence of vulnerabilities; indeed, even applications perceived by users as high-quality are not immune to them.

Keywords: Security Testing, Metamorphic Testing, Vulnerability Testing, Mobile Apps Testing, GUI Based Testing.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AG	Authorization Grant
AndDev	Android Device
APIs	Application Programming Interfaces
APPs	Applications
AR	Authentication Request
ASEs	Academic Search Engines
AT	Access Token
Auth-Server	Authorization Server
AVD	Android Virtual Device
BNA	Based on the Nature of the Application
BVA	Boundary Value Analysis
CAI	Coverage analyzers and instrumenters
CB	Code-Based
CCFG	Control-Flow Graph of Call-Backs
CFB	Control Flow-Based
CFG	Control Flow Graph
CPRs	Cyber-physical systems
CR	Capture/Replay
CSUR	ACM Computing Surveys
DFB	Data Flow-Based
DT	Decision Tables
EG	Error Guessing
EP	Equivalence Partitioning
FB	Fault-Based
FN	False Negatives
FOSS	Free Open-Source Software
FP	False Positives
FRs	Functional Requirements
FS	Formal Specifications
FSM	Finite State Machine
GPS	Global Positioning System

GQM	Goal–Question–Metric
GUI	Graphical User Interface
HTTP-CC	HTTP Connection Channel
HTTPS-CC	HTTPS Connection Channel
IDB	Input Domain-Based
JSS	Journal of Systems & Software
MB	Model-Based
MITM	Man-in-the-Middle
ML	Machine Learning
MT	Metamorphic Testing
MUT	Mutation Testing
MVC	Multi-Valued Composite
NFRs	Non-Functional Requirements
NTSSLDC	No Trusted SSL/TLS Digital Certificate
OC	Oracle/file comparators and assertion checking
OP	Operational Profile
PR	Protected Resources
PT	Pairwise Testing
PV	Performance Variation
q.a	quality attribute
q.c	quality characteristic
q.r	quality requirement
QoS	Quality of Service
RandT	Random Testing
RE	Reliability evaluation
REng	Requirements Engineering
Res-Server	Resource Server
RO	Resource Owner
RQ	Research Question
RT	Regression Testing
SEIE	Based on the Software Engineer's Intuition and Experience
SIS	Sign-In Screen
SLR	Systematic Literature Review
SMS	Systematic Mapping Study
SQuaRE	Systems and software Quality Requirements and Evaluation
SSDC	Self Signed Digital Certificate
SSL	Secure Sockets Layer

SUT	System Under Testing
SWEBOK	Software Engineering Body Of Knowledge
T	Tracers
TaaS	Testing as a Service
TG	Test Generators
TH	Test Harnesses
TSL	Transport Security Layer
TSSLDC	Trusted SSL/TLS Digital Certificate
TTL	Time-to-live
UB	Usage-Based
UML	Unified Modeling Language
UOH	User Observation Heuristics
UPC	User and Password Credentials
WM	Workflow Models
ZIPT	Zero-Integration Performance Testing

CONTENTS

CHAPTER

1

INTRODUCTION

Mobile devices have significantly entered our daily lives, becoming an integral part of a growing global market, and a staggering 7,516 billion smartphone users are expected by 2026 worldwide (O'DEA, 2020). Among mobile operating systems, Android stands out as the most prevalent, powering 70% of devices, while iOS is found on 27.7% of them (LARICCHIA, 2022). Towards grasping the sheer ubiquity of mobile applications (apps) on Android devices, let us consider more than 111.3 billion apps were downloaded in 2021 from Google Play Store, the official app marketplace for Android OS (CECI, 2022). Such mobile apps have become indispensable in our daily routines, spanning across entertainment (e.g., Netflix), gaming (e.g., Among Us), transportation (e.g., Uber), social media (e.g., Instagram, Facebook, Twitter, TikTok), education (e.g., Duolingo), and e-commerce (e.g., Amazon, eBay, Zalando).

However, since the development of apps extends beyond entertainment to encompassing safety-critical and time-sensitive domains, their quality has become paramount. Security is a fundamental aspect, since mobile apps often track our movements, monitor our interests, and increasingly control IoT-connected devices that shape our environment (SEQUEIROS *et al.*, 2020). Usability is equally crucial; an app's success or failure depends on user satisfaction and judgment (HENRY, 2021). Furthermore, mobile apps must operate efficiently, avoiding excessive CPU, memory, and energy consumption (ALOTAIBI; CLAUSE; HALFOND, 2020) while delivering robust performance in terms of throughput and response times. They must also consistently meet user expectations across the myriad of devices and platforms in use by their customer base (YU *et al.*, 2021).

The aforementioned considerations underscore the importance of addressing Security, Usability, and Performance, all of which falling under the purview of Non-Functional Requirements (NFRs). NFRs, also commonly referred to as quality requirements (q.r), provide criteria for evaluations of a system's performance rather than specify particular behaviors. In contrast, Functional Requirements (FRs) define the functions of a system or its components. A function is essentially a specification of the behavior between outputs and inputs. In simpler terms,

NFRs focus on "how the system should perform" as opposed to "what the system should do" (ECKHARDT; VOGELSANG; FERNÁNDEZ, 2016).

In the realm of security, the proliferation of available apps has been accompanied by a surge in vulnerabilities (i.e., security faults) and ensuring security controls has become more difficult (VILLAMIZAR *et al.*, 2020). According to a 2020 report by Skybox Security (SECURITY, 2020), vulnerabilities in mobile operating systems witnessed a 50 percent increase, primarily driven by Android faults, coinciding with the convergence of home networks and personal devices with corporate networks, owing to the mass transition toward remote workforces. Such developments underscore the urgent need for organizations to enhance access controls and gain comprehensive visibility into all entry and exit points within their network infrastructure (SECURITY, 2020). In this scenario, mobile apps play a critical role in managing and sharing users' sensitive information, including family addresses, private contacts, phone numbers, emails, messages, and credit card credentials. Protecting this wealth of data from malicious mobile attacks is of paramount importance (MCAFEE, 2017).

Most malicious attacks take advantage of vulnerabilities in mobile apps, such as unsecured sensitive data storage (OWASP, 2016a), unencrypted networking channels (OWASP, 2016b), and insecure authentication (OWASP, 2016c). A vulnerability is a special type of fault related to security properties that can be exploited by an attacker (FELDERER *et al.*, 2016) and the main reason for security and privacy breaches in commercial software. Vulnerabilities are very often caused by software faults made by developers in their projects, as pointed out by Xie, Lipford and Chu (2011) from an analysis of bug tracking systems. Moreover, they can also occur due to a lack of understanding and improperly specification of security requirements in software development projects (VILLAMIZAR *et al.*, 2020).

While on the one hand vulnerabilities are introduced mainly in function of a lack of developers´ specific knowledge about software security issues (POTTER; MCGRAW, 2004; ARKIN; STENDER; MCGRAW, 2005; XIE; LIPFORD; CHU, 2011; NAGAPPAN; SHIHAB, 2016; RIBEIRO; CRUZES; TRAVASSOS, 2018; VILLAMIZAR *et al.*, 2020; JUNIOR *et al.*, 2022), on the other hand they are hardly detectable for different reasons such as *(i)* complexity and inefficiency of the known testing techniques (RIBEIRO; CRUZES; TRAVASSOS, 2018; JUNIOR *et al.*, 2022), *(ii)* lack of testing of apps against real-world known vulnerabilities (JUNIOR *et al.*, 2022), and *(iii)* difficult definition and implementation of oracles in executable test cases for assessing the absence of vulnerabilities (RIBEIRO; CRUZES; TRAVASSOS, 2018; CHEN *et al.*, 2016; JUNIOR *et al.*, 2022).

1.1 Problem Statement

Authentication and authorization issues represent pervasive security vulnerabilities, consistently ranking as the third highest concern in the OWASP Top 10 vulnerabilities (OWASP,

2023b). In light of such a pressing concern, Metamorphic Testing (MT) technique was adopted in this PhD research for the detection of vulnerabilities in Android mobile apps that have affected mainly username and password authentication methods. MT supports the creation of new test cases and is usually adopted as a valid alternative to alleviate the oracle problem definition (CHEN; CHEUNG; YIU, 1998). Oracle Problem occurs when the correctness of outputs generated from valid input domain data cannot be judged or its judgement is difficult through practical means (WEYUKER, 1982; BARR *et al.*, 2015). At its core, MT revolves around the definition of a set of Metamorphic Relationships (MRs), which are relations derived from properties to be upheld by the tested program (CHEN; CHEUNG; YIU, 1998; SEGURA *et al.*, 2016).

This research direction has been motivated by two primary factors and challenges. First, username and password authentication methods are some of the most commonly adopted ones in mobile apps (OWASP, 2023a) and new vulnerabilities in those methods are frequently identified and reported by both users and developers. Notably, the Common Weakness Enumeration (CWE), a list of software and hardware weaknesses, listed at least six types of vulnerabilities affecting authentication methods among the Top 25 Most Dangerous Software Weaknesses in 2021[1].

Furthermore, the OWASP report on the "Top 10 Mobile Risks" underscored the significance of risks related to username and password authentication methods[2]. In its 2023 version updated by OWASP, risks associated with those methods have ascended to occupying the third position, indicating their elevated prominence as critical security concerns in mobile apps (OWASP, 2023b). Such developments have accentuated the imperative nature of investigating vulnerabilities in those specific authentication methods regarding mobile app security.

Secondly, MT has been successfully applied in a variety of application domains, from web services to embedded systems (CHAN; CHEUNG; LEUNG, 2005; CHAN *et al.*, 2007; ZHOU *et al.*, 2007; MAYER; GUDERLEI, 2006; CHAN; CHEUNG; LEUNG, 2007; CHAN; HO; TSE, 2007; TSE; YAU, 2004; PULLUM; OZMEN, 2012; JIANG; XUXIAN, 2013; NAKAJIMA; BUI, 2016; MURPHY; KAISER; HU, 2008). However, most of the analyzed papers focused on the detection of functional faults, with notable applications in areas such as validation and quality assessment, leaving the potential application of MT for detecting faults related to NFRs largely unexplored (SEGURA *et al.*, 2018).

From the study of Segura *et al.* (2018), which introduced a set of guidelines for the application of MT for detecting faults related to NFRs (e.g., Performance), several subsequent studies have adopted MT for such a purpose (AZIMIAN *et al.*, 2019; JOHNSTON *et al.*, 2019; MAI *et al.*, 2019; AYERDI *et al.*, 2022; RAHMAN; IZURIETA, 2023; CORRADINI; PASQUA; CECCATO, 2023). Azimian *et al.* (2019) employed it to identify known bugs and energy hotspots

[1] See vulnerabilities ranked at (11, 14, 18, 20) from CWE (2021)

[2] See Insecure Communication (M3), Insecure Authentication (M4), Insecure Authorization (M6) (OWASP, 2023b)

(i.e., performance issues) in Android apps. Their approach involved the design of MRs based on known bugs and energy hotspots, which were then applied to Android apps for checking for violations.

Johnston *et al.* (2019) applied the CWE framework to Adobe Experience Platform Launch Tag Manager software specifically for identifying performance anomalies, creating test pages that implemented Adobe tags, and comparing them to test pages using the tags through Experience Platform Launch, thus anticipating functional equivalence while observing potential differences in page-load performance.

Mai *et al.* (2019) designed an approach based on MT concepts to evaluate the security of Web systems. They adopted the technique proposed by Huang *et al.* (2003), according to which an intentionally invalid input (i.e., *source input*) and a valid one (i.e., *follow-up input*) are generated for each MR. They first selected a set of known vulnerabilities that usually can occur on Web systems, then designed a set of 22 system-agnostic MRs to detect them, and, finally, automatically captured security properties from the Web system to check whether the MRs had been violated or not.

Ayerdi *et al.* (2022) introduced an MR pattern called PV (Performance Variation) specifically designed to detect failures in Cyber-Physical Systems (CPS) that integrate software with physical processes. PV simplifies the identification of performance MRs in CPS, providing a valuable solution to the test oracle problem. The authors established MRs to assess banking functions and successfully demonstrated their effectiveness. The approach involves capturing the properties of banking functions, which are characteristics susceptible to compromise during system vulnerabilities, and automating testing procedures using those MRs.

Corradini, Pasqua and Ceccato (2023) expanded upon Mai *et al.* (2019) earlier research by proposing 76 system-agnostic MRs tailored for automating security testing in web systems. Such MRs collectively address 39% of the OWASP security testing activities not automated by existing state-of-the-art techniques.

To the best of our knowledge, this PhD research represents the first endeavor to employ MT for the identification of vulnerabilities in Android apps. It has not only addressed the existing lack in guidelines for such an application, but also advanced the use of MT in the critical domain of mobile app security.

1.2 Research Questions and Objectives

Given the challenges outlined in the problem statement and, as discussed in Section 1.1, this PhD research addresses those challenges by proposing a novel testing technique based on the principles of MT for primarily identifying vulnerabilities in Android mobile apps that are associated with username and password authentication methods. Therefore, the following

Research Question (RQ) and specific objectives have been formulated:

Research Question: *Can the testing technique based on MT concepts enhance the detection of vulnerabilities in Android mobile apps associated with username and password authentication methods?*

Specific Objectives: Towards achieving the overarching goal and effectively addressing the RQ, the following specific objectives have been delineated:

- *Investigation of the use of NFRs dynamic testing techniques in mobile apps:* an extensive Systematic Mapping Study (SMS) on the dynamic testing of NFRs in mobile apps was conducted focusing on NFRs addressed, testing strategies adopted, mobile platforms explored, and supporting tools. The findings are expected to provide a solid foundation for upcoming proposals in this field, offering valuable insights into the current state of dynamic NFRs testing in mobile apps and areas that require further investigations;
- *Characterization of prevalent and known vulnerabilities in mobile apps:* a comprehensive investigation analyzed the prevalent, critical, and known vulnerabilities in mobile apps and involved the extraction of vulnerabilities and security risks from prominent sources, including OWASP (MUELLER; SCHLEIER; WILLEMSEN, 2019; OWASP, 2023b), CWE (CWE, 2023a; CWE, 2021), and relevant scientific studies (AVANCINI; CECCATO, 2013a; LIU *et al.*, 2018a; SHI; WANG; LAU, 2019a; OPASIAK; MAZURCZYK, 2019; WANG *et al.*, 2020a). It identified and selected prevalent, real-world vulnerabilities in Android apps, providing a curated set of vulnerabilities to be used in the application of the proposed testing technique within this PhD research;
- *Elaboration of MRs for the identification of vulnerabilities in mobile apps:* a strategy was defined for crafting MRs and detecting vulnerabilities in mobile apps. It involved a search for relevant scientific studies that offered MRs for identifying vulnerabilities in other application domains, such as the web (MAI *et al.*, 2019; CHALESHTARI *et al.*, 2023a). The study of Mai *et al.* (2019) served as the primary source for the development of a testing approach rooted in MT concepts within the mobile app domain, facilitating the creation of a set of MRs tailored for vulnerability identification specific to mobile apps and enhancing the foundation for the proposed approach;
- *Developing a tool-supported GUI-based MT vulnerability testing technique for Android mobile apps:* A tool-supported testing technique was developed to identify five of the most prevalent and representative real-world vulnerabilities associated with username and password authentication methods, as outlined by OWASP;
- *Execution of a wide experiment:* an experiment validated the vulnerability testing technique based on MT concepts in commercial mobile apps. During the research process, many previous studies proposing security testing techniques were validated with the use of a limited number of open-source apps (MAI *et al.*, 2019; RAHMAN; IZURIETA, 2023; CHALESHTARI *et al.*, 2023b; CHALESHTARI *et al.*, 2023a). Such a limitation led

to the validation of the technique across a wide spectrum of commercial Android apps, ensuring they are widely used and representative of the diverse landscape of mobile app development. The technique aimed at a more comprehensive and robust assessment of the proposed methodology's effectiveness and practicality.

1.3 Contributions

The objectives described and the development of the proposal have led to the following main contributions:

- a pioneering endeavor to harness MT for the identification of vulnerabilities in Android mobile apps;
- a meticulous characterization of a spectrum of real-world vulnerabilities relevant within the framework of user-perceived quality;
- a comprehensive and tool-supported GUI-based MT vulnerability testing technique tailored for Android mobile apps;
- a wide-ranging experimentation with some of the most frequently used Android mobile apps readily accessible via Google Play stores in both Brazil and Italy.

1.4 Thesis Outline

This document is structured as follows: Chapter 2 provides an extensive review of the current state of NFRs, MT technique, and mobile apps testing, establishing the foundational knowledge required for the comprehension of the subsequent chapters; Chapter 3 is devoted to a comprehensive understanding of security testing, deepening the theoretical framework surrounding vulnerabilities in username and password authentication methods towards a critical security context that underpins the entire thesis; Chapter 4 provides an overview of an SMS and the discussion of the results of 56 primary studies that clearly address NFRs dynamic testing in mobile applications, with a special focus on NFRs testing in the domain of security testing for mobile applications; Chapter 5 introduces the MT-based technique developed for detecting vulnerabilities in username and password authentication methods, addressing both design and implementation of the testing environment created for automating the approach and offering insight into the experimental structure and in-depth discussions on the outcomes; finally, Chapter 6 provides the conclusions, findings, implications, and future research directions.

CHAPTER

2

STATE-OF-THE-ART

2.1 Preliminary Remarks

This chapter provides the theoretical background of the main topics addressed in this thesis, namely Non-Functional Requirements (NFRs), Metamorphic Testing (MT) approach, and Mobile Applications Testing. Some parts of the chapter were based on a paper published at ACM Computing Surveys (CSUR) (JUNIOR *et al.*, 2022) and on another paper published at the Proceedings of the 13^{th} International Conference on Software Testing, Validation and Verification (ICST) (JUNIOR, 2020).

- JUNIOR, M. C., AMALFITANO, D., GARCES, L., FASOLINO, A. R., ANDRADE, S. A., DELAMARO, M. (2022). Dynamic testing techniques of non-functional requirements in mobile apps: A systematic mapping study. **ACM Computing Surveys (CSUR)**, 54(10s), 1-38. Available: <https://doi.org/10.1145/3507903>;
- JUNIOR, M. C.. Automated verification of compliance of non-functional requirements on mobile applications through metamorphic testing. In: **Proceedings of the 13^{th} International Conference on Software Testing, Validation and Verification (ICST)**. IEEE, 2020. p. 421-423. Available: <https://doi.org/10.1109/ICST46399.2020.00053>.

The chapter is organized as follows: Section 2.2 presents the essential concepts and characteristics of NFRs, along with the challenges associated with testing approaches applied to them; Section 2.3 provides the core concepts of the MT approach and explores the challenges of its application in the context of NFRs; Section 2.4 covers the fundamental concepts and characteristics of mobile applications testing and discusses the challenges for the application of software testing approaches to NFRs in the mobile applications domain; finally, Section 2.5 is devoted to reflections on the contributions of the background to the understanding of the thesis.

2.2 Non-Functional Requirements: Classification, Challenges, and Testing

Software requirements are often regarded as key indicators of both success and quality of software (GLINZ, 2007). Requirements Engineering (REng) has been widely recognized as one of the most crucial stages in the software development life cycle, playing a vital role in any software development process (AHMAD *et al.*, 2019). When neglected, it can lead to failures and increase time and costs of such development (ECKHARDT; VOGELSANG; FERNÁNDEZ, 2016).

REng community classifies requirements into Functional and Non-Functional (FRs and NFRs, respectively). The former defines the functions, i.e., specifications of the behavior between outputs and inputs, of a system or its components, whereas NFRs, commonly referred to as q.r, specify criteria for the judgement of the operation of a system, rather than specific behaviors. In other words, FRs outline the requirements for a software system to address stakeholders' needs and NFRs determine the degree to which a product or system provides functions that meet those needs when used under specified conditions (PRESSMAN, 2016; SOMMERVILLE, 2011).

The *IEEE Standard Glossary of Software Engineering Terminology* (COMMITTEE *et al.*, 1990) does not define term "Non-Functional Requirement" clearly, although it distinguishes others such as design requirements, interface requirements, and performance requirements. According to Glinz (2007) and Afreen, Khatoon and Sadiq (2016), NFRs can be seen as properties or qualities to be exhibited by a product, encompassing attributes such as appearance, speed, and accuracy. Additionally, they represent system properties that comprise quality functions (e.g., performance, usability, security, among others) (GLINZ, 2007; AFREEN; KHATOON; SADIQ, 2016).

Figure 1 shows an example of a login system for illustrating the difference between FRs and NFRs. A login system is a fundamental component of software applications – users must provide credentials (e.g., username and password) to access secure areas or personalized features within the application. It authenticates users and grants appropriate access privileges based on their credentials.

Below is the FR for this login system:

FR: User Authentication - The login system must allow users to enter their username and password to authenticate their identity.

NFR is related to the system's performance:

NFR: Performance - The login system should respond to user login requests within 3 seconds, providing a seamless and efficient user experience.

As illustrated in Figure 1, FRs specify the functionalities of the software system for

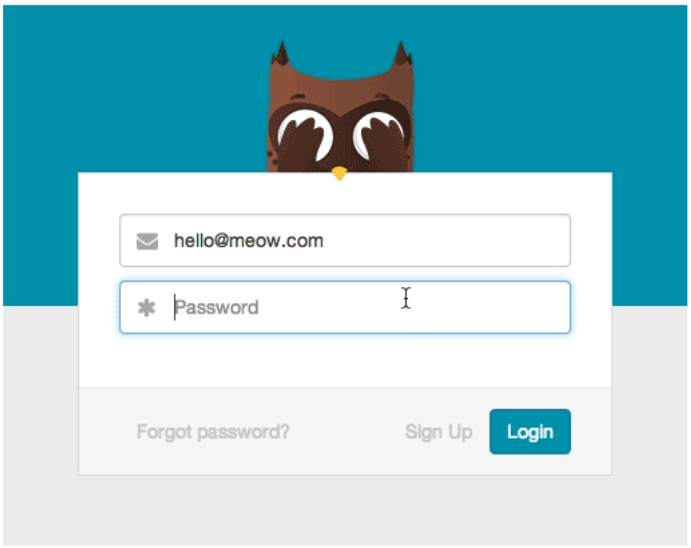

Figure 1 – Example of FR and NFR specification.

Source: Elaborated by the author.

addressing stakeholders' needs. In the example of the login system, FR must enable user authentication through a username and a password. On the other hand, NFRs focus on the quality attributes and constraints that define how well the software system performs its functions. In the login system example, NFR pertains to performance, emphasizing the system's responsiveness and ability to handle user login requests within 3 seconds for an optimal user experience.

2.2.1 Classification of Non-Functional Requirements

NFRs are an essential aspect addressed in international standards within the software and systems quality initiative (ADAMS, 2015). Both previous ISO/IEC Standard 9126 (ISO/IEC 1991) (ISO, 2001) and its replacement, ISO/IEC Std 25010 (ISO, 2011) titled *"Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models"* (ISO, 2011), encompass NFRs, their definitions, and methodologies for measuring them as a fundamental component of any systems development effort (ADAMS, 2015).

As described by Junior *et al.* (2022), ISO/IEC/IEEE 29119-1, titled *"Software and systems engineering-Software testing-Part 1: Concepts and definitions"* (ISO, 2013), strictly correlates NFRs with the quality characteristics (q.c.) of a software product. Such characteristics are defined in SQuaRE, which proposes a model that categorizes software product quality properties into eight characteristics, namely, functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability, and portability. ISO/IEC/IEEE 29119-1 classified requirements into two main categories, namely, FRs and NFRs. The former aligns with the Functional Suitability q.c. outlined in ISO/IEC 25010, whereas NFRs are associated with the remaining seven characteristics (also known as quality attributes, q.a) outlined in SQuaRE.

SQuaRE (ISO, 2011) also defines a set of sub-characteristics for each q.c. , providing a more detailed assessment of the several aspects that define the quality of a software product. In what follows are the descriptions of the NFRs defined by ISO (2011):

- *Performance efficiency:* represents the software's performance in terms of number of resources it uses when running under specified conditions. It includes sub-characteristics like Time Behavior, Resource Utilization, and Capacity;
- *Compatibility:* describes the software system's ability to exchange information with other software products, systems, or components and its performance while sharing the same hardware or software environment with them. It encompasses sub-characteristics like Coexistence and Interoperability;
- *Usability:* indicates the software system's capacity to be used by specific users for achieving well-defined goals effectively, efficiently, and with satisfaction in a given context. It includes sub-characteristics like Appropriateness recognizability, Learnability, Operability, User Error Protection, User interface aesthetics, and Accessibility;
- *Reliability:* reports the software system's ability to execute given functions under specified conditions for a defined period. It involves sub-characteristics like Maturity, Availability, Fault Tolerance, and Recoverability;
- *Security:* defines the software system's capability to protect access to information and data, preventing unauthorized access from individuals or other software systems based on their types and levels of authorization. It includes sub-characteristics like Confidentiality, Integrity, Non-repudiation, Accountability, and Authenticity;
- *Maintainability:* characterizes the software system's ease of modification by maintainers in an effective and efficient manner. Modifications may include corrections, improvements, adaptations to changes in the environment, requirements, and functional specifications. Installing updates or upgrades is also considered a form of maintenance. It involves sub-characteristics like Modularity, Reusability, Analyzability, Modifiability, and Testability;
- *Portability:* expresses the effectiveness and efficiency with which a software system can be transferred to new hardware, software, and operational or usage environments. It encompasses sub-characteristics like Adaptability, Installability, and Replaceability.

ISO/IEC Std 25010 (ISO, 2011) replaced ISO/IEC 9126 model (ISO, 2001), which was discontinued in 2013. Such classification models serve as a basis for evaluating software, establishing a minimum set of quality criteria. Such classification models serve as a basis for evaluations of software, establishing a minimum set of quality criteria. ISO/IEC 9126 had 6 indicators that branched out into 27 sub-characteristics, whereas ISO/IEC 25010 comprises 8 indicators and 31 sub-characteristics.

2.2.2 Problems Associated with Non-Functional Requirements

Classification models for NFRs (ISO, 2001; ISO, 2011) are rarely taken into account in the software development industry (CHUNG *et al.*, 2012). Recent research has indicated little attention has been devoted to NFRs; moreover, NFRs are often poorly understood and not adequately considered during software development (CHUNG *et al.*, 2012; MAIRIZA; ZOWGHI; NURMULIANI, 2010). For instance, they are neither extracted with the same level of detail as FRs, nor rigorously described in requirement documents (MAIRIZA; ZOWGHI; NURMULIANI, 2010). The reasons for those neglects are (i) lack of support tools and (ii) rare reporting of NFRs, even informally.

FRs can be elicited directly from the system users through software resource requests (AHMAD *et al.*, 2019), whereas NFRs can be extracted from user feedback (AHMAD *et al.*, 2019; BEYER *et al.*, 2018; ZOU *et al.*, 2017; ROSEN; SHIHAB, 2016). As an example, users often report issues related to software quality features on social media platforms such as *Stack Overflow*[1] when they feel directly affected by different NFRs (e.g., usability, performance, and security). The shared content contains statements about software development, tools, and product qualities that can help software organizations improve their products (AHMAD *et al.*, 2019; GROEN *et al.*, 2017; ZOU *et al.*, 2017; ROSEN; SHIHAB, 2016).

In real software systems, NFRs are as important as FRs (ECKHARDT; VOGELSANG; FERNÁNDEZ, 2016). The current competitive market demands software products that are not only functionally adequate, where NFRs are considered critic (AMELLER *et al.*, 2012). Moreover, contemporary software systems consider NFRs essential properties to be ensured (e.g., energy efficiency and portability, which are critical features in mobile applications) (RASHID; ARDITO; TORCHIANO, 2015).

Some studies have highlighted and discussed the importance of NFRs in the software development industry (VARA *et al.*, 2011; POORT *et al.*, 2012; CARACCIOLO; LUNGU; NIERSTRASZ, 2014; LÓPEZ *et al.*, 2018). Vara *et al.* (2011) conducted an empirical study in the form of a questionnaire-based survey for gathering insights on the significance of NFRs in the industry. 31 professionals from 25 organizations identified the most important types of NFRs for them and the rationale behind their decisions. The five most selected types were Usability, Maintainability, Performance, Reliability, and Flexibility, and, conversely, those selected as most important were Usability, Maintainability, Performance, and Reliability.

The results provided by Vara *et al.* (2011) were similar to those reported in Poort *et al.* (2012) and Caracciolo, Lungu and Nierstrasz (2014), who analyzed and classified NFRs commonly specified by industry professionals (software architects, developers, project managers, among others). Poort *et al.* (2012) identified Usability, Performance, and Security as the NFRs most commonly considered, whereas Caracciolo, Lungu and Nierstrasz (2014) reported

[1] https://stackoverflow.com/

Performance, Reliability, and Compatibility as the most important in projects.

López *et al.* (2018) conducted a survey to assess how industry professionals handled NFRs in projects and concluded (i) most companies collect data both automatically and manually, (ii) NFRs are managed alongside FRs, (iii) functionality sometimes or often takes priority over quality, and (iv) the most frequently reported critical NFRs in software projects are Reliability, Performance, and Security.

2.2.3 An Overview on Non-Functional Requirements Testing

Although some authors have emphasized the importance of NFRs (GLINZ, 2007; BAJPAI; GORTHI, 2012), a limited number of software testing approaches have addressed them (CARACCIOLO; LUNGU; NIERSTRASZ, 2014; RIBEIRO; TRAVASSOS, 2017; RIBEIRO; CRUZES; TRAVASSOS, 2018). The lack of NFRs testing may be the cause of low-quality software failing to meet user expectations, leading to project failure (RIBEIRO; TRAVASSOS, 2017). An adequate testing of most types of NFRs is challenging due to their nature and, when expressed in non-measurable terms, becomes time-consuming or even impossible.

Caracciolo, Lungu and Nierstrasz (2014) highlighted the use of automated testing is not common in the validation of NFRs – 59% of the study participants reported using non-automated techniques. Automated techniques are commonly adopted for validating NFRs related to user properties (response time, throughput, among others) and security (authorization, authentication, and others). One of the reasons automated validation is not common may be the limited availability of tools that meet the needs of some professionals. Poort *et al.* (2012) demonstrated industry professionals identified obstacles that hinder an early verification of errors associated with NFRs in projects, such as lack of NFR elicitation in the early stages of project development.

Ribeiro and Travassos (2017) conducted a systematic review for identifying NFRs and software testing approaches that deal with testable NFRs and provided research opportunities and a collection of reports on NFRs and software testing approaches. Approximately 224 NFRs were identified, of which 87 were described, and 47 testing approaches were observed. Only eight approaches were empirically evaluated. No testing approach was identified for 11 of the testable NFRs. The results indicate the software testing approaches identified do not cover the most frequent and testable NFRs and some testing approaches for NFRs are not frequently observed. Furthermore, most testing approaches neither are evaluated through experiments, nor fully cover the software testing phases (planning, design, implementation, execution, and analysis), making their use risky.

Ribeiro, Cruzes and Travassos (2018) conducted a case study in three Brazilian organizations for characterizing verification practices for Security and Performance NFRs and claimed most verification techniques used are *ad-hoc*, thus hampering the definition of a criterion for the selection of test cases. As an example, the use of tools with no exact knowledge of the technique

implemented creates uncertainty about their detection capability. Additionally, the number of False Positive (FP) defects identified by security tools can be a problem.

The aforementioned authors claimed no written Performance NFRs could be used as oracles in some cases; therefore, no verification activities were performed for assessing whether the software had met its NFRs, but, rather, for evaluating the system's capability. In some other cases, verification activities were based on subjective or imprecise requirements, i.e., on users' opinions about the system's behavior. Among the challenges for verifying the Security and Performance NFRs cited by Ribeiro, Cruzes and Travassos (2018) are (i) lack of training for dealing with NFRs, (ii) lack of tools and supporting techniques, and (iii) lack of descriptions of NFRs in software documents, or incorrect or incomplete descriptions.

2.3 Metamorphic Testing Approach

The mechanism that judges the correctness of an output or the expected behavior of a particular program being run is known as "oracle" (HOFFMAN, 2001). Oracles verify whether the results from a System Under Testing (SUT) with certain test data are correct. In the context of automated testing, they are crucial for reducing testing costs and improving the quality of software verification and validation (HOFFMAN, 2001).

"Oracle Problem", a problem known to researchers and extensively investigated, arises when the correctness of outputs generated from valid input domain data cannot be judged or its judgement is difficult with the use of practical means (WEYUKER, 1982; BARR *et al.*, 2015).

Chen, Cheung and Yiu (1998) introduced the concept of MT, an approach that has been adopted as an alternative to alleviate the oracle problem. According to the authors, *"Metamorphic testing is a technique conceived to alleviate the oracle problem"*. Unlike conventional testing methods, MT does not verify each specific output, but, rather, the relationships between inputs and outputs from multiple executions of SUT (ZHOU; XIANG; CHEN, 2016). Such relationships are known as MRs, which are relations derived from properties to be upheld by the tested program (CHEN; CHEUNG; YIU, 1998; SEGURA *et al.*, 2016). As claimed by the author, those relationships are based on specific characteristics of the SUT and serve as the foundation for effective testing. Therefore, during the MT implementation, some inputs of the program (referred to as "source inputs") are first generated as source test cases, and then an MR can be used to generate new inputs (referred to "follow-up test cases"). Unlike the traditional methods of checking the test result of each individual test case, MT verifies both source and follow-up test cases, as well as their outputs in the corresponding MR (BARUS *et al.*, 2016). Any violation suggests SUT may have a defect.

2.3.1 Practical Guidelines for the implementation of Metamorphic Testing

Segura *et al.* (2016) illustrated the idea of MT through an example of SP(G, s, d), a program that measures the shortest path between a source vertex *s* and a destination vertex *d* in a graph G. An MR for this program is *if the source and destination vertices are swapped, the size of the shortest path would be equal to* |*SP(G, s, d)*| = |*SP(G, d, s)*|.)|. Let us suppose a source test (G, a, b) is selected by some test case generation method (e.g., random). According to this MR, new follow-up test cases can be easily generated by swapping the source and destination vertices (G, b, a). After executing the program with both test cases, their outputs can be checked against the MR for confirming whether it has been satisfied or not, i.e., if the outputs are equal. An MR violated means MT has failed and the program contains a defect. Depending on both size and complexity of G, the a *priori* knowledge of |SP(G, s, d)| becomes difficult. Therefore, even with no knowledge of the expected value for each test case, MR can reveal the existence of defects in the program. Figure 2 illustrates the example.

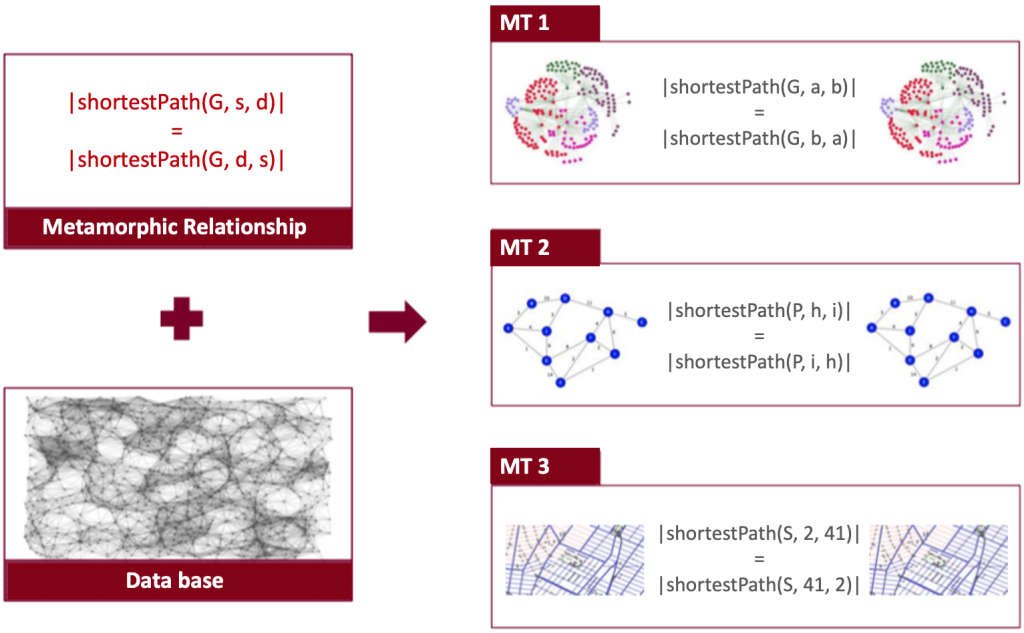

Figure 2 – Example of a metamorphic testing application – Shortest path program.

Source: Adapted from Segura *et al.* (2016).

Another scenario that exemplifies the application of MT is provided through the use of Academic Search Engines (ASEs) (ANDRADE *et al.*, 2019). For instance, a user might be interested in checking the availability of scientific studies on test oracles on *IEEE Xplore*[2] platform. Therefore, the user can formulate a search string that includes terms *"test oracle"* (Q1) and then choose to search in scientific studies that involve the use of test oracles in image

[2]<https://ieeexplore.ieee.org/>

processing. In this case, the user can create a similar search string as *"test oracle"* AND *"image processing"* (Q2). In the first search, a larger number of studies is expected to be returned by the ASE, since the second search string is more restrictive, returning studies that contain both *"test oracle"* and *"image processing"* terms (Q2 $\leq$ Q1). Figure 3 illustrates the example.

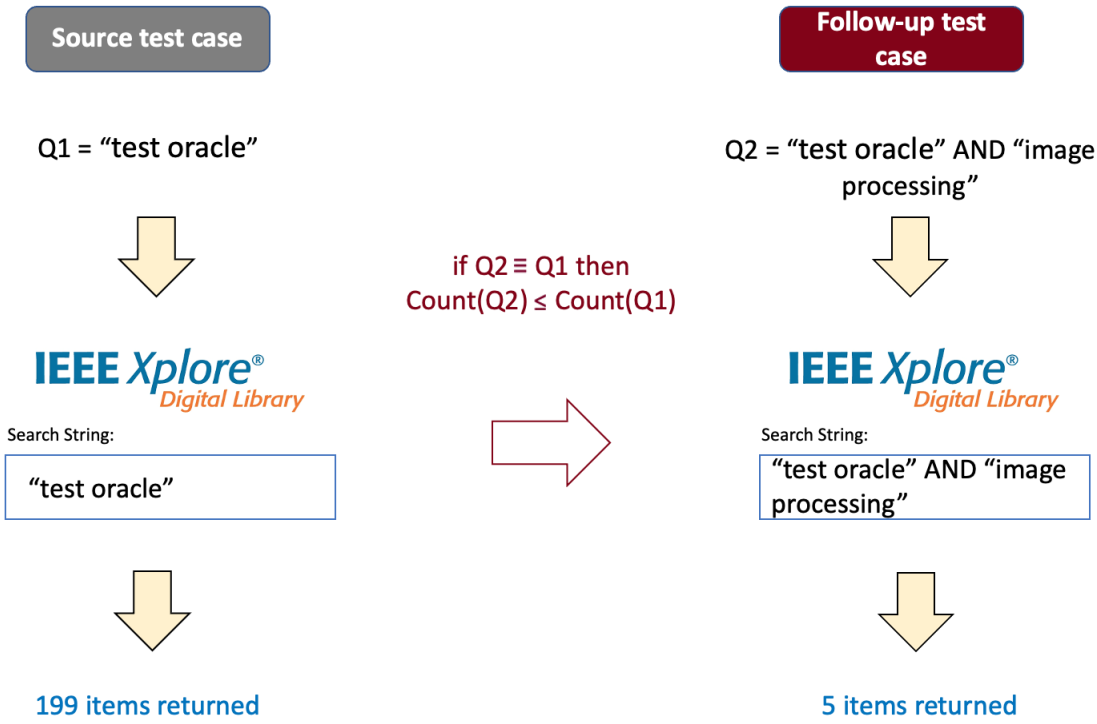

Figure 3 – Example of a metamorphic testing application – Academic Search Engines.

Source: Elaborated by the author.

Based on the aforementioned examples, Segura *et al.* (2016) provided a general MT process for testing a program `P` implementing a function `f` and an example of its application (SEGURA; ZHOU, 2018), which consists of the following four steps:

(i) identification of the properties of `f` and their representation by MRs;
(ii) generation or selection of a test suite, a.k.a. *source test cases*, I_1, which can be defined by a traditional testing technique such as random testing or model based;
(iii) use of MRs to generate a new test suite, I_2, defined as *follow–up test cases*;
(iv) execution of I_1 and I_2 and checking of whether respective outputs O_1 and O_2 violate or not MRs. A violated MR means `P` contains errors.

MT has been effectively applied in several application domains. Through an analysis of studies of MT published between 1998 and 2015, Segura *et al.* (2016) identified 12 different application domains, of which the most popular are web services and applications (16%), followed by computer graphics (12%) and embedded systems (10%). Domains from other areas

such as financial software, optimization programs, and encryption programs (21%) were also identified. However, only 4% of the studies reported results in numerical programs, although this seems to be the predominant domain used for illustrating MT concepts.

2.3.2 Challenges for the use of Metamorphic Testing

MT simplifies system evaluation, making it more practical for systems in which the oracle problem occurs. Chen *et al.* (2018) described some advantages of applying MT, namely, simple concept, straightforward implementation, ease of automation (given the availability of MRs), and low cost and, although they are not unique, MT is one of the few techniques that encompass all of them (CHEN *et al.*, 2018).

Despite the advantages of MT, Chen *et al.* (2018) and Segura *et al.* (2016) pointed out some challenges for advancing research on MT, of which the following stand out:

- Understanding of empirical studies for a unified understanding of MT: the increasing number of real-world software that uses MT indicates the acceptance of the approach; however, the literature lacks a comprehensive evaluation of its overall effectiveness. Although several experimental studies have adopted Mutation Testing (MUT) to assess the defect-detection effectiveness of MT, most of them have focused on a specific application domain and used a single measure (e.g., mutation-based metrics) for evaluating effectiveness;
- Systematic identification and selection of MRs: MRs are key to MT, alleviating the oracle problem. Although many MRs have been identified for various application domains, most of them were detected in an *ad-hoc* and *non-systematic* manner;
- Effective test case generation: MT effectiveness depends on the set of MRs and metamorphic inputs used, whereas follow-up test cases depend on source test cases. As observed by Segura *et al.* (2016), 57% of source test cases in previous studies were generated randomly and 34% were created from existing ones. Therefore, an investigation on the impact of source test cases on the effectiveness of MRs is a field yet to be explored;
- Evaluation of the feasibility of using thresholds for avoiding False Positives (FP) and False Negatives (FN): MRs may sometimes be violated with no indication of a real defect if MT aims to evaluate NFRs, thus resulting in an FP. On the other hand, MRs may also produce FNs, where the relationship is satisfied despite the program having a defect. Strategies for assessing the truth of FPs and FNs, especially regarding non-functional MRs, are essential.

2.3.3 Metamorphic Testing in the Context of Non-Functional Requirements

Several studies have reported a vast set of articles relating MT to its successful application in various domains, from web-services to embedded systems (CHAN; CHEUNG; LEUNG, 2005; CHAN; CHEUNG; LEUNG, 2007; CHAN; HO; TSE, 2007; CHAN *et al.*, 2007; ZHOU *et*

al., 2007; MAYER; GUDERLEI, 2006; TSE; YAU, 2004; PULLUM; OZMEN, 2012; JIANG; XUXIAN, 2013; NAKAJIMA; BUI, 2016; MURPHY; KAISER; HU, 2008; SUN *et al.*, 2023). However, most of such articles focus on FR, with notable applications in areas such as validation and quality assessment, leaving the potential application of MT for detecting issues related to NFRs largely unexplored (SEGURA *et al.*, 2018).

Segura *et al.* (2017) claimed most MRs are defined for deterministic programs in which the relationship is either satisfied, or violated for certain inputs (e.g., `merge([2, 3], [1, 5]) = merge([1, 5], [2, 3])`). On the other hand, measuring non-functional properties such as execution time, memory consumption, or energy usage is inherently non-deterministic. For instance, the battery energy consumed by a mobile application can vary from one execution to another due to device workload, communication issues, or automatic updates. In practice, MRs can sometimes be violated with no indication of a performance defect, resulting in a false positive (FP) (SEGURA *et al.*, 2017).

Towards illustrating the use of MT as an effective approach for verifying compliance with defects associated with Performance, let us consider the following example presented by Segura *et al.* (2018): *"some Chrome users report unexpected levels of memory consumption when loading images of different sizes. It was expected that rendering large images would consume more memory than rendering small images. However, due to issues with the garbage collector, if a small image is loaded after a larger image, memory usage increases"*. Inspired by this error, the following MR could be defined:

$$\texttt{M(loadImg(img}_1\texttt{))} \geq \texttt{M(loadImg(img}_2\texttt{))}$$

where `M` represents the memory consumed and img_2 is an image derived from img_1 but with a smaller size due to a cropping or reduction of its quality.

The application of MT for verifying compliance with NFRs is still an open research topic. In one of the few attempts to use MT for that purpose, a framework was proposed to test software modules to be deployed on wireless sensors regarding functional correctness and energy efficiency (CHAN *et al.*, 2007). The authors generated MRs using the idea adjacent wireless sensors should behave similarly. In another study, Chen *et al.* (2016) applied MT for cybersecurity testing for detecting security-related issues in software obfuscation systems.

Segura *et al.* (2018) established a proof of concept for the use of MT on Performance. Examples of software and case studies of software product lines demonstrated the feasibility of the technique in detecting performance errors. Al-tekreeti, Abdrabou and Naik (2019) proposed a model-based test generation methodology to assess the impact of interactions between wireless network quality and application configuration on the behavior of mobile network application performance. However, they observed the performance model was computationally intensive for one of the application groups used, which made the inversion problem solution costly. Therefore, the authors used MT to mitigate the cost of test oracles.

Mai *et al.* (2019) proposed an approach based on MT concepts to evaluate the security of Web systems. They adopted the technique designed by Huang *et al.* (2003) according to which for each MR, an intentionally invalid input (i.e., *source input*) and a valid input (i.e., *follow-up input*) are generated. In this sense, they authors first selected a set of known vulnerabilities that usually can occur on Web systems, designed a set of 22 system-agnostic MRs to detect these vulnerabilities, and finally automatically captured security properties from the Web system to check whether the MRs had been violated or not.

Rahman and Izurieta (2023) significantly extended Mai *et al.* (2019) by introducing a substantial expansion of the MRs catalog. They meticulously developed 54 additional system-agnostic MRs, thereby amplifying the original collection from 22 to a comprehensive total of 76. To showcase the practical efficacy of the expanded catalog, they employed automated testing procedures within two well-known web systems, namely, Jenkins and Joomla. Remarkably, the augmented approach demonstrated its keen ability to detect a substantial 85% of vulnerabilities present within those systems.

Chaleshtari *et al.* (2023a) employed MT to evaluate banking software functionalities based on their inherent properties. They devised a comprehensive set of 11 MRs meticulously categorized into three primary groups aligned with fundamental banking operations, namely, Deposit, Withdrawal, and Transfer. MRs capture the essential properties of banking functions—specifically the characteristics that become compromised when the system faces potential risks.

The aforementioned studies demonstrate the significant interest of the community in using MR concepts as an effective approach to uncover performance issues, hence, those associated with NFRs.

2.4 Mobile Application Testing: Concepts, Challenges, and Trends

A mobile application is defined as software developed for the current generation of mobile devices known as smartphones (MUCCINI; FRANCESCO; ESPOSITO, 2012). Muccini, Francesco and Esposito (2012) defined a mobile application as mobile software (i.e., applications that run on electronic devices) that, in addition to user input, is also context-sensitive, i.e., it adapts and reacts to the context in which it is executed performing physical environment detection and context-triggered actions, for instance. Amalfitano *et al.* (2013) defined it as self-contained software designed for a mobile device and that performs specific tasks for mobile users.

Mobile applications have gained great popularity in recent years and mobile devices have powerfully entered our daily life in a growing worldwide market where 7,516 billion smartphone users are expected in 2026 (O'DEA, 2020). Regarding mobile operating systems, Android is the

most diffused one, since it is installed on 70% of the devices and 27.7% of them are equipped with iOS (LARICCHIA, 2022). Towards an idea of the diffusion of mobile applications (a.k.a. apps) installed on Android devices, let us consider over 111.3 billion apps were downloaded in 2021 from Google Play Store, the official apps market for Android operating system (CECI, 2022).

In general, mobile applications can be classified into three categories, namely, native apps, web-based apps, and hybrid mobile apps. As claimed by Kirubakaran and Karthikeyani (2013), native mobile apps are implemented and executed on mobile devices with limited resources and driven by user inputs. In contrast, Web applications are hosted on a server and users access them over the Internet through a web browser installed on their mobile device (AMALFITANO *et al.*, 2013). Finally, hybrid mobile apps use native codes for specific platforms for providing a client to the user and access device functionalities; however, their main logic is written as a web application and loaded dynamically at runtime (COPPOLA; MORISIO; TORCHIANO, 2019).

Due to the popularity of mobile devices and applications, software testing and quality assurance activities have become fundamental in this application domain (NAGAPPAN; SHIHAB, 2016) and application developers and companies make significant efforts to providing high-quality apps and maintaining a competitive edge (NAGAPPAN; SHIHAB, 2016). As discussed by Wasserman (2010), mobile applications are not free from defects and new approaches from SE are required for testing them.

Software testing for mobile applications refers to the various types of testing applied to different types of applications (native, web, and hybrid) that run on mobile platforms using well-defined software testing methods and tools. The aim is to ensure quality in functions, behaviors, performance, and service quality, as well as in attributes such as mobility, usability, interoperability, connectivity, security, and privacy (GAO *et al.*, 2014). Recent research on mobile application testing has aimed at solutions to technical platform-related issues. Neto *et al.* (2016) identified the following main fields on which mobile application testing has concentrated:

- Functional and Behavioral Testing: validates service functions, Application Programming Interfaces (APIs), external behaviors of systems, UIs and their gestures, location-based functions, user profiles, system data, and user data;
- Structural Testing: traverses the different possible execution paths of a code and ensures quality of the application under test. Items that can be monitored during program execution with the application of the testing technique generally check and validate performance characteristics;
- Quality of Service (QoS) Requirements Validation: evaluates data load, performance, reliability/availability/privacy, scalability, and system data throughput;
- Usability Testing: evaluates content and alerts on user interface, user operation flows, and scenarios, media, and support for gesture-based interactions;

- Compatibility and Connectivity Testing: evaluates application compatibility with different web browsers, platforms, and connectivity with communication networks.

2.4.1 A Comprehensive Analysis of Mobile Applications Testing

Janicki, Katara; and Pääkkönen (2012) conducted a pioneering survey in the field of software testing for mobile applications. It was a semi-supervised and group-administered survey involving 49 mobile software engineers towards the identification of expectations and challenges associated with automatic test generation and execution. It also detected obstacles faced due to the introduction of automatic test tools in the industry and explored the use of metrics and reports for persuading companies to adopt that technology.

Starov *et al.* (2015), Sahinoglu, Incki and Aktas (2015), and Porras, López and Coronas (2015) made significant contributions in the field of mobile application testing. Starov *et al.* (2015) provided an overview of the key challenges in mobile application testing, discussing cloud testing issues and exploring cloud services and testing-as-a-service resources that might enhance mobile application testing, covering several types of mobile testing features. Sahinoglu, Incki and Aktas (2015) focused on mobile application testing, relating it to different test levels (e.g., system, acceptance, unit, component, and integration) and q.c's (e.g., compatibility, concurrency, conformance, performance, security, and usability) while describing the way various issues on the validation of those apps were addressed.

Porras, López and Coronas (2015) analyzed 83 primary studies, providing an extensive overview of automated testing approaches for mobile applications, testing techniques, and empirical assessments. They also investigated the main challenges related to the automated testing of mobile applications. Among the main approaches identified were model-based testing, capture/replay, model-learning testing, systematic testing, fuzz testing, random testing, and scripted based testing. Notably, they observed a growing number of proposals for automated mobile app testing by 2015.

One year later, Zein, Salleh and Grundy (2016) conducted a systematic mapping study, organizing and categorizing evidence gathered from 79 primary studies. The study revealed essential requirements for mobile software development, such as early elicitation of test requirements during the development process, research on real-world development environments, definition of testing techniques for ensuring application lifecycle compliance, focus on mobile service testing, and development of benchmarking studies for security and usability testing. Kaur and Kaur (2018) conducted the first secondary study for identifying and understanding techniques that estimate testing efforts in mobile applications. They summarized several characteristics that distinguished mobile software/applications from traditional software (e.g., desktop or web-based).

Secondary studies in mobile application testing have gained significant attention over

the past two years (TRAMONTANA *et al.*, 2019; ALMEIDA; MACHADO; ANDRADE, 2019; YA'U *et al.*, 2019; AL-AHMAD *et al.*, 2019; KONG *et al.*, 2019; LUO *et al.*, 2020). Luo *et al.* (2020) provided an overview of simulation methods, including data-driven and model-based techniques, for testing mobile context-aware applications. The study discussed the implementation and deployment of each method by testing tool developers and mobile application testers. In their systematic mapping study, Almeida, Machado and Andrade (2019) identified and discussed the state-of-the-art of tools that automate the testing of Android context-aware applications. By surveying 68 primary studies, they found 80 tools, of which only a few were oriented towards testing context-aware characteristics in Android apps. The study highlighted a lack of tools supporting automatic generation or execution of test cases in high-level contexts and asynchronous context variations.

Regarding cloud domain, the mapping conducted by Ya'u *et al.* (2019) surveyed 23 primary studies for identifying testing approaches used in mobile applications. The authors found diverse methods for testing GUI (Graphical User Interface), compatibility, functionality, and security characteristics; however, only a were supported by automated testing tools. The study also revealed a lack of portability among mobile platforms and limited exploration of TaaS (Testing as a Service).

Two recent studies (TRAMONTANA *et al.*, 2019; KONG *et al.*, 2019) investigated automatic testing for mobile applications. From an analysis of 131 primary studies, Tramontana *et al.* (2019) examined automatic testing tools based on supported testing activities, characteristics of the techniques presented, and evaluation methodologies adopted to validate them. Kong *et al.* (2019) offered an overview on existing Android testing approaches analyzing 103 primary studies and focusing on the identification of testing approaches, test concerns, and levels addressed by such approaches, and on the way they were built and validated. Based on the findings, the authors proposed a taxonomy of android app testing, considering testing objectives (e.g., NFRs), test targets (e.g., parts of the App to be tested), test levels (e.g., integration, system, and unit), and test techniques, such as methods (e.g., model-based and search-based ones), testing environments (e.g., emulator and real device), and testing types (e.g., white, black, or gray boxes).

More recently, Junior *et al.* (2022) conducted a comprehensive study on testing techniques for mobile applications, focusing on a broad set of NFRs, including performance, security, usability, portability, reliability, and compatibility, among others. The study considered various mobile platforms, such as Android, iOS, Blackberry, LG, and Windows Phone, and analyzed the involvement of both academic community and industry in such an important research field. It also characterized the existing techniques and tools for NFRs testing in mobile applications and identified future trends in the field.

2.4.2 Challenges Associated with Mobile Applications Testing

Despite the importance of applying software testing activities to mobile applications, mobile applications impose some additional requirements less commonly found in traditional software applications (NETO *et al.*, 2016). Interaction with other applications, connectivity, diverse input methods, context sensitivity, heterogeneity of hardware configurations and software platforms, security, user interface, and energy consumption limitations are some of those peculiarities. Most of such characteristics, defined as NFRs, can be subject to testing, since they are fundamental aspects of a mobile application (NETO *et al.*, 2016).

Several studies have emphasized the challenges and limitations frequently encountered in the testing activities for mobile applications (JOORABCHI; MESBAH; KRUCHTEN, 2013; VASQUEZ; MORAN; POSHYVANYK, 2017; VASQUEZ *et al.*, 2017; CRUZ; ABREU; LO, 2019). Joorabchi, Mesbah and Kruchten (2013) conducted a survey for understanding the main challenges faced by developers during the development of applications for different mobile devices and observed support for automated testing was very limited for native mobile applications. Moreover, the available tools and emulators did not support essential features for mobile testing, such as mobility, location services, sensors, or different gestures and inputs. Regarding methods that tested native mobile applications, the authors reported 64% of the participants tested a mobile application manually, 31% adopted a hybrid approach (combining manual and automated testing), and only 3% applied fully automated testing. Additionally, 80% indicated developers tested their own applications, 53% had dedicated testing teams, and 27% relied on testers still in training.

Vasquez, Moran and Poshyvanyk (2017) assessed the current state and challenges in mobile application testing. Despite the abundance of techniques and tools available for automated testing, manual testing continued to be commonly used due to factors such as personal preferences, organizational constraints, and lack of essential functionalities in tools. Other reasons that hindered the adoption of automated testing in mobile applications included (i) diversity of common input scenarios in mobile applications (GPS locations, orientation changes, gestures, among others), (ii) time and budget constraints related to mobile applications testing practices, and (iii) lack of specific test oracles for mobile applications. The results of the study were similar to those of Vasquez *et al.* (2017), who evaluated the understanding of developing mobile applications testing activities in Android projects through a survey. The authors identified the participants' preference for automated testing, highlighting some of their reasons, namely, (i) changes in requirements, (ii) lack of time for test decisions and process, (iii) size of mobile applications, (iv) lack of knowledge about automated testing tools and techniques, (v) learning curve and usability of available automated testing tools, and (vi) cost of maintenance of automated testing artifacts.

Cruz, Abreu and Lo (2019) investigated the understanding and the challenges faced by mobile applications developers regarding software testing. Data were collected from multiple

sources, such as F-droid[3], Github[4], and Google Play Store[5] and the authors observed most published mobile applications were not covered by automated testing. Developers relied on manual testing to ensure a proper functioning of their applications – such testing is known to be less reliable and increase technical debt. On the other hand, a comparison of the use of automated testing year by year showed a significant increase among new applications. Furthermore, there is a statistically significant and substantial relationship between use of automated testing and number of minor code issues that appear in a project.

According to the aforementioned studies, despite the large number of techniques and tools for automated testing in mobile applications, developers and testers continue to test their applications manually due to (i) lack of understanding and support for automated testing, (ii) limited project time and budget, (iii) specific characteristics of mobile devices, among other reasons. An interesting detail observed by Vasquez, Moran and Poshyvanyk (2017) is the lack of specific test oracles for mobile applications. Therefore, current tools and practices heavily rely on developers and testers to manually verify expected outcomes and manually code oracles using assertions or exceptions when automation APIs are employed.

2.4.3 Non-Functional Testing in Mobile Applications

Kirubakaran and Karthikeyani (2013) and Muccini, Francesco and Esposito (2012) highlighted the following peculiarities of mobile applications that make the testing activity challenging: limited energy, memory, and bandwidth, rapid changes in context and connectivity type, constant interruptions caused by system and communication events, need for adaptation in input interface for a wide range of different devices, limited time-to-market, and high multitasking and communication with other applications. Most of such peculiarities are related to NFRs.

Maia and Rocha (2019) conducted an SMS to identify the most referenced NFRs in the field of mobile applications. They considered the characteristics and sub-characteristics reported in ISO 25010 (ISO, 2011) and identified NFRs not described in it. The four most commonly referenced NFRs were Usability, Performance Efficiency, Functional Suitability, and Reliability, whereas among the sub-characteristics, the most frequently cited were Appropriateness Recognisability, User interface aesthetics, and Accessibility for Usability, Confidentiality for Security, Functional correctness for Functional Suitability, and Fault tolerance for Reliability. The most frequently referenced q.c's of NFRs not described in ISO/IEC 25010 (ISO, 2011) were Information Quality, Data Persistence, and Sense of Community.

Based on the results reported by Maia and Rocha (2019), Maia, Gonçalves and Rocha (2019) conducted a survey to identify the NFRs most relevant for mobile applications. The authors asked the participants to select their favorite application category and answer questions about

[3] <https://f-droid.org/>
[4] <https://github.com/>
[5] <https://play.google.com/>

NFRs, considering only the specific selected app category. According to 500 valid responses, the authors concluded the most commonly used app category was messaging apps, for which the participants selected Confidentiality and Functional correctness as the most relevant NFRs. In a more general analysis, the most pertinent sub-characteristics were Functional Correctness, Utility, Confidentiality, Interface Visibility, Navigation, and Information Quality.

Although some studies have defined software testing techniques to assess security (OPASIAK; MAZURCZYK, 2019), energy consumption (JABBARVAND; MALEK, 2017), and usability (KLUTH; KREMPELS; SAMSEL, 2014) in mobile applications, some limitations hinder a more comprehensive, effective, and practical approach - for instance, automated testing in this context. Therefore, when developers and the industry adopt automated testing techniques in mobile devices, costs and efforts involved in the production of the applications increase (JOORABCHI; MESBAH; KRUCHTEN, 2013; KOCHHAR *et al.*, 2015).

Nagappan and Shihab (2016) explored current and future research trends across various stages of the software development life cycle in the context of mobile applications. Regarding NFRs, they emphasized challenges and future directions for testing approaches focused on security and performance. (i.e., Performance). Regarding energy efficiency (i.e., Performance), they discussed the lack of accurate estimations of energy usage. Furthermore, they suggested future research should identify practical ways to optimize energy usage in applications and develop approaches to be adapted to mobile platforms other than Android. As for security in mobile applications, they reported the challenge of static software analysis, in which most approaches encounter a high rate of FPs, and a lack of data for statically distinguishing secure codes from insecure ones. Finally, they outlined some promising areas for future research in security, including advances in the state-of-the-art of static analysis and gaining of insights into the reasons for developers to write vulnerable codes in mobile applications.

Typically, NFR descriptions in software projects can be derived from requirements documents and social media platforms. Towards a deeper understanding of the nature of NFRs in mobile applications, Wang, Liang and Lu (2018) conducted an exploratory study by analyzing user reviews. The database comprised 1278 sentences randomly collected from 4000 sentences from iBook (iOS) and WhatsApp (Android) applications (2000 sentences from each app). As a result, users reported quality issues mostly related to Reliability NFR (45.9%), followed by Usability NFR (33.7%). Quality issues on Performance Efficiency NFR (9.5%), Portability NFR (9.3%), and Security NFR (1.6%) were also reported. The proportions indicate a need for greater attention to NFRs in the development of mobile applications.

More recently, Junior *et al.* (2022) performed an SMS towards a broad discussion on NFRs dynamic testing in mobile applications. It focused on addressed NFRs, adopted testing strategies, explored mobile platforms, and supporting tools, identifying (i) security, performance, and usability testing of mobile applications were of most interest to the software engineering community, (ii) Android was the mostly adopted mobile platform, and (ii) most proposed

techniques were tool-supported.

2.5 Final Remarks

This chapter has addressed crucial concepts and insights necessary for a comprehensive understanding of this thesis. As discussed, recent research has highlighted an inadequate attention devoted to NFRs, i.e., a deficiency that frequently leads to an oversight of those pivotal aspects during the software development process (CHUNG *et al.*, 2012; MAIRIZA; ZOWGHI; NURMULIANI, 2010). Unlike FRs, which can be directly derived from system users, NFRs often require insights from user feedback, rendering their evaluation more intricate and dynamic efforts (AHMAD *et al.*, 2019). Despite the significance of NFRs, the software testing methodologies developed to address them remain limited (RIBEIRO; TRAVASSOS, 2017; RIBEIRO; CRUZES; TRAVASSOS, 2018) and NFRs testing is far from straightforward and often plagued by the "oracle problem", which is particularly prevalent in this context. The scarcity of testing approaches for NFRs remains an ongoing challenge, potentially leading to the development of subpar software and project failures (CARACCIOLO; LUNGU; NIERSTRASZ, 2014; RIBEIRO; TRAVASSOS, 2017; RIBEIRO; CRUZES; TRAVASSOS, 2018).

MT has emerged as a promising alternative testing approach to mitigating the “Oracle Problem”, since it has been successfully applied in various domains (CHAN; CHEUNG; LEUNG, 2005; CHAN; CHEUNG; LEUNG, 2007; CHAN; HO; TSE, 2007; CHAN *et al.*, 2007; ZHOU *et al.*, 2007; MAYER; GUDERLEI, 2006; TSE; YAU, 2004; PULLUM; OZMEN, 2012; JIANG; XUXIAN, 2013; NAKAJIMA; BUI, 2016; MURPHY; KAISER; HU, 2008). Yet, the potential of MT to address NFRs issues remains largely untapped (SEGURA *et al.*, 2018). Notably, recent studies conducted by Chan *et al.* (2007), Al-tekreeti, Abdrabou and Naik (2019), and Mai *et al.* (2019) have embarked on exploring the integration of MT in NFRs testing and applied the MT approach to assess diverse aspects, such as energy efficiency in wireless sensors (CHAN; CHEUNG; LEUNG, 2007), security concerns in software obfuscation systems (CHEN *et al.*, 2016), performance issues of wireless networks in mobile network applications (AL-AHMAD *et al.*, 2019), and security vulnerabilities in Web Systems (MAI *et al.*, 2019; CHALESHTARI *et al.*, 2023a).

In a final analysis, this chapter has provided profound insights into the realm of mobile applications, their testing approaches, and the intricate landscape of NFRs testing. Since mobile applications inherently demand additional requirements not commonly found in traditional software (e.g., heightened security measures, energy efficiency considerations, hardware-software heterogeneity, and refined user interfaces) (NETO *et al.*, 2016), NFRs have emerged as common in those applications and performance, Security, and Usability have significantly appeared as the most critical NFRs (NAGAPPAN; SHIHAB, 2016; MAIA; ROCHA, 2019; MAIA; GONÇALVES; ROCHA, 2019; JUNIOR *et al.*, 2022).

The chapter has laid the foundation for a comprehensive exploration of NFRs and their testing challenges, setting the stage for further investigations into strategies and approaches towards ensuring both quality and reliability of mobile applications.

CHAPTER

3

FUNDAMENTALS OF SECURITY TESTING

3.1 Preliminary Remarks

Security requirement represents one of the most critical and widely considered NFRs in software systems (POORT *et al.*, 2012; LÓPEZ *et al.*, 2018; CARACCIOLO; LUNGU; NIER-STRASZ, 2014; RIBEIRO; CRUZES; TRAVASSOS, 2018), being fundamental in contemporary software ones (BAJPAI; GORTHI, 2012). However, as emphasized by Ribeiro, Cruzes and Travassos (2018), most security verification techniques employed are often *ad-hoc*. In the realm of mobile apps, such applications frequently require access to security and privacy-sensitive features on mobile devices, such as security settings and personal data. Consequently, vulnerabilities, i.e., security faults, can jeopardize the application's quality and lead to unwarranted attacks or unauthorized access (MCAFEE, 2017). Despite the existence of various security testing approaches (OPASIAK; MAZURCZYK, 2019), security testing in mobile apps remains predominantly a manual, costly, and intricate process (RIBEIRO; CRUZES; TRAVASSOS, 2018).

This chapter provides a comprehensive understanding of security requirements and testing, describing the background to such requirements, covering key concepts, and exploring the challenges associated with those critical NFRs. Concepts of testing techniques commonly employed in security are elucidated through descriptions and discussions and a theoretical foundation is offered for a prevalent, significant, and selected class of vulnerabilities, specifically, authentication and authorization in Android apps.

The discussions presented are based on the following paper, submitted to the Journal of Systems & Software (JSS) (JUNIOR *et al.*, 2023):

- JUNIOR, M. C., AMALFITANO, D., VISONE, B., FASOLINO, A. R., DELAMARO, M.. A Metamorphic Testing Technique for Detecting Authentication Vulnerabilities in

Android Mobile Apps. **Submitted to the Journal of Systems & Software (JSS)**, 2023. p. 1-28

The chapter is structured as follows: Section 3.2 explores security requirements and testing; Section 3.2.1 delves into the key concepts of security testing tailored specifically to Android apps; Section 3.3 describes username and password authentication, one of the most prevalent authentication methods in Android apps, and covers the common security threats associated with it; finally, Section 3.4 reports the contributions of the foundational knowledge presented to the understanding of the overarching thesis.

3.2 A Guide to Security Requirements and Testing

According to the *System and Software quality models defined in the ISO/IEC 25000 SQUARE series* (ISO, 2011), security requirement, a.k.a. "security characteristic", represents *the degree to which a software product or a software system protects information and data so that users or other systems have access to data appropriate to their types and levels of authorization*. As defined by ISO (2011), security characteristic can be further broken down into more specific attributes, often referred to as "quality sub-characteristics", which include aspects such as confidentiality, integrity, non-repudiation, responsibility, and authenticity and illustrate the various facets through which a security characteristic manifests itself within a software system.

As elucidated by Felderer *et al.* (2016), security characteristics are presented in two fundamental forms, namely, positive requirements and negative requirements. The former entail an explicit definition of the anticipated security functionalities of a security mechanism. For instance, a positive requirement concerning the security property of authorization can be phrased as "User accounts are disabled after three unsuccessful login attempts." On the other hand, negative requirements specify the actions to be avoided by the application. For a same security property, a negative requirement can be formulated as "The application should not be compromised or misused for unauthorized financial transactions by a malicious user." The positive and functional perspective on security requirements aligns with software quality standard ISO/IEC 9126 (ISO, 2001), which classifies security as a functional quality characteristic. In contrast, the negative, non-functional perspective is endorsed by ISO/IEC 25010 (ISO, 2011).

Most modern software systems encompass functions related to financial transactions and management of personal data (RIBEIRO; CRUZES; TRAVASSOS, 2018), which often demand a high degree of assurance, leading to a growing interest in unauthorized access and potential attacks (MCAFEE, 2017). Furthermore, security aspects permeate several layers of an application, including network, operating system, and application layer. Each layer presents its distinct set of security threats and corresponding security requirements. For instance, the network layer is susceptible to threats such as denial-of-service attacks or network intrusions,

where an attacker inundates the target with numerous packet requests, overloading it so that it cannot respond to any request. At the operating system level, a diverse range of malware poses significant threats. Lastly, at the application level, threats often revolve around the proper verification of authenticated sessions, which is typical of Android apps, so that unauthorized users can potentially gain access to the application's private functionalities.

Security testing has emerged as a process that evaluates whether the software's features align with the expected security criteria, as highlighted by Felderer *et al.* (2016). In simpler terms, *security testing verifies and validates the software system's requirements concerning security aspects, including but not limited to confidentiality, integrity, availability, authentication, authorization, and non-repudiation* (FELDERER *et al.*, 2016). As claimed by Tian-yang, Yin-Sheng and You-yuan (2010), the literature offers two primary approaches to security testing, namely, security functional testing and security vulnerability testing. The former validates the correct implementation of specified security requirements, encompassing both security properties and mechanisms. In contrast, security vulnerability testing uncovers unintended vulnerabilities within the software under examination - vulnerabilities represent errors in a system's design, implementation, operation, or management (TIAN-YANG; YIN-SHENG; YOU-YUAN, 2010; POTTER; MCGRAW, 2004). Those vulnerabilities are typically exploited by attackers who aim to execute improper actions in the system (e.g., collection of sensitive user data).

Security vulnerability testing serves as a proactive measure for identifying vulnerabilities and preventing their exploitation by malicious individuals (POTTER; MCGRAW, 2004). They can usually be approached from two distinct perspectives, namely, (i) Simulation of Attacks or Penetration Testing and (ii) Risk-Based Testing and Vulnerability Identification, described in what follows.

- **Simulation of Attacks or Penetration Testing:** security experts or testers deliberately simulate real-world attacks or perform penetration testing towards compromising the security of a system. They might intentionally overload a system with excessive traffic or requests for assessing the way it responds under stress. The objective is to identify vulnerabilities and weaknesses that might disrupt the system's availability;
- **Risk-Based Testing and Vulnerability Identification:** identify risks in a system and generate tests that uncover vulnerabilities associated with those risks. Testers perform an initial risk assessment to detect critical areas of an application or system and subsequently design tests specifically for evaluations of those high-risk areas. As an example, if an application handles sensitive financial data, the risk-based approach prioritizes the testing of authentication and encryption mechanisms.

The two perspectives provide comprehensive approaches to security vulnerability testing, i.e., one actively seeks vulnerabilities through simulated attacks, whereas the other proactively identifies risks and tailors tests that assess vulnerabilities associated with those risks. Regarding

security testing techniques, Felderer *et al.* (2016) highlighted the four testing techniques commonly used in the security context, namely, (i) Model-Based Security Testing, (ii) Code-Based Testing and Static Analysis, (iii) Penetration Testing and Dynamic Analysis, and (iv) Security Regression Testing, described in what follows.

- **Model-Based Security Testing (MBST)**: a Model-Based Testing (MBT) approach that validates software system requirements related to security properties. It combines security properties like confidentiality, integrity, availability, authentication, authorization, and non-repudiation with a model of the SUT and identifies whether the specified or intended security features hold;
- **Code-Based Testing and Static Analysis:** : involve a detailed analysis of the application's source code and fall under the category of white-box security testing techniques, i.e., they require no executable SUT, and, rather, examine the application's source code to identify vulnerabilities. Such techniques can be particularly valuable when applied in the early stages of software development, detecting vulnerabilities prior to deployment. Code review can be conducted through manual inspection or automated processes. Automated code reviews are commonly referred to as Static Code Analysis (SCA) or Static Application Security Testing (SAST) and rely on specialized tools that automatically analyze a software component's program code (such as an application or library) for identifying and reporting potential security issues, including vulnerabilities;
- **Penetration Testing and Dynamic Analysis:** in contrast to Code-Based Testing and Static Analysis techniques, Penetration Testing and Dynamic Analysis belong to the black-box security testing techniques category. They require no access to the source code or other development artifacts of the SUT and, instead, are executed while the SUT is running. During a penetration test, an application or system is evaluated from an external perspective through simulations of conditions similar to those of a real attack by a malicious third party;
- **Security Regression Testing:** a fusion of regression and security testing and appropriately called security regression testing, its primary objective is to guarantee alterations made to a system do not compromise its security. Consequently, it is of paramount importance and the interest in its application has been steadily growing. In essence, regression testing techniques ensure modifications made to existing software do not lead to unintended consequences to its unaltered and modified portions, assuring the altered components continue to function as expected.

3.2.1 Security Testing in Android Applications

Numerous testing techniques for security testing in Android apps have emerged and can be broadly categorized into three primary classes, namely, (i) static testing, in which testing approaches scrutinize software development artifacts (e.g., requirements, design, or code) without

executing them, (ii) dynamic testing, which, according to Felderer *et al.* (2016), assess apps by observing their execution during runtime, and (iii) hybrid testing, which represents a fusion of static analysis and dynamic testing methods (FELDERER *et al.*, 2016).

Static testing techniques have been used mainly for the detection of sensitive data leaks (ARZT *et al.*, 2014; ZHANG; TIAN; DUAN, 2021), Inter-Component Communication (ICC) vulnerabilities (OCTEAU *et al.*, 2013; OCTEAU *et al.*, 2015; BAGHERI *et al.*, 2021; WANG; YANG; MA, 2023), permission misuse (JIANG; XUXIAN, 2013; LI *et al.*, 2014), and code verification (PAYET; SPOTO, 2012; LORTZ *et al.*, 2014). Arzt *et al.* (2014) proposed FlowDroid, an approach that performs static taint analysis on Android apps with a context-, flow-, field-, object-sensitive and lifecycle-aware analysis. Similiarly, Zhang, Tian and Duan (2021) designed a static taint analysis tool, called FastDroid, that supports efficient taint analyses of apps on a large scale, maintaining high precision and recall.

Payet and Spoto (2012) developed a tool called Julia, which performs a code verification through formal analyses of the Android app. Octeau *et al.* (2013) presented Epicc, a static analysis technique for identifying ICC vulnerabilities in Android apps, addressing the need for scalable and accurate ICC discovery. Lortz *et al.* (2014) proposed a tool called Cassandra, which checks if an Android app complies with its privacy requirements.

Octeau *et al.* (2015) introduced the concept of Multi-Valued Composite (MVC) constant propagation, addressing the need for inferring all possible values of complex objects in a program, considering correlations between fields. The authors presented a COAL solver, a generic solver that infers MVC values in an interprocedural, flow and context-sensitive manner.

Bagheri *et al.* (2021) designed FLAIR, a technique that efficiently analyzes Android apps for ICC vulnerabilities in response to incremental system changes. It replaces the standard Alloy relational logic analyzer, used by COVERT as a backend analysis engine, with its new formal analyzer that automatically adapts to incremental system modifications, ensuring the continued expressiveness and analyzability of the initial framework. Wang, Yang and Ma (2023) introduced IAFDroid, a comprehensive analysis framework that merges static and taint analysis techniques to achieve a heightened level of accuracy in identifying collusion attacks within Android apps.

Dynamic testing techniques have been mainly used for the detection of memory leaks (SHAHRIAR; NORTH; MAWANGI, 2014a; LIANG *et al.*, 2018a) and ICC vulnerabilities (SALVA; ZAFIMIHARISOA, 2015a). Liang *et al.* (2018a) proposed an analysis system called AppLance, which runs the app through its UI (User Interface), logs the execution data, and finally analyzes them to identify memory leaks. Similarly, Shahriar, North and Mawangi (2014a) pdesigned an approach that runs the app using a fuzz testing method based on known memory leaks vulnerabilities for finding them on Android apps. Salva and Zafimiharisoa (2015a) introduced APSET, a tool that takes an ICC vulnerabilities scenarios set, formally expressed with models, generates test cases from the models, and finally checks if the app has ICC vulnerabilities.

Hybrid testing techniques have been largely applied on Android apps for detection of vulnerabilities associated with SSL/TSL (KNORR; ASPINALL, 2015a; LIU *et al.*, 2018a; WANG *et al.*, 2020a), Single Sign-On (SSO) (SHI; WANG; LAU, 2019b), ICC (ROMDHANA *et al.*, 2023), Inter-Application Communication (IAC) (AVANCINI; CECCATO, 2013a; GUO *et al.*, 2014a; HAY; TRIPP; PISTOIA, 2015a; ALHANAHNAH *et al.*, 2020), sensitive data leaks (RUMEE; LIU, 2015; KENG *et al.*, 2016a), WebView (HASSANSHAHI *et al.*, 2015), and assessments in general (MAHMOOD *et al.*, 2012; YEH *et al.*, 2014). Similarly, Liu *et al.* (2018a) and Wang *et al.* (2020a) designed an approach that first performs a static analysis for identifying the vulnerable code, generates an ACG (Activity Control Graph) to trace a path between main and the vulnerable activities, and finally runs the potentially vulnerable activity for detecting SSL/TSL vulnerabilities on Android apps. Knorr and Aspinall (2015a) proposed an approach that performs a static analysis for identifying SSL/TSL vulnerabilities and then the Android apps is executed for confirming the existence of vulnerabilities. Shi, Wang and Lau (2019b) first generated Finite State Machines (FSMs) from the app code and then designed test inputs from the models for detecting SSO vulnerabilities.

Towards addressing ICC vulnerabilities, Romdhana *et al.* (2023) developed RONIN, an approach based on deep reinforcement learning for dynamically generating exploits for ICC vulnerabilities in Android apps. By manipulating Intent parameters through a sequence of actions guided by positive, neutral, or negative feedback, RONIN learns to build Intents that expose specific vulnerabilities.

In the realm of IAC vulnerabilities detection, Guo *et al.* (2014a) proposed a compositional approach that integrates static and dynamic automated testing techniques to detect vulnerabilities arising from inter-component messaging. The method begins with a static analysis to obtain initial detection results and parameter information, followed by dynamic testing to automatically generate attack cases and simulate attack behaviors. Hay, Tripp and Pistoia (2015a) presented a comprehensive testing algorithm for IAC vulnerabilities, identifying eight vulnerability types and proposing solutions for automated discovery challenges such as path coverage and custom data fields. The approach, called IntentDroid system, leverages lightweight platform-level instrumentation to recovering IAC-relevant behaviors.

Avancini and Ceccato (2013a) introduced a testing approach for mobile software, focusing on thoroughly testing routines that validate input values from IAC messages. The method suggests automatic test case generation to identify discrepancies between the intended behavior declared by an application and the actual functionalities implemented in its code. Alhanahnah *et al.* (2020) developed DINA, a hybrid analysis approach that identifies and counteracts such attacks, addressing such a challenge by appending reflection and Dynamic Class Loading (DCL) invocations to control-flow graphs and performing incremental dynamic analyses to detect misuse of those features. The authors evaluate DINA's effectiveness in identifying hidden IAC behaviors in real-world apps, demonstrating its potential to enhance security vetting on a large scale.

In the quest to uncover data leaks, Rumee and Liu (2015) designed DroidTest, a method that tests Android apps before their release on the market. Unlike traditional methods that require access to the program's source code, DroidTest is a server-side black-box testing approach that focuses solely on the input and output of the application under test. The testing process involves generating correlated test inputs from existing test cases, whose only difference lies in the input containing sensitive information. By observing variations in the output among correlated test cases, the system identifies potential data leakage. Keng *et al.* (2016a) introduce MAMBA, a testing system that identifies privacy issues in Android apps through path searches of user events within Control Flow Graphs (CFGs) derived from static analyses of app bytecode. It then constructs test cases consisting of user events for directing the app's activity transitions swiftly from the initial activity to specific target activities of interest. The process efficiently uncovers potential accesses to privacy-sensitive data within the apps.

Hassanshahi *et al.* (2015) described an investigation into a type of attack known as "Web-to-Application Injection" (W2AI) targeting Android apps and addressed a relatively unexplored class of attacks, wherein a malicious attacker can exploit vulnerabilities in apps through a malicious website accessed on an Android browser, i.e., WebViews. An automated scanner, called W2AIScanner, identifies and confirms such vulnerabilities. The findings revealed a significant number of vulnerabilities in Android apps, underscoring the need for increased attention and security measures by developers.

Regarding vulnerability assessments by hybrid testing techniques, Mahmood *et al.* (2012) proposed a framework for automated security testing of Android apps on the cloud. It employes numerous heuristics and software analysis techniques to intelligently guide the generation of test cases for boosting the likelihood of discovering vulnerabilities. Yeh *et al.* (2014) introduced CRAXDroid, a pioneering method that identifies impersonation attacks in mobile apps and leverages dynamic code analyses for extracting user interfaces from mobile applications, subsequently scrutinizing the extracted screenshots to flag instances of impersonation.

In spite of a number of testing techniques for security previously describe in this section, they have the following limitations: (i) static testing techniques often report vulnerabilities that may not exist (false positive), (ii) most dynamic and hybrid testing techniques use an exploratory testing technique, i.e., tests are dynamically designed, executed, or modified, and (iii) techniques do not address known and frequent vulnerabilities, such as those reported by CWE and OWASP.

3.3 Secure username and password authentication in Android applications

This section delves into the security woes plaguing username and password authentication in Android apps. Ranked third in the OWASP Top 10 (OWASP, 2023b), such vulnerabilities pose a critical threat. Their omnipresence in mobile apps (OWASP, 2023a) makes them ripe

targets for attack, and reports of exploitation are frequent and alarming. The seriousness is further underscored by the six distinct authentication flaws listed in the CWE[1]. Recognizing this ongoing risk, OWASP's 2023 update highlights the need for urgent investigations and mitigation of those vulnerabilities to securing mobile apps (OWASP, 2023b).

Android apps frequently handle information that requires secure communications with a remote service - typically a backend server - to safely deliver those data. Therefore, apps must integrate authentication services that safeguard both their own integrity and that of users (BIANCHI *et al.*, 2017; MA *et al.*, 2019). Such services encompass authentication and authorization characteristics, of which the former confirms the validity of a transmission, message, or sender, or provides a means to verify an individual's authorization to access specific categories of information. On the other hand, authorization grants access privileges to a user, program, or process (BIANCHI *et al.*, 2017).

Authentication methods are integral components of applications, guaranteeing only authorized individuals can access protected resources, which not only enhances security, but also grants better control over user interactions (BIANCHI *et al.*, 2017; MA *et al.*, 2019). For instance, let us consider an entertainment app like Netflix, which offers a catalog of movies and TV shows. The app tailors both content and quality based on the user's subscription type employing an authentication schema in which users input their credentials, thus granting them access to their personalized profiles and the app's content while maintaining security and user control.

Mobile apps commonly employ one method or a combination of authentication ones, as reported by Bianchi *et al.* (2017), according to whom Android apps frequently adopt the following authentication methods: (i) username and password authentication, (ii) *third-party authentication services*, (iii) *text messages (SMS)*, and (iv) *device information*. The former stands as the most traditional approach for user authentication; the authenticating app directly prompts the user to provide their login credentials, which are transmitted to the app's backend server, where they are rigorously assessed for accuracy. After a successful verification, the server issues a token to the app, which functions as a shared secret string, simplifying authentication for all subsequent interactions between the client and the server.

Third-party authentication services empower users to log in an app using their credentials from platforms such as Google or Facebook. The approach streamlines the authentication process, significantly enhancing user convenience. Once authentication has been completed, the app receives a token and includes it with its requests to the backend, which can then use it to request additional user information from the third- party service, enabling the creation of a more comprehensive user profile (BIANCHI *et al.*, 2017). In contrast, *Authentication services by Text messages (SMS)* enables apps to use SMS-based authentication, in which a one-time code is sent to the user's mobile number. The user must enter it to complete the authentication process. Finally,

[1]See vulnerabilities ranked at (11, 14, 18, 20) (CWE, 2021)

authentication services based on *device information* enable apps to employ device-related data for authentication, which might involve verifying the device's unique identifiers or characteristics for granting access.

De-facto industry standard authorization protocol OAuth 2.0 (OAUTH, 2012) was designed to enable a website, or, in general, an application to access Protected Resources (PR) on behalf of a user. It defines the Abstract Protocol Flow (OAUTH, 2012) describing the flow of interactions client apps, such as web applications, desktop applications, mobile apps, and living room devices should follow to access a user´s PRs.

Figure 4 shows the protocol to be implemented by a client mobile app (a.k.a. client app) installed on an Android Device (AndDev) for accessing the user's PR. The client app first sends an Authentication Request (AR) to the user, a.k.a. Resource Owner (RO) , who replies with an Authorization Grant (AG) . Among the four AG types defined by OAuth 2.0, User and Password Credentials (UPC) are the ones most commonly implemented by developers (MA *et al.*, 2019). Towards gathering the UPC, client apps usually render a Sign-In Screen (SIS) , like the one in Figure 4, where the user inserts the username and the password before tapping on the *SIGN IN* button. The client then sends the AG to an Authorization Server (Auth-Server) , which validates (or not) the UPC. If it is valid, the AR is accepted and Auth-Server replies to the client by sending an Access Token (AT) . The client dispatches the AT to the Resource Server (Res-Server) , which validates (or not) it. If it is valid, Res-Server sends the PR to the client.

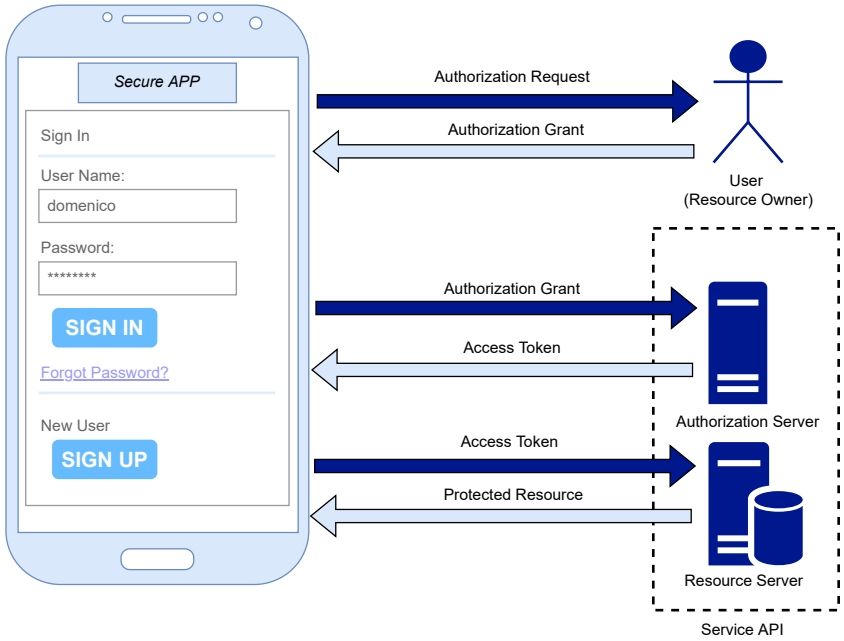

Figure 4 – Abstract Protocol Flow of user authorization and access to protected resources defined by OAuth 2.0.

Source: Adapted from Junior *et al.* (2023).

Netflix is an example of an app using OAuth authorization protocol. A registered user that wishes to use the well-known video-streaming Netflix services must preliminary be authorized

by entering their credentials (i.e., mail/username and password) in the sign-in screen of the app. If the correct credentials have been sent, the Netflix catalog along with the user's preferences are downloaded from the server so that the user can access all services available.

OWASP has defined specific guidelines reported in both OWASP Mobile Application Security Testing Guide (MUELLER; SCHLEIER; WILLEMSEN, 2019) and OWASP Web Security Testing Guide (SAAD; MITCHELL, 2019) for verifying whether the users´ credentials and protected resources have not been intercepted and modified by malicious attackers. The guidelines are summarized in what follows:

(i) *Verification of the SSL certificate:* when a communication is established between a client and its Auth-Server, the connection must be secure and safeguard any sensitive data of the client's traffic. Therefore, OWASP suggests verifying the use of Secure Sockets Layer (SSL) or Transport Security Layer (TSL) cryptography protocols for ensuring confidentiality, integrity, and authenticity of the data exchanged over a secure computer network (WEI; WOLF, 2017). Basically, the protocols exploit encryption algorithms to scramble the data for preventing transmission against Man-in-the-Middle (MITM) attacks (WANG *et al.*, 2020b), in which malicious attackers can capture, view, and modify the network traffic between the app and the server. Towards achieving this security level, the trustness of the SSL/TSL digital certificates should be checked, according to OWASP. SSL/TSL certificate is a type of public-key that binds the ownership details of the APIs exposed by a server to a cryptography key for ensuring a trusted SSL/TSL encryption. Without a trusted SSL/TSL issued by a legitimate third-party Certificate Authority (*CA*)[2], the network traffic cannot be encrypted and becomes vulnerable to MITM attacks. OWASP suggests verifying the use of a Trusted SSL/TLS Digital Certificate (TSSLDC) for encrypting the communication between the client and the server, thus limiting that risk;

(ii) *Transmission of sensitive data over encrypted channels:* OWASP recommends checking the use of HTTPS Connection Channel (HTTPS-CC) , instead of an HTTP Connection Channel (HTTP-CC) for the implementation of an encrypted communication between the client app and the server, thus limiting further MITM attacks;

(iii) *Proper management of the Access Token's lifetime:* the lifetime of the AT should be properly handled towards avoiding an attacker stealing an authenticated session, the user's account, and their personal data (MUELLER; SCHLEIER; WILLEMSEN, 2019). OWASP suggests checking whether two measures that mitigate this security aspect have been implemented. Its recommendations include verification of whether the AT is destroyed after the user has signed out from the app and whether a new one is generated when the user signs in again. Additionally, the *expiration time parameter* of the AT should not be too long towards further improvements in the security of the app. For instance, high-risk apps (e.g., bank apps) usually define short-lived session tokens (from 2 to 5 minutes), whereas

[2] An entity that issues digital certificates

low-risk apps (e.g., education) usually define long-lived session tokens (from 15 to 30 minutes). Access tokens are objects that have an expiration time, a.k.a. Time-to-live (TTL), which is the maximum time interval within which an AT is valid. Before the end of the TTL, i.e., before the AT expires, the client should refresh the AT by asking Auth-Server for a new one;

(iv) *Proper verification of authenticated session:* OWASP recommends verifying whether the app limits access to functionalities and screens - collecting sensible data - only to authenticated and authorized users. If this is not guaranteed, the app is vulnerable to *Authentication Bypass*, in which a malicious attacker can access, read, and modify personal user's data. Moreover, the guidelines suggest checking whether Res-Server verifies if the user is authenticated and authorized every time it requests for PR.

Guidelines not met indicate the applications contain weaknesses that might result in vulnerabilities related to user authentication that can potentially be exploited by malicious attackers.

3.4 Final Remarks

This chapter provided a comprehensive overview of key concepts pertaining to security requirements, with a particular emphasis on security testing. Additionally, it elucidated the principles of testing techniques commonly applied in the domain of security, offering detailed descriptions and insightful discussions, and also established a theoretical groundwork for a crucial and specifically chosen category of authentication and authorization vulnerabilities in Android apps.

As described in the chapter, security requirement is one of the most critical and widely considered NFRs in software systems (POORT *et al.*, 2012; LÓPEZ *et al.*, 2018; CARACCIOLO; LUNGU; NIERSTRASZ, 2014; RIBEIRO; CRUZES; TRAVASSOS, 2018). However, like most NFRs, security requirement and especially security testing are typically overlooked in the software development process (NAGAPPAN; SHIHAB, 2016; RIBEIRO; CRUZES; TRAVASSOS, 2018; JUNIOR *et al.*, 2022), thus increasing faults of security, commonly defined as vulnerabilities, and making software systems vulnerable to cyber attacks.

CHAPTER

4

SECURITY DYNAMIC TESTING TECHNIQUES IN MOBILE APPLICATIONS: FINDINGS FROM A SYSTEMATIC MAPPING STUDY

4.1 Preliminary Remarks

Several secondary investigations in the field of mobile application testing have offered comprehensive and structured analyses of the diverse contributions in such an expansive research domain and have predominantly focused on specific aspects, such as exploration of automated testing methodologies (PORRAS; LÓPEZ; CORONAS, 2015; ALMEIDA; MACHADO; ANDRADE, 2019), automated functional testing for mobile applications (TRAMONTANA *et al.*, 2019; KONG *et al.*, 2019), and meticulous examinations of testing approaches concerning mobile context-aware applications (LUO *et al.*, 2020). Other secondary studies have investigated mobile application testing techniques in general (ZEIN; SALLEH; GRUNDY, 2016) (ZEIN; SALLEH; GRUNDY, 2016). While those reviews often touch upon challenges and contemporary resolutions related to NFRs testing for mobile devices and applications, they have tended to overlook a comprehensive survey of the cutting-edge advancements within this specific domain, as discussed in Section 2.4.1.

This chapter provides an overview of an SMS and a discussion on the results of 56 primary studies that clearly addressed NFRs dynamic testing in mobile applications. Towards discussions focused on the key objective of this PhD work, only the results and discussions directly related to security testing in mobile applications are described. The full content of the article published as a result of the SMS is provided in Appendix A, where readers can delve deeper into the details of the study and its outcomes.

An SMS is a branch of a Systematic Literature Review (SLR), wherein a broader review of primary studies is conducted for identifying evidence and gaps towards guiding future systematic reviews and detecting areas that require further primary studies (KITCHENHAM, 2004; PETERSEN; VAKKALANKA; KUZNIARZ, 2015).

The main goal of the SMS conducted was to promote a broad discussion on the NFRs dynamic testing in mobile applications that focus on addressed NFRs, adopted testing strategies, explored mobile platforms, and supporting tools. To the best of our knowledge, this is the first secondary study specific for NFRs dynamic testing in the mobile applications context. The discussions presented were based on the following paper published at the ACM Computing Surveys (CSUR) (JUNIOR *et al.*, 2022):

- JUNIOR, M. C., AMALFITANO, D., GARCES, L., FASOLINO, A. R., ANDRADE, S. A., DELAMARO, M. (2022). Dynamic testing techniques of non-functional requirements in mobile apps: A systematic mapping study. **ACM Computing Surveys (CSUR)**, 54(10s), 1-38. Available: <https://doi.org/10.1145/3507903>.

The chapter is organized as follows: Section 4.2 details the goals, Research Questions (RQ), and metrics and Section 4.3 provides the results and the main discussions on security testing in mobile applications.

4.2 Goals, Research Questions, and Metrics

The SMS aimed to identify, evaluate, and collect available and relevant studies on dynamic NFR testing techniques in mobile applications. The Goal–Question–Metric (GQM) approach (CALDIERA; ROMBACH, 1994) designed the research and the main goal was refined into three research goals, as shown in Table 1. A set of RQs and the rationale indicating the expected answers were also defined for each goal (see Table 2). The RQs were specified in a generic form towards obtaining, over time, topics already explored and research trends to be investigated, as suggested by Petersen, Vakkalanka and Kuzniarz (2015). Table 3 shows the metrics planned towards answering each RQ and the related data to be extracted from the primary studies.

Table 1 – Goals of the Systematic Mapping.

Goal ID	Goal	Rationale
G1	Characterize the studies proposing techniques for testing NFRs in mobile apps	This SM aims to characterize research in the area of NFR testing techniques for mobile apps.
G2	Describe the techniques from the literature that test NFR in mobile apps.	This SM aims to characterize the NFR testing techniques for mobile apps.
G3	Analyze the tool support of NFR testing techniques for mobile apps.	This SM aims to characterize tools from the literature that support the execution of NFR testing techniques for mobile apps.

Source: Adapted from Junior *et al.* (2022).

Table 2 – Research Questions.

Goal ID	RQ ID	Research Question	By answering this RQ, the researcher will find out
G1	$RQ1_1$	What is the number of articles published per year?	the efforts devoted by the research community over the years for consolidating this research area
	$RQ1_2$	What is the articles count by venue type?	the venues in which studies relevant to this research area have been published.
	$RQ1_3$	What is the articles count by collaboration type?	the collaboration type between academia and industry regarding proposals of techniques for testing NFRs for mobile apps.
	$RQ1_4$	What are the most influential articles?	the most relevant publications in this research area.
	$RQ1_5$	What is the articles count by country?	the leading countries in this research area.
G2	$RQ2_1$	What are the quality characteristics addressed for testing NFRs in mobile apps?	the quality characteristics that have received more interest (or are more critical) to be tested in mobile apps.
	$RQ2_2$	What mobile platforms are tackled by NFR testing?	the tendency of proposals of testing techniques based on specific mobile platforms.
	$RQ2_3$	What testing strategies are adopted for NFRs testing?	the tendency for use of specific strategies for testing NFRs for mobile apps.
	$RQ2_4$	What testing approaches are used to test NFRs in mobile apps?	the direction of the testing approaches adopted for NFRs in mobile apps.
G3	$RQ3_1$	How many of the identified NFRs testing techniques are Tool-supported?	the percentage of tool-supported NFRs testing techniques for mobile apps.
	$RQ3_2$	What are the adoption levels (in %) of "Proprietary", "Freeware", "Free and Open" tools?	the software licenses of the tools that support NFR testing in mobile apps[1].
	$RQ3_3$	What tool types are used to support the execution of NFR testing techniques for mobile apps?	the tendencies of the tool types that support NFRs testing techniques for mobile apps.

Source: Adapted from Junior *et al.* (2022).

Table 3 – Proposed Metrics.

RQ ID	Metric	Extracted Data
$RQ1_1$	Count the number of papers grouped by year	Publication year.
$RQ1_2$	Count the number of papers grouped by Venue	Publication venue $\in$ {Workshop, Symposium, Conference, Journal}.
$RQ1_3$	Count the number of papers grouped by Collaboration Type.	Collaboration type $\in$ {Academic, Industrial, Crosscutting}.
$RQ1_4$	Report the first 10 papers with more citations.	Number of citations
$RQ1_5$	Count the number of papers grouped by country	First author's affiliation country.
$RQ2_1$	Count the number of papers grouped by quality characteristics	Quality characteristics under test.
$RQ2_2$	Count the number of papers grouped by mobile platform.	Mobile platforms tackled.
$RQ2_3$	Count the number of papers grouped by testing approach	Testing approaches adopted.
$RQ2_4$	Count the number of papers grouped by testing strategy	Testing strategy adopted.
$RQ3_1$	Count the number of papers grouped by "tool-supported technique" and "no tool-supported technique"	The testing technique is tool-supported or not.
$RQ3_2$	Count the number of tools grouped by license	Extract the license of each tool identified from $RQ3_1$.
$RQ3_3$	Count the number of tools grouped by type	Derive the type of each tool extracted from $RQ3_1$.

Source: Adapted from Junior *et al.* (2022).

The SMS was motivated by the lack of requirements essential for a comprehensive understanding of the current advancements in platform-independent dynamic testing techniques specifically tailored for mobile applications, with an emphasis on NFRs as opposed to only FRs. The SMS was structured in accordance with the guidelines outlined by Petersen, Vakkalanka and Kuzniarz (2015) and the protocol was established through extensive deliberations among all participating researchers. The following section provides the main results related to security testing in mobile applications.

4.3 Results

As previously described, this section details the findings from the SMS conducted by Junior *et al.* (2022), providing a succinct and comprehensive perspective that harmonizes directly

with the core theme of this thesis, namely, security testing in mobile applications. Table 13 in Appendix A shows all primary studies selected from the SMS.

4.3.1 NFR testing techniques for mobile apps

This sub-section addresses the distribution of the primary studies regarding NFR and year (Subsection 4.3.1.1), mobile platform types covered in the studies (Subsection 4.3.1.2), security testing strategies (Subsection 4.3.1.3), and testing approaches (Subsection 4.3.1.4) aligned with distinct security. More comprehensive and specific outcomes on other types of NFRs can be found in Appendix A.

4.3.1.1 Distribution of NFRs addressed by primary studies

Figures 5 and 6 show the collection of NFRs identified in primary studies, which encompass performance, security, usability, portability, reliability, and compatibility, along with their distribution across different years.

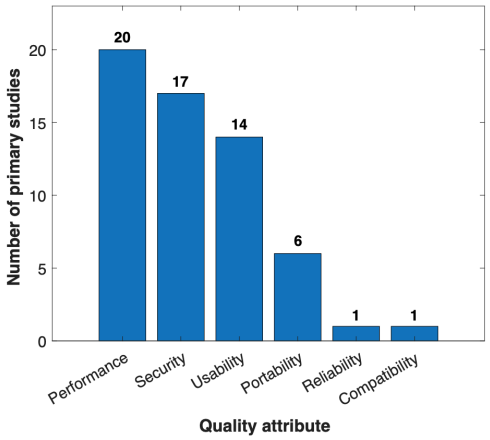

Figure 5 – Quality attributes reported in the primary studies.

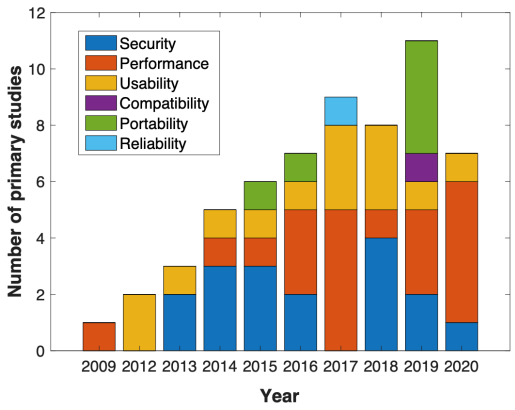

Figure 6 – Quality attributes reported in the primary studies by year.

Source: Adapted from Junior *et al.* (2022).

According to Figure 5, performance (35.7%, 20/56), security (30.3%, 17/56), and usability (25.0%, 14/56) are the most frequently explored NFRs, followed by Portability (10.7%, 6/56). In contrast, Compatibility and Reliability were addressed in only 1.8%(1/56) of the selected studies. As depicted in Figure 6, primary studies focused on security testing, primary studies began in 2013 and performance testing studies followed suit in 2014, with both domains experiencing continuous annual contributions. The only exception was 2017, during which no primary study on security testing was published.

An additional analysis on the primary studies under consideration aimed at understanding the motivations for researchers focusing on specific NFRs. Table 4 provides the reported reasons and corresponding references for security, which is the focus of this thesis. Among the most

prevalent motivations is a recurring theme, namely, *developers often neglect security during mobile app implementation, which, therefore, requires specialized techniques and tools for this NFR*. The statement underscores the critical importance of robust security measures in the design and development of mobile applications. As collectively highlighted in the primary studies, addressing security concerns is pivotal to safeguarding user data, maintaining user trust, and preventing potentially severe consequences of security vulnerabilities. Therefore, targeted approaches and strategies are required for enhancing security testing in mobile apps.

Table 4 – Motivations for addressing Security testing.

NFR	Motivation	References
Security	Developers do not have proper knowledge about security; consequently, they are not careful with this NFR	(AMIN *et al.*, 2019), (RASTOGI; CHEN; ENCK, 2013), (SHI; WANG; LAU, 2019a), (GUO *et al.*, 2014b), (LIU *et al.*, 2018b), (BHATNAGAR; MALIK; BUTAKOV, 2018), (SALVA; ZAFIMIHARISOA, 2015b), (YUSOP *et al.*, 2016), (HAY; TRIPP; PISTOIA, 2015b), (KENG *et al.*, 2016b), (KNORR; ASPINALL, 2015b), (AVANCINI; CECCATO, 2013b), (LEE; VERWER, 2018), (WANG *et al.*, 2020b), (LIANG *et al.*, 2018b), (SHAHRIAR; NORTH; MAWANGI, 2014b), (YANG *et al.*, 2014)
	Security testing is not a simple task and should be investigated	(AVANCINI; CECCATO, 2013b), (LEE; VERWER, 2018), (LIANG *et al.*, 2018b)
	Programmers developing mobile apps that communicate with a server usually do not follow security guidelines to implement SSL/TLS protocols	(SHI; WANG; LAU, 2019a), (LIU *et al.*, 2018b), (WANG *et al.*, 2020b)
	Security testing is neglected with respect to other NFRs; therefore, automated tools must detect vulnerabilities	(GUO *et al.*, 2014b)
	Apps are not tested against known vulnerabilities	(BHATNAGAR; MALIK; BUTAKOV, 2018)
	The usage of third-party apps and frameworks may introduce vulnerabilities	(SALVA; ZAFIMIHARISOA, 2015b)

Source: Adapted from Junior *et al.* (2022).

4.3.1.2 Mobile Platforms

Figure 7 shows the distribution of mobile platforms across primary studies. Android takes the lead, being the focal point in $91.1\%(51/56)$ of the selected studies, followed by iOS, with $4.4\%(2/56)$ representation. A mere $1.8\%(1/56)$ accounts for other platforms, such as Windows Phone, Blackberry, and LG, each one featured in only one study. Two studies adopted an independent testing platform, presenting a distinct approach.

Notably, five studies introduced testing techniques for hybrid mobile applications that amalgamate native and web application elements and are crafted by frameworks like React Native[2] or Flutter[3], facilitating code use across several platforms. The prominence of Android can be attributed to its global prevalence and open-source nature. Its extensive developer community supplies both versioning and issue tracking repositories for applications, providing researchers with convenient access to a plethora of open-source, real-world mobile applications. The environment is conducive for comprehensive analyses of experimental results while representing a typical mobile application usage.

[2] <https://reactnative.dev/>
[3] <https://flutter.dev/>

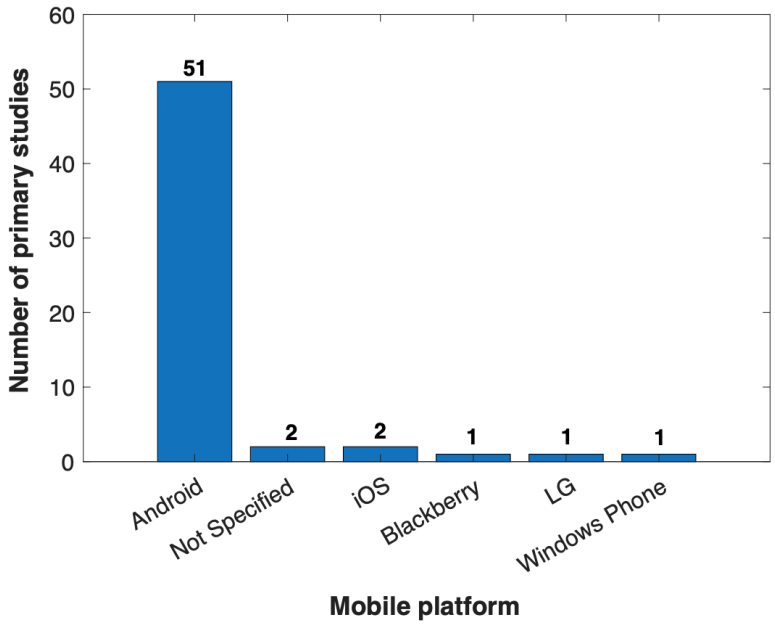

Figure 7 – Mobile platforms reported in primary studies.

Source: Adapted from Junior *et al.* (2022).

Interestingly, all selected primary studies that addressed security testing focused exclusively on Android mobile platform, which can be attributed to its widespread usage and open-source nature, thus enabling a more comprehensive access to the application source code and facilitating in-depth security analyses.

4.3.1.3 Security Testing Strategies

Junior *et al.* (2022) identified the testing strategies employed for Security testing in the examined papers. The distribution of those strategies is illustrated in Figure 8. The hybrid strategy stands out as the predominant approach, accounting for $64.7\%(11/17)$ of cases. Conversely, white box (WB) and black box (BB) were equally represented, accounting for $17.6\%(3/17)$ of cases. For a comprehensive overview of the testing strategies adopted in each study, please refer to Table 12 in Appendix A.

White Box Strategies. Figure 8 shows the exclusive use of WB strategies for security testing $(3/7)$ and the following two distinctive ones can be identified within this category:

*WB*1: *Strategy involving generation of test cases from application models.* Several model types are considered, encompassing Finite State Machine (FSM), which models the overall app behavior (SALVA; ZAFIMIHARISOA, 2015b) and amalgamations of use cases and user interface models (YUSOP *et al.*, 2016).

*WB*2: *Strategy involving injection of mutants into the source code of the tested application.* The nature of the mutants implementing intents that can potentially introduce vulnerabilities is employed for security testing (AVANCINI; CECCATO, 2013b).

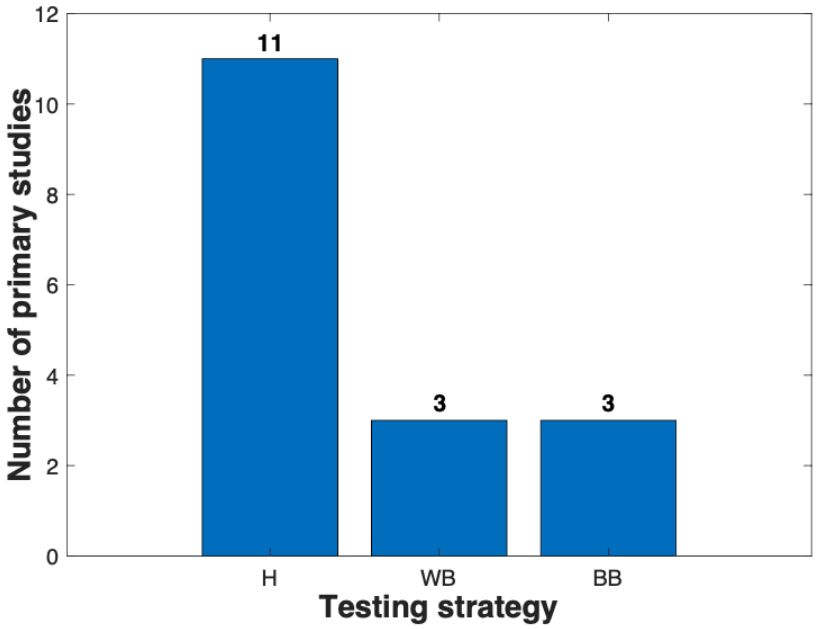

Figure 8 – Testing strategies adopted in Security testing.

Source: Adapted from Junior *et al.* (2022).

Hybrid Strategies. The bar chart in Figure 8 highlights H strategies have emerged in security testing (11/17) and the following two hybrid ones are identified:

*H*1: *Strategy involving generation of test cases from inferred models.* The strategy leverages artifacts and models inferred through app reverse engineering. Specifically, it uses FSM (SHI; WANG; LAU, 2019a; LEE; VERWER, 2018), as well as Control Flow Graph (CFG) models of two types, namely, Control-Flow Graph of Call-Backs (CCFG) (KENG *et al.*, 2016b).

*H*2: *Strategy based on app exploration and code scanning.* The strategy amalgamates two pivotal phases, namely, app dynamic exploration and code scanning. In the former, the app undergoes real-time testing through UI (User Interface) exploration (LIU *et al.*, 2018b; BHATNAGAR; MALIK; BUTAKOV, 2018), targeted API and intent invocation (AMIN *et al.*, 2019), execution of tests emulating attacks (GUO *et al.*, 2014b), or tests tailored to unearth known errors (KNORR; ASPINALL, 2015b). Exploration can be guided by models like CFGs (HAY; TRIPP; PISTOIA, 2015b), or even by source code insights (WANG *et al.*, 2020b; YANG *et al.*, 2014). In the latter phase, the app's code is scrutinized to unveil potential quality issues.

Black Box Strategies. Figure 8 displays a pervasive use of BB strategies in security testing (3/7). An analysis of testing methodologies revealed five distinct BB strategies:

*BB*1: *Strategy founded on app exploration,* which involves dynamic app exploration through the

user interface, where the app is tested in real-time to uncover vulnerabilities(RASTOGI; CHEN; ENCK, 2013; WANG *et al.*, 2020b; SHAHRIAR; NORTH; MAWANGI, 2014b).

4.3.1.4 Approaches for Security testing

According to the Software Engineering Body Of Knowledge (SWEBOK) classification (BOURQUE; FAIRLEY; SOCIETY, 2014), testing approaches can be (i) Based on the Software Engineer's Intuition and Experience (*SEIE*), (ii) Input Domain-Based (IDB), (iii) Code-Based (*CB*), (iv) Fault-Based (FB), (v) Usage-Based (*UB*), (vi) Model-Based (*MB*), and (vii) Based on the Nature of the Application (*BNA*). *SEIE* refers to approaches in which tests are generated according to both skills and knowledge of the software engineer involved in the test process. It comprises two sub-categories, namely, *Ad Hoc* (*AH*) and *Exploratory Testing* (*ET*). *IDB* category encompasses approaches in which tests are generated only by taking into account the properties of the inputs of the software SUT. It is specialized in the following four sub-categories: *Equivalence Partitioning (EP)*, *Pairwise Testing (PT)*, *Boundary Value Analysis (BVA)*, and *Random Testing (RandT)*. *CB* category is comprised of testing approaches in which tests are generated by taking into account the properties of the code implementing the SUT. The category is specialized in two sub-categories, namely, *Control Flow-Based (CFB)* and *Data Flow-Based (DFB)*.

FB includes testing approaches in which tests are generated for revealing specific types of faults, usually defined through a *fault model*. It is further specialized in two sub-categories, namely, *Error Guessing (EG)* and MUT. *UB* characterizes testing approaches that generate tests that resemble the behavior of the human user of the SUT as much as possible. It comprises two sub-categories, namely, *Operational Profile (OP)*, and *User Observation Heuristics (UOH)*. *MB* describes testing approaches that exploit models for deriving tests. A model is an abstract – a formal one in some cases - representative of the SUT. Model-based approaches can be further specified according to the models used to represent the SUT. Therefore, model-based testing approaches include the following four sub-categories: *Decision Tables (DT)*, FSM, *Formal Specifications (FS)*, and *Workflow Models (WM)*. Finally, *BNA* encompasses testing techniques through which tests are designed and execution is based only on the specific nature of the SUT (e.g, object-oriented software, component-based software, concurrent programs, and others).

Testing approaches for security testing

As shown in Figure 9, 7/17 techniques rely on a single approach. Three *MB* approaches generate tests from *FSM* (SHI; WANG; LAU, 2019a; SALVA; ZAFIMIHARISOA, 2015b) or *WM* (e.g., UML (Unified Modeling Language) use case and user interface models) (YUSOP *et al.*, 2016). The two *FB* approaches belong to *EG* and *MUT* subcategories. The former requests the app through tests simulating known attacks (GUO *et al.*, 2014b), whereas the latter injects source code specific vulnerability mutants into the app (AVANCINI; CECCATO, 2013b). *CB* approach of subcategory *CFB* generates test cases from CFG (KENG *et al.*, 2016b), whereas

SEIE, classified as *ET*, tests the app on the fly while UI is explored searching for vulnerabilities (RASTOGI; CHEN; ENCK, 2013).

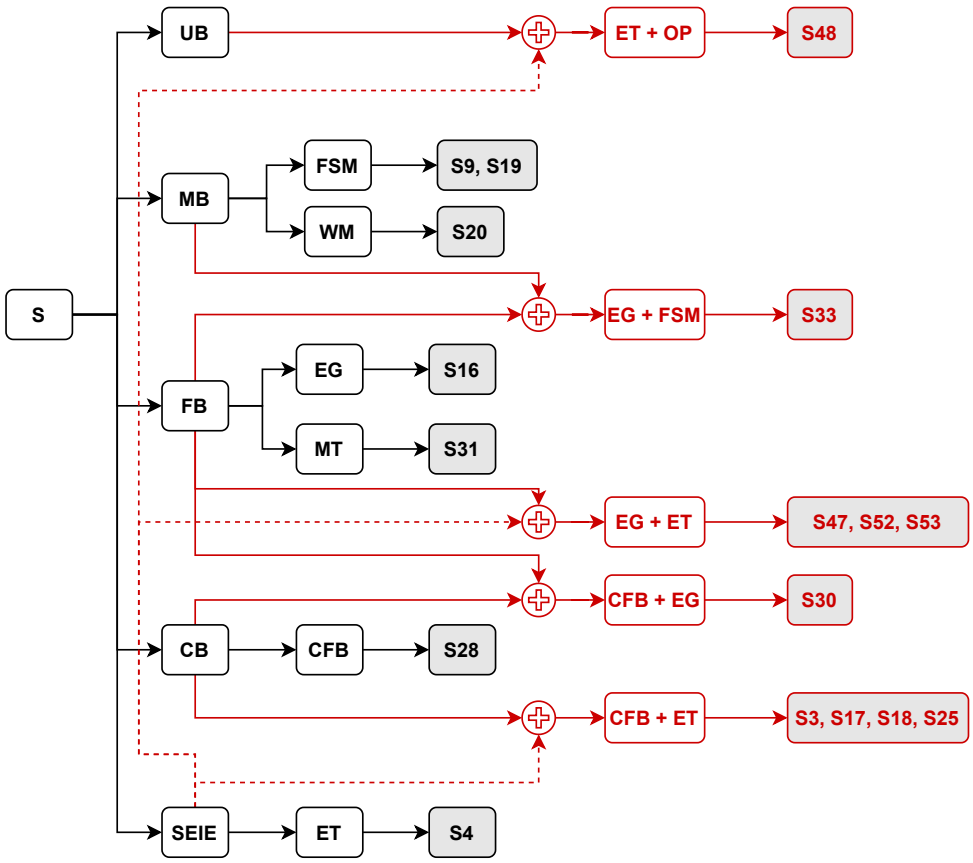

Figure 9 – Mapping of approaches for security testing.

Source: Adapted from Junior *et al.* (2022).

Five different combinations of two approaches were presented in 10/17 primary studies. *FB* was combined with *SEIE* in *EG + ET* subcategory approaches that explore the app for triggering specific vulnerabilities (WANG *et al.*, 2020b; SHAHRIAR; NORTH; MAWANGI, 2014b; YANG *et al.*, 2014). *SEIE* and *CB* are used jointly as *CFB + ET*. The approaches belonging to this subcategory find vulnerability by both exploring the app on the fly and scanning the source code (AMIN *et al.*, 2019; LIU *et al.*, 2018b; BHATNAGAR; MALIK; BUTAKOV, 2018; HAY; TRIPP; PISTOIA, 2015b). *SEIE* was also combined with *UB* as *ET + OP*, which explores the app on the fly for tracing log files that are analyzed so that vulnerabilities can be found (LIANG *et al.*, 2018b). The integration of *FB* and *CB* in *CFG + EG* tests the app using both test cases designed for finding known errors and code scanning (KNORR; ASPINALL, 2015b). Finally, the combination of *FB* and *MB* as *EG + FSM* generates test cases that cover paths of the FSM modeling the app behavior that may reveal potential vulnerabilities (LEE; VERWER, 2018).

4.3.2 Tools for supporting NFR testing for mobile applications

This sub-section examines tool-supported NFR testing techniques, encompassing their categorization by tool support (Subsection 4.3.2.1), licensing model governing the associated software (Subsection 4.3.2.2), and the specific functionality they offer for each quality characteristic testing, as classified by SWEBOK (Subsection 4.3.2.3). All discussions provide a continuous correlation drawn between the findings regarding security testing.

4.3.2.1 Tool support to NFR testing of mobile apps

69.6% $(39/56)$ of the selected studies incorporated or used a tool, whereas the remaining 30.4% $(17/56)$ did not clarify whether the technique presented had tool support. The studies that introduced or leveraged a tool are listed in Table 14 in Appendix A, which includes the tool name (if provided), its type based on the SWEBOK classification adopted, and its accessible URL (if applicable). Notably, $13/39$ studies outlined a tool without explicitly naming it. No tool was shared or employed across multiple studies.

76.4%$(13/17)$ of the selected primary studies that addressed security testing provided a support tool. The emphasis on the use of support tools underscores the growing recognition of the complexity and critical nature of security testing in the context of mobile applications.

4.3.2.2 Testing tool licensing

The distribution of licenses of the tools offered by the authors showed diverse patterns. 69%$(27/39)$ were disseminated with no designated license, 23.0%$(9/39)$ embraced Free Open-Source Software (FOSS) license, 2.6% $(1/39)$ adopted Freeware license, and 5.2% $(2/39)$ were categorized as Proprietary software. Notably, the Proprietary tools in the selection encompassed IntentDroid, developed by IBM, and ZIPT (Zero-Integration Performance Testing), a collaboration between the University of Illinois and Google. Tools operating under a Proprietary license were primarily introduced through primary studies conducted either wholly, or partially by the Industry, whose involvement seemed to impact the accessibility of the tools. In contrast, tools with FOSS and Freeware licenses were typically associated with primary studies developed exclusively by Academic institutions, making their executable files or code readily available for public access.

Interestingly, regarding security testing, 61% of the techniques supported by tools lack accessible sources for the use of the tools, whereas 30.7% adopt FOSS license - only one technique falls under the category of Proprietary software. Such a distribution underlines the increasing demand for open-source solutions within the security testing domain, and most practitioners have recognized the benefits of collaboration and transparency in the critical aspect of mobile application development.

4.3.2.3 Types of NFR testing tools

The tools documented in the literature for facilitating testing procedures for NFRs have been categorized according to their functionalities, which aligns with the classification framework proposed by *SWEBOK* (BOURQUE; FAIRLEY; SOCIETY, 2014). A testing tool can fall within one or more of the following categories: Test harnesses (TH), Test generators (TG), Capture/replay (CR), Oracle/file comparators and assertion checking (OC), Coverage analyzers and instrumenters (CAI), Tracers (T), Regression testing (RT), and Reliability evaluation (RE).

TH encompasses tools that establish elaborate controlled environments using drivers and stubs. Such environments enable tests to be generated and executed on-the-fly or to be stored, planned, and subsequently executed. Additionally, those tools can record the outcomes of test executions. TG belongs to a category that offers sophisticated features for automated test case generation, whereas CR tools are designed to capture user interactions with the tested application frequently through its user interface. The interactions are then stored in a shared repository, promoting their collective replay or the replay of selected subsets. OC tools enable the definition and execution of intricate oracles for automatically identifying failures in the tested application during runtime. CAI involves tools that generate test cases through the analysis of source code. The category includes tools that insert probes into the source code, as well as those that perform source code analysis and reverse engineering. T offers functionalities for logging execution data such as paths, energy consumption, memory usage, and network traffic in text files. RT tools automatically select and execute only the necessary test cases following code changes. Lastly, RE tools provide graphical user interfaces to support test engineers in complex statistical analyses of reliability-related metrics.

Figure 10 illustrates the distribution of tool types employed in the selected primary studies. Table 14 in Appendix A leads to the emergence of a comprehensive understanding, i.e., 30 of the tools extending their support to multiple NFRs. The bar chart reveals *TH* tools are the most frequently employed ones across NFRs testing techniques, *OC*, *TG*, *CAI*, and *CR* tools. Interestingly, they are the least employed type in the context of NFRs testing techniques, which might be attributed to their specific nature and applicability to certain testing scenarios. The distribution highlights the prevalence of *TH* tools, indicating their versatile utility in tackling various NFRs testing challenges.

Figure 11 depicts the number of studies that incorporated specific tool types for Security testing, highlighting the various features and their combinations. A paper ID, wherein the tool was introduced, is attributed to each type or combination. Tools employed for security purposes must encompass *TH*, *TG*, *CAI*, and *OC* features, which demand a comprehensive test harness for exploring the application and identifying vulnerabilities and the capacity to generate security tests based on models. Additionally, the tools should offer the capability to scrutinize or instrument codes that guide app navigation or identify known vulnerabilities.

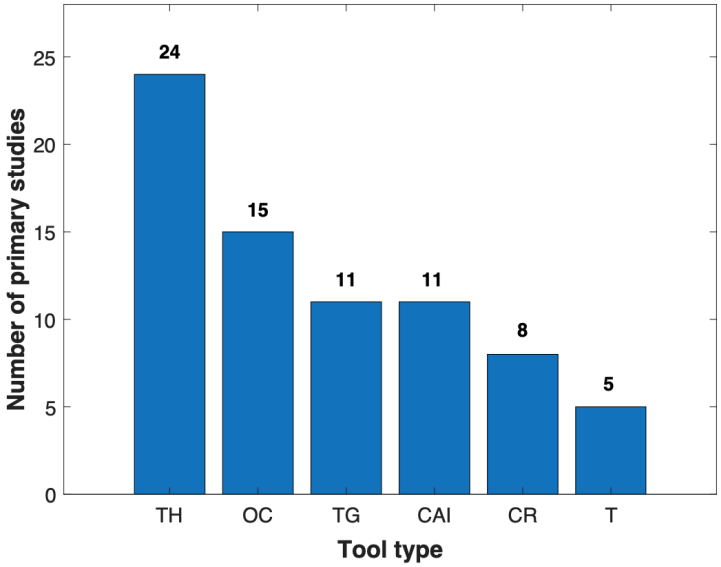

Figure 10 – Types of supporting tools.

Source: Adapted from Junior *et al.* (2022).

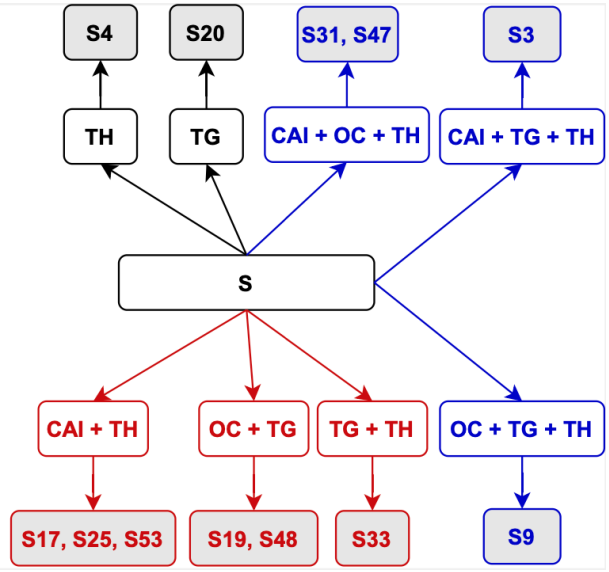

Figure 11 – Tool types addressed by security testing techniques.

Source: Adapted from Junior *et al.* (2022).

4.4 Discussion

This section summarizes and discusses the main findings on security testing on mobile applications reported by Junior *et al.* (2022).

4.4.1 *Security is one of the critical and relevant NFR*

Security retains its status as one of the utmost critical NFRs in mobile applications. Such a significance is reflected in the distribution of NFRs in Figure 5, where security occupies the

second position among the NFRs most frequently tackled in the primary studies selected from the SMS conducted by (JUNIOR *et al.*, 2022). The heightened focus on security is justified by the ever-increasing vulnerabilities associated with mobile applications, emphasizing the imperative need to prioritize robust security testing practices.

4.4.2 Security testing is not a simple and accessible task

As highlighted in Table 4, the motivations underlying the choice for addressing security testing in mobile applications are quite intriguing. Notably, a significant driver is the deficiency in a proper knowledge about security practices. Furthermore, several studies have characterized security testing as an intricate endeavor, offering insights into the reasons behind the often observed lack of robust security testing in mobile application projects and highlighting the complexity of security testing and the need for specialized techniques that effectively address that critical NFR.

A noteworthy observation has arisen from one of the motivations provided in Table 4, specifically highlighting "Apps are not tested against known vulnerabilities", which is a critical gap in the security testing of mobile applications. With the proliferation of new vulnerabilities and threats in the digital realm, failures to assess applications against known vulnerabilities introduce significant risk, requiring such a concern be comprehensively addressed through robust security testing practices.

4.4.3 Android is the most addressed and vulnerable mobile platform in the context of security

As elucidated in Section 4.3.1.2, Android has been the most studied mobile platform in mobile applications and particularly accentuated in the context of security testing. The rationale behind it can be attributed to its widespread usage, status as an open-source platform, and pivotal role in the contemporary mobile app landscape. As shown in Figure 7, Android prevails as a prime target for security exploration, reflecting its both prominence and potential vulnerabilities.

4.4.4 Some tools for security testing are not easily accessible

Section 4.3.2.1 furnishes insight into the availability of tool types employed in NFRs testing techniques. Within the scope of security testing, an intriguing trend becomes evident. $76.4\%(13/17)$ of the studies dedicated to security testing integrated the use of supporting tools. However, a more detailed examination revealed only $38.4\%(5/13)$ of those studies, including those of (AMIN *et al.*, 2019; SHI; WANG; LAU, 2019a; SALVA; ZAFIMIHARISOA, 2015b; HAY; TRIPP; PISTOIA, 2015b; LEE; VERWER, 2018), made those support tools accessible and available for security testing purposes, thus highlighting the varying degrees of tool accessibility

in the specific domain of security testing, which may lead to implications for the replication and expansion of those studies.

4.5 Final Remarks

This chapter elucidated the outcomes of the SMS conducted by Junior *et al.* (2022), focusing on the dynamic testing techniques pertaining to NFRs in the context of mobile applications. The authors conducted a meticulously structured SMS that encompassed various facets of the aforementioned techniques, as well as prominent aspects such as the predominant NFRs targeted in mobile applications, the most extensively explored mobile platform, prevalent testing strategies and approaches, and availability and classification of tool types used in that domain.

CHAPTER

5

A GUI-BASED METAMORPHIC TESTING TECHNIQUE FOR DETECTING AUTHENTICATION VULNERABILITIES IN ANDROID MOBILE APPS

5.1 Preliminary Remarks

This chapter provides a GUI-Based MT technique for the detection of weaknesses that may cause real vulnerabilities affecting username and password authentication methods in Android mobile apps due to an improper implementation of username and password authentication methods, i.e., disregarding the guidelines suggested by OAuth 2.0 (OAUTH, 2012). It also describes a *Metamorphic Vulnerability Testing Environment* that supports the execution of the proposed technique and discusses the results from an experiment in which the technique tested 163 popular Android apps extracted from Brazilian Google Play[1] and Italian Google Play[2]. To the best of our knowledge, this is the first study that exploits MT for finding vulnerabilities in Android mobile apps.

The discussions presented were based on a paper submitted to the Journal of Systems & Software (JSS) (JUNIOR *et al.*, 2023):

- JUNIOR, M. C., AMALFITANO, D., VISONE, B., FASOLINO, A. R., DELAMARO, M.. A GUI-Based Metamorphic Testing Technique for Detecting Authentication Vulnerabilities in Android Mobile Apps. **Submitted to the Journal of Systems & Software (JSS)**, 2024. p. 1-28.

[1] <https://play.google.com/store/apps?hl=pt_BR>
[2] <https://play.google.com/store/apps?hl=it_IT>

The chapter is structured as follows: Section 5.2 details the MT technique for the detection of selected vulnerabilities and outlines the testing environment created for automating it; Section 5.3 addresses the process that defined MRs designed to identify vulnerabilities related to username and password authentication methods; Section 5.4 describes an experiment that validated the MT technique in real-world Android apps; finally, Section 5.6 is devoted to a comprehensive discussion of the chapter, with a particular focus on the experiment's outcomes.

5.2 Metamorphic-based Vulnerability Testing Technique

This section describes the proposed metamorphic-based security vulnerability testing technique displayed in Figure 12. The process relies on the execution of three consecutive steps, namely, **"App exploration"**, **"*Follow-up* test cases generation"**, and **"*Follow-up* test cases execution"**. The figure also shows, for each step, the input required, the output artifacts produced, and the tool that supports its execution. In what follows is a detailed description of each step.

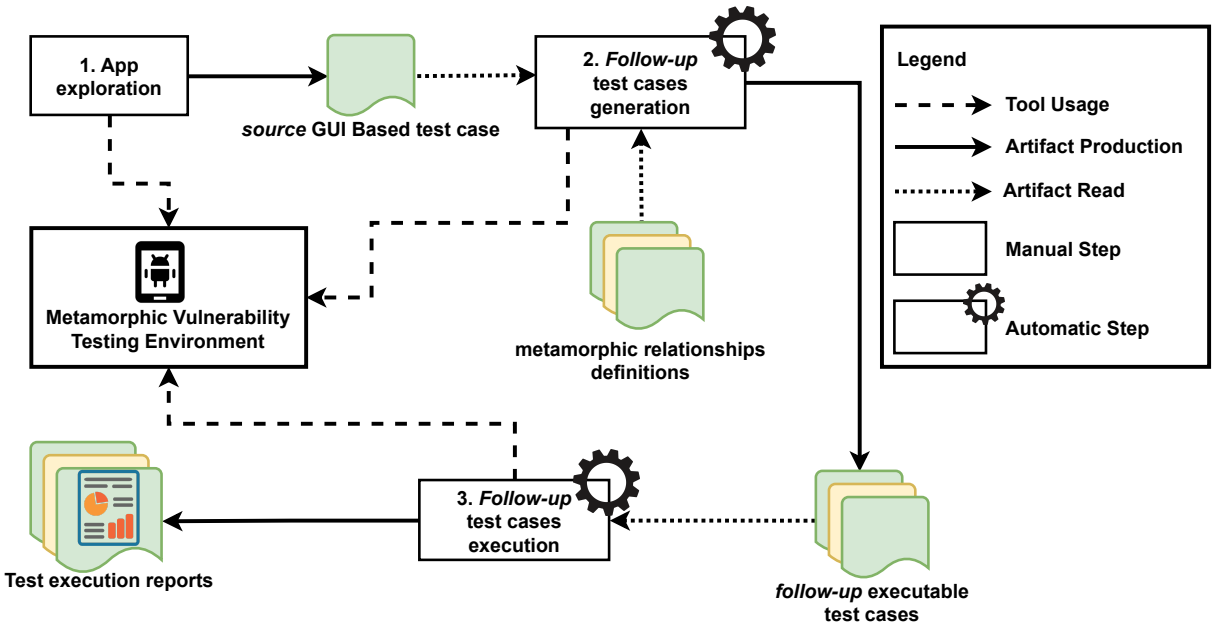

Figure 12 – Metamorphic-based vulnerability testing technique proposed.

Source: Adapted from Junior *et al.* (2023).

1. **App Exploration**: the GUI of the Application Under Test (AUT) is manually explored by the tester. The sequences of user events, along with the descriptions of Graphical User Interface (GUI) resulting from the execution of such events, are stored in a *source* GUI-based test case. GUIs are described in terms of their composing widgets, which are characterized by the values assumed by their attributes (AMALFITANO *et al.*, 2018). The tester must perform the scenario represented in Figure 13, showing a real exploration in Booking application so that a *source* GUI-based test case can be implemented and used for testing the absence of the vulnerabilities considered. According to the figure, the

requested scenario relies on the execution of the four steps described below along with their rationale:

- DEVICE SET UP PRECONDITIONS - APPLICATION UNDER TESTING (AUT) LAUNCH: Android Device (AndDev), where the test technique is deployed, is configured by 1) installing a Trusted SSL/TLS Digital Certificate (TSSLDC) on it and 2) redirecting the communication with both Authorization Server (Auth-Server) and Resource Server (Res-Server) through a controlled proxy. Lastly, AUT is launched for being manually explored.
- USER AUTHENTICATION: Resource Owner (RO) authenticates using its User and Password Credentials (UPC) on the Sign-In Screen (SIS) provided by AUT. At the end of the step, a `Welcome` GUI is rendered on AndDev.
  Rationale: this step supports both MR_1 and MR_4, and, according to them, RO is expected to fail to sign in the application if AndDev does not have a TSSLDC installed, or if UPCs are not transmitted over an HTTPS Connection Channel (HTTPS-CC).
- ACCESS TO PROTECTED RESOURCES (PRS): RO navigates the user interfaces of AUT to reach its PR.
  Rationale: this step aids MR_2 and MR_5. According to MR_2, Access Token (AT) is supposed to be periodically updated, whereas, for MR_5, the user cannot reach PRs without being authenticated.
- USER SIGN OUT - USER AUTHENTICATION: RO first signs out from the AUT and then authenticates again.
  Rationale: this step supports MR_3, since AT is expected to be destroyed after the user has signed out and a new one is expected to be generated when the user signs in again.

2. ***Follow-up* test cases generation:** the MRs listed in Section 5.3.2 are automatically applied to the *source* GUI-based test cases produced in the previous step and a set of six *follow-up* executable test cases are generated. As addressed in the next section, this step produces six *follow-up* executable test cases even if five MRs are defined. Two test cases are produced for MR_1 for checking whether the authentication is refused only when either No Trusted SSL/TLS Digital Certificate (NTSSLDC) , or Self Signed Digital Certificate (SSDC) are installed on the device. The *Follow-up* test cases are actually GUI-based tests that rely on the execution of all or a subset of the steps recorded during the App Exploration Exploration - if needed, the preconditions of the device are modified. Table 5 describes the test cases designed for each MR regarding their preconditions, replayed steps, and oracles to check if the test either passes, or fails.

   A detailed description of the *Follow-up* test cases is summarized in what follows.

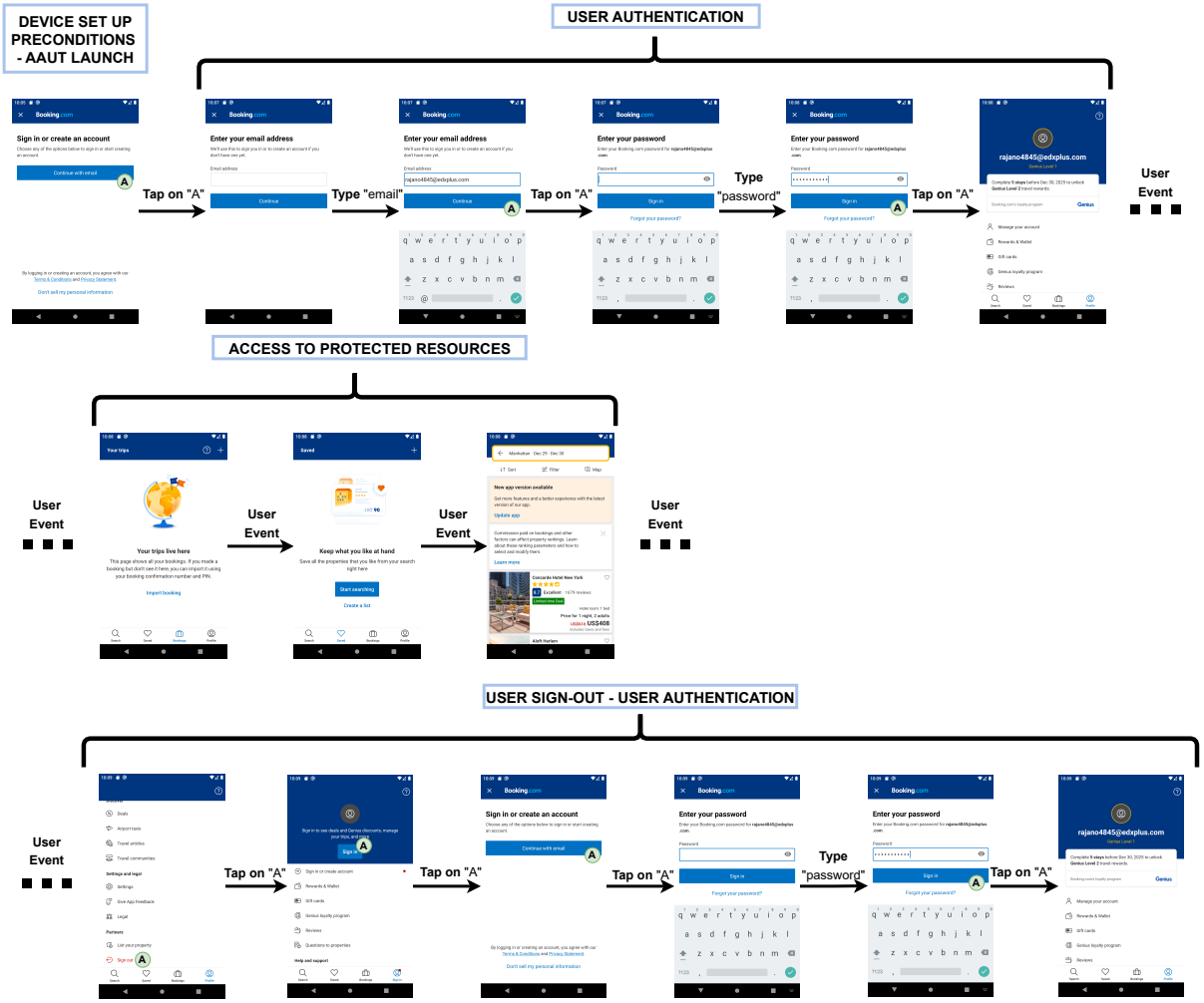

Figure 13 – Example of a *source* GUI-based test case scenario executed on a real Booking application.

Source: Adapted from Junior *et al.* (2023).

FUTC$_1$: designed to support the execution of MR$_1$. USER AUTHENTICATION step is replayed on AndDev where, as preconditions, only NTSSLDC is installed. After the execution of the step, the resulting GUI is described and the description is compared with the one of the user interface obtained in the **App Exploration** step. The test case checks if the two descriptions are different, since the authentication of the user is expected to be refused in this case.

FUTC$_2$: proposed as an alternative way to apply MR$_1$ w.r.t. TC$_1$, this test case is the same as **FUTC$_1$** except that, as preconditions, only SSDC are installed.

FUTC$_3$: introduced in relation to MR$_2$, this test case relies on the same preconditions of the **App exploration** step. It replies to the same user actions for USER AUTHENTICATION and Access Token (AT1) is collected. A one-hour navigation of GUI, which can also be random, is then performed and, at the end, Access Token (AT2) is collected. The two ATs are expected to be different so that the absence of the vulnerability can be checked assessing whether their IDs are not the same.

Table 5 – Design of the six test cases related to the Metamorphic Relationships defined.

MR	Test Case	Preconditions	Test Steps	Test Oracle
MR_1	TC_1	Only NTSSLDCs are installed on AndDev	**REPLAY[**USER AUTHENTICATION (CORRECT UPC)**]**	The resulting GUI is not the one collected in the **App exploration** step, i.e., authentication must be refused
	TC_2	Only SSDCs are installed on AndDev	**REPLAY[**USER AUTHENTICATION (CORRECT UPC)**]**	The resulting GUI is not the one collected in the **App exploration** step, i.e., authentication must be refused
MR_2	TC_3	Same preconditions used in the **App exploration** step	**REPLAY[**USER AUTHENTICATION (CORRECT UPC)**]** - `Collect AT1` - 60 MINUTES OF RANDOM EXPLORATION - `Collect AT2`	`AT1.Id` $\neq$ `AT2.Id`
MR_3	TC_4	Same preconditions used in the **App exploration** step	**REPLAY[**USER AUTHENTICATION (CORRECT UPC)**]** - **REPLAY[**ACCESS TO PR**]** - **REPLAY[**USER SIGN OUT**]** - `Collect AT1` - **REPLAY[**USER AUTHENTICATION (CORRECT UPC)**]** - `Collect AT2`	`AT1.Id` $\neq$ `AT2.Id`
MR_4	TC_5	`mitmproxy` is configured to redirecting the network traffic over an HTTP-CC	**REPLAY[**USER AUTHENTICATION (CORRECT UPC)**]**	The resulting GUI is not the one collected in the **App exploration** step, i.e., authentication must be refused
MR_5	TC_6	Same preconditions used in the **App exploration** step	**REPLAY[**USER AUTHENTICATION (INCORRECT UPC)**]** - **REPLAY[**ACCESS TO PRS**]**	The resulting GUIs are not the ones collected in the **App exploration** step, i.e., PR are not reachable through the graphical user interface.

Source: Adapted from Junior *et al.* (2023).

FUTC$_4$: implemented to perform MR_3, this test case exploits the same preconditions adopted in the **App exploration** step and replies to all events collected in it. Two ATs, AT1 and AT2, must be collected after USER SIGN OUT and the subsequent USER AUTHENTICATION steps, respectively. As in the previous test case, the two ATs are expected to be different for the assessment of the absence of vulnerability.

FUTC$_5$: introduced for executing MR_4, it behaves similarly to **FUTC$_1$** and **FUTC$_2$**, differing only from the point of view of preconditions, since the network traffic is redirected over an HTTP-CC.

FUTC$_6$: proposed for applying MR_5, it first replays USER AUTHENTICATION with incorrect credentials. ACCESS TO PR step is then executed for checking whether access to PRs has been denied, as expected.

3. ***Follow-up* test cases execution**: the *follow-up* test cases produced in the previous step are

automatically executed on AndDev.

5.2.1 Metamorphic Vulnerability Testing Environment

This section details the implementation of *Metamorphic Vulnerability Testing Environment*, designed for supporting the execution of GUI-Based MT technique.

5.2.1.1 Architectural overview

Figure 14 shows an architectural diagram of the testing environment regarding its basic components. One of the components is AndDev, which can be either *Real*, or a *Virtual Device*, and where AUT is preliminary installed and the preconditions are set.

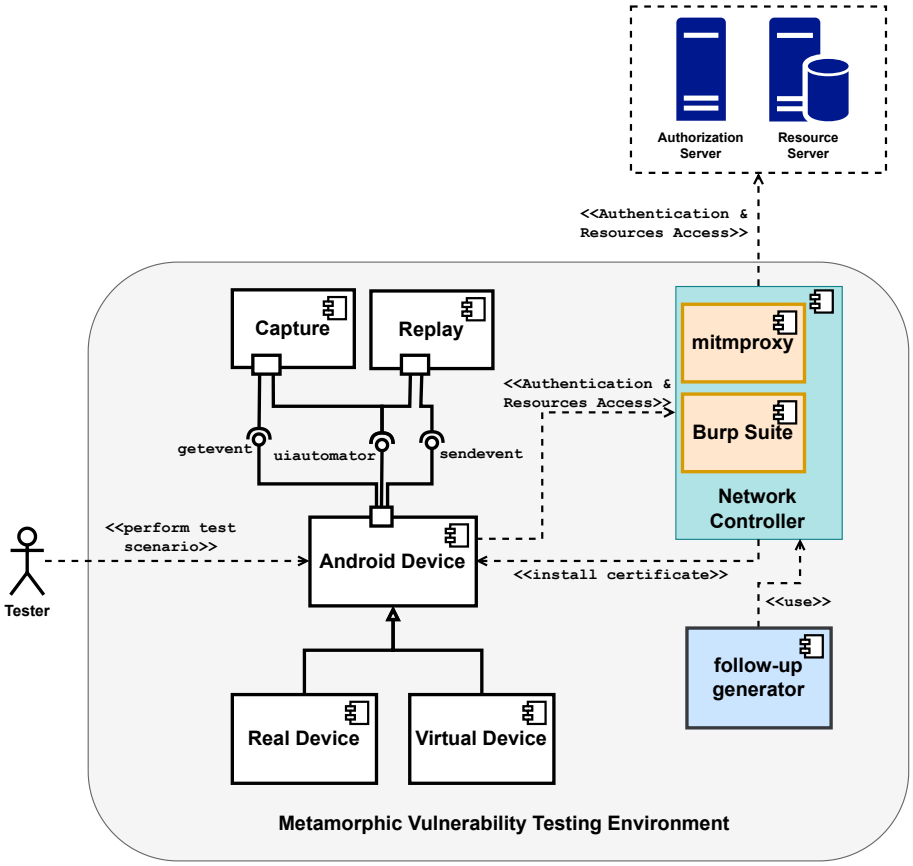

Figure 14 – Metamorphic Vulnerability Testing Environment architecture.

Source: Adapted from Junior *et al.* (2023).

The tester manually interacts with GUI's device to execute the test scenario shown in Figure 13 and, concomitantly, the *Capture* component captures all actions executed by the tester. More precisely, it exploits the combination of `getevent` (ANDROID, 2023a) and the `uiautomator` (ANDROID, 2023b) tools provided by Android. `getevent` captures the user events triggered by the tester and saves, at kernel level, those performed during the AUT exploration., while `uiautomator` is exploited to obtain a dump XML file describing the composing

widgets of GUI. `Getevent` provides a live dump stream of kernel-level input events, which contains information on the input events, such as timestamps, device names, event types, and screen coordinates.

The Capture component produces a single source GUI-based test case implemented in JUnit (GAMMA; BECK, 2017). The *follow-up generator* generates, from the aforementioned test case, a `JUnit` test-suite comprised of six *follow-up* test cases implementing the testing logic previously described. More precisely, the component performs the following changes to the source GUI-based test case:

1. modification to the initial preconditions by removing all certificates and installing only NTSSLDC on the device;

2. modification to the initial preconditions by removing all certificates and installing only SSDC on the device;

3. translation of the sequences of user events captured during the App Exploration step towards a format that can be executed on the device through sendevent, whose command does not enable setting the timing between consecutive sent events. Therefore, an ad-hoc solution was developed for replaying the kernel-level events with proper timing. The solution aims to avoid a too quick replay of high-level events and their faithful replay (AMALFITANO *et al.*, 2019);

4. addition of user interface oracles to the test cases by translating the descriptions of the user interfaces during the app exploration into assertions that can be automatically evaluated;

5. insertion of time delays necessary for *i)* the execution of FUTC3 and FUTC4 and *ii)* the guarantee of a correct replay of user events; and

6. addition of specific commands for collecting Access Tokens' descriptions.

The Android Device also performs follow-up test cases using the *Replay* component and exploiting `sendevent` for executing the events on the GUI of AUT. Replay also employs `uiautomator` to obtain the description of the current GUI, in XML format, and evaluate the oracles to check whether the GUI encountered is the expected one. Figure 15 shows an example for the booking app, with the snapshot of GUI indicating the user authentication has been refused (left image). This is an example of expected GUI for MR_1 and MR_4, in which the obtained GUI is expected to be different from the one obtained during the app exploration, when the user successfully authenticates (rightmost image). The middle of the figure shows an excerpt of the XML description related to GUI on the left and that also highlights the portion of code rendering the widgets of the user interface communicating "Something went wrong" during user authentication. If the application is vulnerable, according to MR1 and MR4, the user successfully

authenticates and GUI is equal to the one obtained during the app exploration. Since the two GUIs would have the same XML description, the oracle would be able to identify the weakness.

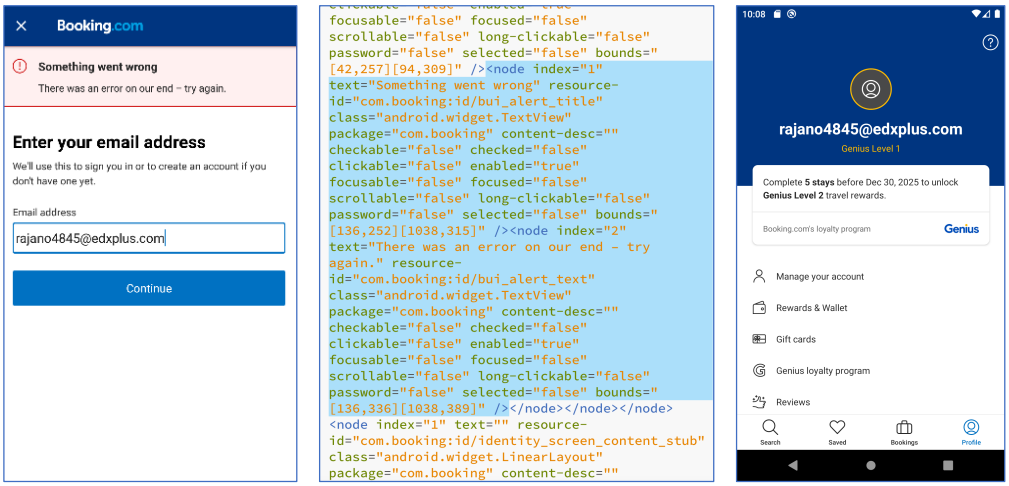

Figure 15 – Snapshot of the expected GUI (left), excerpt of the XML description of the expected GUI (middle), and snapshot of a GUI different from the expected one (right).

The combination of `getevent`, `sendevent`, and `uiatomator` tools was used by the authors for implementing a Capture and Replay testing approach for Android apps (AMALFITANO *et al.*, 2019). The ability to capture and execute events at the kernel level enables the adoption of the proposed metamorphic technique even for Android applications for which the source code is not available.

The *Network Controller* component has been introduced towards a full control of the network used by AndDev for communicating with both authentication and resource servers of AUT. It exploits the features provided by two free tools, namely, *Burp Suite* (PORTSWIGGER, 2016) and *mitmproxy* (CORTESI *et al.*, 2010), and provides the following features:

- *generation of digital certificates:* it generates and installs either NTSSLDC, or TSSLDC on the device;
- *redirection of the network traffic over a controlled network:* it acts as a proxy between AUT and both Auth-Server and Res-Server and enables selecting the connection channel for the data exchange, i.e., HTTP-CC rather than HTTPS-CC and vice-versa;
- *network traffic monitoring:* it monitors and analyzes the network traffic; more precisely, it intercepts and analyzes AT.

5.3 Metamorphic Relationships for Detecting Authentication Vulnerabilities

This section outlines the process that defined the MRs for detecting vulnerabilities associated with username and password authentication methods. It relied on the execution of two

steps, namely, *Vulnerabilities selection and weaknesses description* and *MRs definition* described in what follows.

5.3.1 Vulnerabilities selection and weaknesses description

This step focused on two categories of real-world vulnerabilities reported in OWASP, namely, *M3: Insecure Communication (OWASP, 2016b)* and *M4: Insecure Authentication (OWASP, 2016c)*. Since OWASP also traces each vulnerability to a list of weaknesses described in CWE, the weaknesses associated with the two vulnerability categories considered were manually analyzed. Among them, those that may be introduced if the guidelines defined by OAuth 2.0 (OAUTH, 2012) and reported in Section 3.3 for the implementation of a secure *Abstract Protocol Flow* are not properly followed by the developers were chosen.

Table 6 summarizes the weaknesses selected. CWE ID, CWE Name, a textual description, and the OWASP vulnerability category are listed for each weakness.

Table 6 – Weaknesses related to vulnerabilities due to improper design and implementation of username and password authentication methods.

CWE ID	CWE Name	Description	OWASP Category
CWE-295 (CWE, 2023c)	Improper Certificate Validation	A mobile app is potentially vulnerable if the Authentication Server accepts the user authentication request even if communication with the client is not established over an SSL/TSL secure channel.	M3: Insecure Communication (OWASP, 2016b)
CWE-613 (CWE, 2023f)	Insufficient Session Expiration	A mobile app is potentially vulnerable if the AT is not refreshed or destroyed after a long user´s inactivity time.	M3: Insecure Communication (OWASP, 2016b)
CWE-384 (CWE, 2023e)	Session Fixation	A mobile app is potentially vulnerable if the AT is not destroyed after the user has signed out.	M3: Insecure Communication (OWASP, 2016b)
CWE-311 (CWE, 2023d)	Missing Encryption of Sensitive Data	A mobile app is potentially vulnerable if the Authentication Server accepts the user authentication request even if communication with the client is not established through an HTTPS-CC protocol.	M3: Insecure Communication (OWASP, 2016b)
CWE-288 (CWE, 2023b)	Authentication Bypass Using an Alternate Path or Channel	A mobile app is potentially vulnerable if it does not limit access to functionalities and screens, collecting sensible data for only authenticated and authorized users.	M4: Insecure Authentication (OWASP, 2016c)

Source: Adapted from Junior *et al.* (2023).

5.3.2 Definition of Metamorphic Relationships

One or more MRs were manually defined for each weakness shown in Table 6. They were specified according to the template proposed by Segura *et al.* (2017) and described in the following subsections. The template is based on the structure of MRs from the literature and inspired by related and widely adopted templates in various fields of software engineering. It is intentionally simple and flexible to fostering adoption by the MT community and specifies the data to be included in the description of an MR using a natural language, a formal language, or a combination of both. The template is a combination of placeholders and linguistic formulas used to describe an MR. Table 7 shows the ID (MR ID) and the formalization (MR Formalization) of the MR defined for each weakness reported in Table 6.

5.3.2.1 Improper Certificate Validation

MR_1, listed in Table 7, was defined for CWE-295 *Improper Certificate Validation* weakness. More precisely, $Auth(AndDev, UPC, AUT, certificate)$ is the authentication function triggered when RO authenticates through the SIS provided by the AUT installed on an AndDev equipped with a digital certificate. The function returns the expected screen (GUI) after the correct authentication of RO. MR checks whether AR has been refused by the Authentication Server when AndDev exploits either a No Trusted SSL/TLS Digital Certificate (NTSSLDC), or a Self Signed Digital Certificate (SSDC) for encrypting communication between client and server. Indeed, *Auth* function is expected to return a different GUI different from the one obtained from a successful authentication in the two scenarios

5.3.2.2 Insufficient Session Expiration

MR_2, listed in Table 7, was defined for CWE-613 *Insuflcient Session Expiration* weakness. Such a Metamorphic Relationship relies on the AT property that must be refreshed by Auth-Server after a long time, e.g., one hour. More specifically, the property imposes the identifier of the AT evaluated at time t1 must be different from the one collected at time $t1 + \Delta$ with $\Delta \geq 60$ minutes.

5.3.2.3 Session Fixation

MR_3, related to *Session Fixation* weakness, is reported in Table 7. It exploits the property that a new AT must be generated by the Authorization Server when the user signs out from AUT and then signs in again through Sign-In Screen. It also requires the identifier of the AT collected before the user signs out be different than the one evaluated after the subsequent sign-in of the user.

Table 7 – Definition of the six MRs introduced for each weakness listed in Table 6.

CWE ID	MR ID	MR Formalization
CWE-295 (CWE, 2023c)	MR_1	**IN THE DOMAIN OF *Android applications*** **THE FOLLOWING METAMORPHIC RELATION SHOULD HOLD** **MR_1** **IF** $S_{nv} = Auth(AndDev, UPC, AUT, TSSLDC)$ **AND** $S_v = Auth(AndDev, UPC, AUT, NTSSLDC)$ **OR** $S_v = Auth(AndDev, UPC, AUT, SSDC)$ **THEN** $S_{nv} \neq S_v$
CWE-613 (CWE, 2023f)	MR_2	**IN THE DOMAIN OF *Android Applications*** **ASSUMING THAT** the ***ro*** has authenticated at time t_0 **THE FOLLOWING METAMORPHIC RELATION SHOULD HOLD** **MR_2** **IF** the ***RO*** interacts with the AUT GUI up to time $t_1 > t_0$ **AND** the ***RO*** does not interact with the ***AUT*** GUI for a long time interval Δ after $t1$ (such as $\Delta > 60$ minutes). **THEN** $IdAT(t1 + \Delta) \neq IdAT(t1)$.
CWE-384 (CWE, 2023e)	MR_3	**IN THE DOMAIN OF *Android applications*** **ASSUMING THAT** the ***RO*** has authenticated at time t_0 **THE FOLLOWING METAMORPHIC RELATION SHOULD HOLD** **MR_3** **IF** the ***RP*** at time $t2 = t1 + \delta$ (with $\delta \approx 1$ *sec*) performs the sign-out and sign-in through the ***SIS*** of the ***AUT*** **THEN** $IdAT(t2) \neq IdAT(t1)$.
CWE-311 (CWE, 2023d)	MR_4	**IN THE DOMAIN OF *Android applications*** **THE FOLLOWING METAMORPHIC RELATION(S) SHOULD HOLD** **MR_4** **IF** $S_{nv} = Auth(AndDev, UPC, AUT, HTTPS)$ **AND** $S_v = Auth(AndDev, UPC, AUT, HTTP)$ **THEN** $S_{nv} \neq S_v$
CWE-288 (CWE, 2023b)	MR_5	**IN THE DOMAIN OF *Android applications*** **ASSUMING THAT** S1, ..., Sn is the set of *AUT* screens for accessing to the ***PR*** **THE FOLLOWING METAMORPHIC RELATION SHOULD HOLD** **MR_5** **IF** the ***RO*** is not authenticated, **THEN** a not authenticated user can not access to the ***PR*** though S1, ..., Sn.

Source: Adapted from Junior *et al.* (2023).

5.3.2.4 Missing Encryption of Sensitive Data

Regarding CWE-311 *Missing Encryption of Sensitive Data* weakness, MR_4, shown in Table 7, was defined. The property behind it requires Auth-Server refuse AR if AndDev does not exploit an HTTPS-CC for transferring UPC from AUT to Auth-Server. Similarly to MR1, $Auth(AndDev, UPC, AUT, channel)$ function was defined returning the expected screen (GUI) after the authentication of the RO through the Sign-In Screen of AUT when a *channel* is used to encrypt the UPC.

5.3.2.5 Authentication Bypass

Regarding CWE-288 *Authentication Bypass* weakness, MR_5, listed in Table 7, has been proposed. The property on which MR was built requires AUT must not permit unauthenticated users to reach screens showing PR or providing features for modifying them.

5.4 Experimental Evaluation

This section describes the experiment conducted towards (1) understanding whether the proposed MT technique can reveal vulnerabilities in real Android mobile apps, (2) characterizing the diffusion of such vulnerabilities, and (3) understanding whether Android applications with higher user perceived quality are less vulnerable.

According to the GQM approach (CRUZES *et al.*, 2007), the following 3 research questions and their rationale were defined towards reaching those goals:

RQ_1 Is the testing technique able to detect vulnerabilities related to the username and password authentication methods in real Android apps?
Rationale: this RQ aims to evaluate whether the proposed MT technique is suitable for detecting new vulnerabilities known, a priori, by the developers, in real mobile apps available in the official Google market store.

RQ_2 How do different MRs compare regarding their ability to detect vulnerabilities?
Rationale: supposing the answer is yes, the question has two fold aims. On the one hand, it aims to understand the ability of each MR to detect real vulnerabilities. On the other hand, it might provide an overview of the most common user authentication vulnerabilities.

RQ_3 How do vulnerabilities in Android apps vary with the user-perceived quality of the apps?
Rationale: This question aims to understand whether, as expected, Android applications with better user-perceived quality have fewer vulnerabilities.

The metrics reported in Table 8 were adopted so that the proposed questions could be answered.

Table 8 – Weaknesses related to vulnerabilities due to improper design and implementation of username and password authentication methods.

Metric ID	Description
Metric1	Number of detected vulnerabilities per app; the metric is used to answer RQ_1 and RQ_3.
Metric2	Number of detected vulnerabilities by each MR per app; the metric is used to answer RQ_2.
Metric3	Number of user downloads per app; the metric is used in combination with Metric4 to evaluate the user perceived quality for answering RQ_3.
Metric4	Number of stars rated by users per app; the metric is used in combination with Metric3 to evaluate the user perceived quality for answering RQ_3.

Source: Adapted from Junior *et al.* (2023).

5.4.1 Object selection

The inclusion (IC) and exclusion (EC) criteria summarized in what follows were defined for the construction of the sample of objects of the experiment, i.e., AUT, from the most downloaded real Android apps exploiting user and password authentication methods. Open-source applications downloaded from F-Droid were not used, since the interest was in understanding errors in the code that caused vulnerabilities. The sample can be considered significant, since the approach was experimented with the most downloaded real apps.

IC_1 AUT can be freely downloaded from the official Brazilian or Italian Google Play Stores.

IC_2 AUT belongs to one of the following three apps categories: *Food & Drink*, *Shopping*, and *Travel & Local*. The categories were chosen because their apps usually adopt a user and password authentication method for allowing access to the user's PR.

IC_3 AUT is one of the top 100 downloaded apps in its category, which enabled the construction of a sample with the most popular Android apps.

EC_1 AUT fails during either registration, or sign-in process, i.e., authentication could not be completed due to a failure in the selected Android app.

EC_2 AUT does not adopt a user and password authentication method.

EC_3 AUT digital certificate could not be changed and, therefore, MR_1 was applied.

EC_4 AT could not be identified and analyzed after authentication; therefore, MR_2 and MR_3 were applied.

EC_5 The network traffic could not be redirected over an HTTP-CC, as required for the evaluation of MR_4.

The manual application of IC and EC returned 163 apps, of which 57 belong to *Food & Drink*, 65 are in the *Shopping* category, and 41 are *Travel & Local* apps. Figures 16 and 17 graphically show the characteristics of the sample in two box plots that display the distributions of the rated stars (Figure 16) and the number of downloads (Figure 17), respectively. According to the graphs, more than half of the sample's apps were rated with more than 4.3 stars (users can rate the quality of an app from zero to five stars) and have more than 1.000.000 of downloads. Three levels of user perceived quality, namely, *High*, *Medium*, and *Low* were defined for an Android app. An application has a *High user perceived quality* if its rating is greater than or equal to 4.3 stars and its number of downloads is greater than or equal to 1.000.000. Conversely, an Android app is classified as *Low user perceived quality* if its rating is lower than 4.3 stars and it has been downloaded less than 1.000.000 times. In the other cases, an app is clustered as *Medium user perceived quality*. According to this definition, 46 apps were classified as *Low*, 46 were categorized as *Medium*, and 71 were classified as *High* perceived quality.

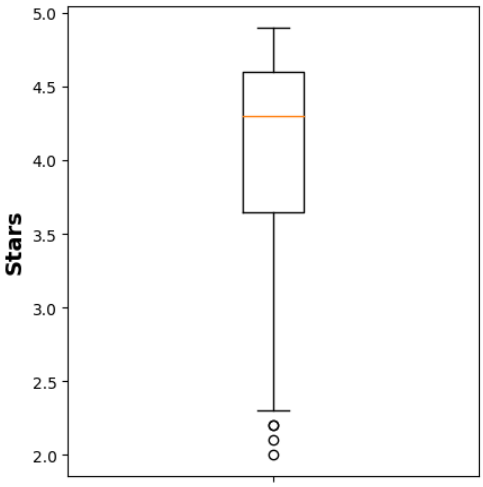

Figure 16 – Distribution of rated stars for the apps of the sample.

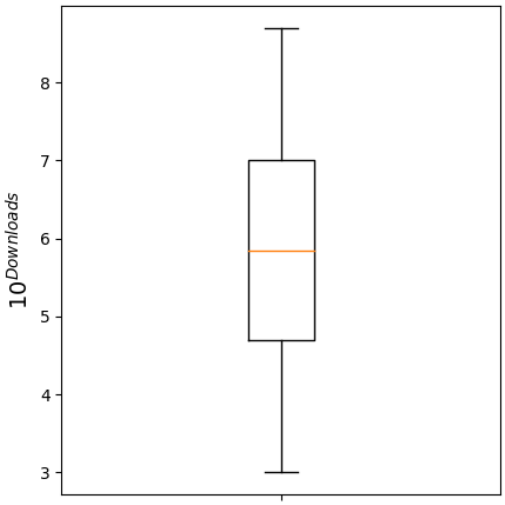

Figure 17 – Distribution of downloads of the apps of the sample.

Source: Adapted from Junior *et al.* (2022).

5.4.2 Experimental procedure

The experimental procedure followed the steps executed for each app of the sample described below.

- *Registration of new user:* a new user account is generated for testing the AUT.
- *Manual AAUT exploration:* AUT was manually explored following the scenario reported in Figure 13 for producing a ***Source* GUI-based test case**. The step was performed on an Android Virtual Device (AVD), equipped with Android Q (vers. 10), which was one of the newest and most diffused versions of Android at the time of the experiment.

- *Follow-up test cases generation and execution:* the ***Source*** **GUI-based test case** was automatically translated in a test suite of ***Follow-up*** **executable test cases** launched on the same *AVD* used in the previous step for avoiding failure due to different characteristics of the user interface. This choice did not influence the results, since the vulnerabilities considered do not rely on the hardware characteristics of the mobile device, and, in particular, of the device screen characteristics. Details of the found vulnerabilities were stored in the *Test execution report*.

- *Vulnerabilities validation:* the vulnerabilities detected were manually validated by experts in security. Additionally, 50 companies were randomly selected owning to a vulnerable application. 13 of them were discarded, since they either provided no contact information, or provided invalid data (e.g., email, online form, Twitter account). The remaining 37 companies were contacted and 9 responded to the message confirming the reported vulnerabilities. Three of them asked our group for a consult for more information on the vulnerabilities reported and the way to fix them. Although most of the 37 companies did not reply to our message, 26 updated their app and fixed the vulnerabilities reported. Towards mitigating additional threats to validity, the same tests were applied on previous Android versions like Android 6.0 (Marshmallow), Android 7.0 (Nougat), which were among the most used in Brazil in 2020, and Android 9.0 (Pie). All vulnerabilities found in Android 10 were also detected in those versions. Since this is the first study for the detection of authentication vulnerabilities in Android mobile apps, there is no baseline for comparisons of the effectiveness of the approach. However, evidence on its ability to detect vulnerabilities not known by developers is a valid demonstration of its effectiveness.

5.4.3 Experimental results and answers to RQs

Table 9 shows the number of vulnerabilities detected and validated by each MR for each app category and Figure 18 displays a clustering of our app sample in six groups based on the app vulnerability and the three levels of user-perceived quality. Each cluster indicates number of apps contained and corresponding percentage. An application is clustered as vulnerable if it exposes at least one vulnerability. The bubble charts in Figures 19 and 20 show two detailed views of the clusters in Figure 18. More precisely, each bubble in Figure 19 reports the number of applications that have a given number of rated stars (x- axis) for which the number of vulnerabilities on the y-axis was found; analogously, Figure 20 shows the distribution of the applications on the basis of their downloads (x-axis) and number of vulnerabilities (y-axis). Towards mitigating security risks to the implicated applications, the comprehensive roster of vulnerable apps has been omitted; nevertheless, the full list is available upon request, securely stored in a private folder[3].

[3]<https://docs.google.com/spreadsheets/d/1ITgACBUII1HX1vapGj32TJksjNr2iX1xYyNiA9z1Ulw/edit?usp=sharing>

Table 9 – Number of vulnerabilities detected by each MR for each app category.

Apps Category	MR_1	MR_2	MR_3	MR_4	MR_5	Total
Food & Drink	8	42	9	8	1	68
Shopping	4	43	13	4	0	64
Travel & Local	3	19	3	2	0	27
Total	15	104	25	14	1	159

Source: Adapted from Junior *et al.* (2023).

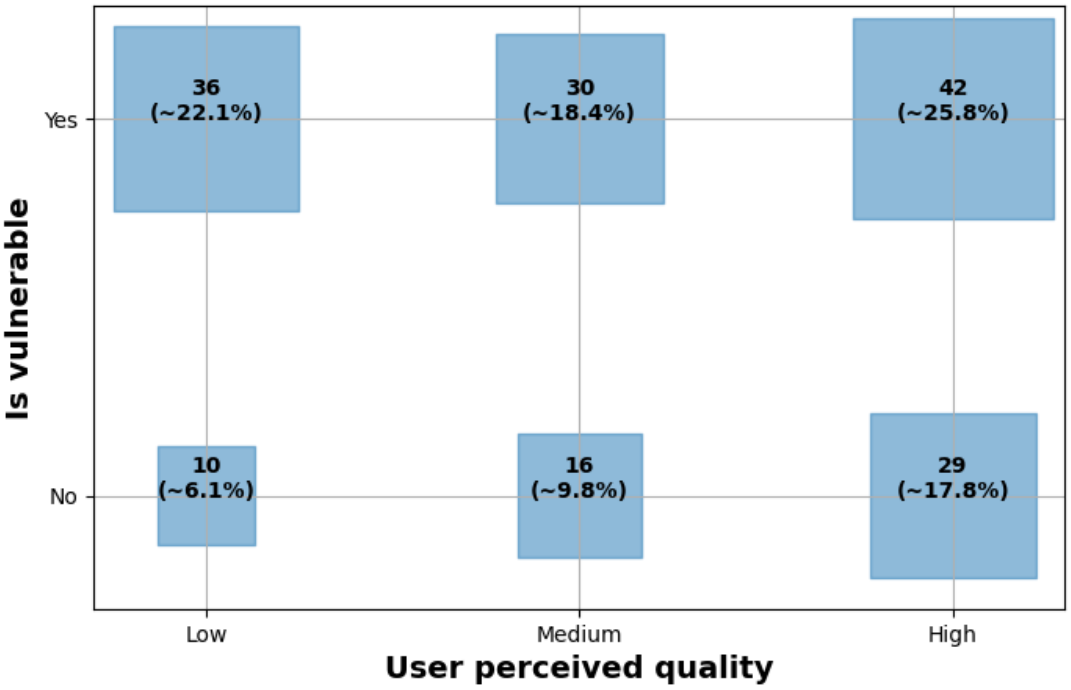

Figure 18 – Clustering of the sample by vulnerability and user perceived quality of AUT.

Source: Adapted from Junior *et al.* (2023).

5.4.3.1 Answer to RQ_1

According to Table 9, the testing approach was able to detect 159 vulnerabilities in the apps and at least one was related to each MR. Figure 18 shows 66.3% (108/163) of apps exposed at least one vulnerability and Figure 20 displays 71 out of the 108 apps exposed one vulnerability, 31 presented two, 4 revealed 4, and the last 2 apps exposed 4 and 5 vulnerabilities, respectively. Therefore, the following answer can be derived for RQ_1:

> RQ_1 *Answer*: the proposed metamorphic-based testing technique is able to detect previously unknown vulnerabilities related to username and password authentication methods in real Android apps.

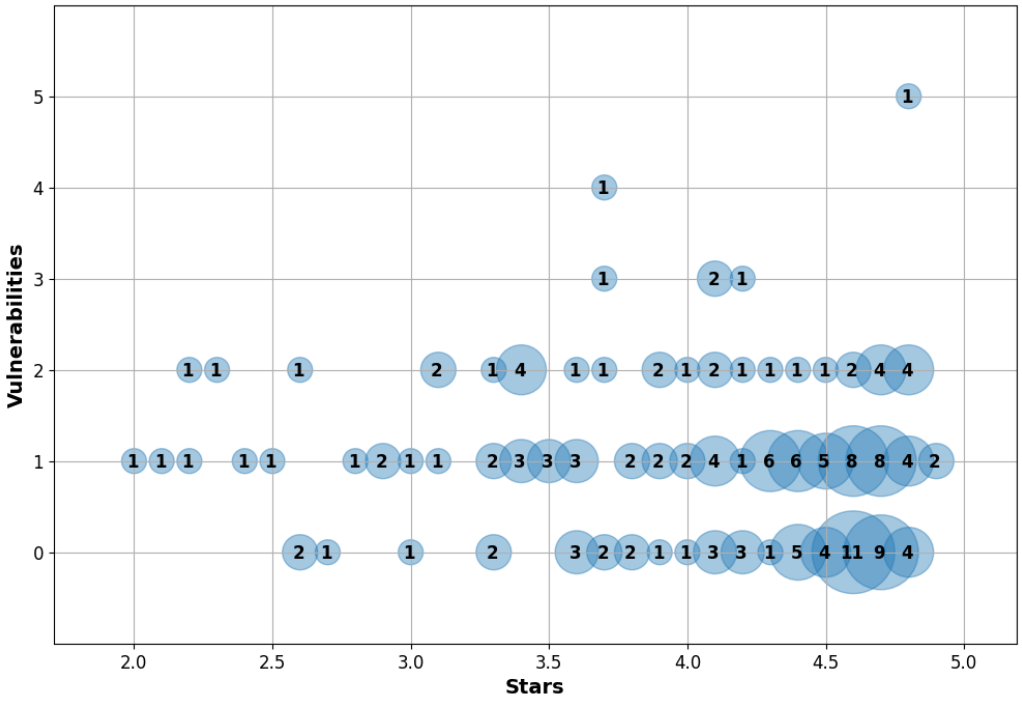

Figure 19 – Distribution of applications per vulnerabilities and rated stars.

Source: Adapted from Junior *et al.* (2023).

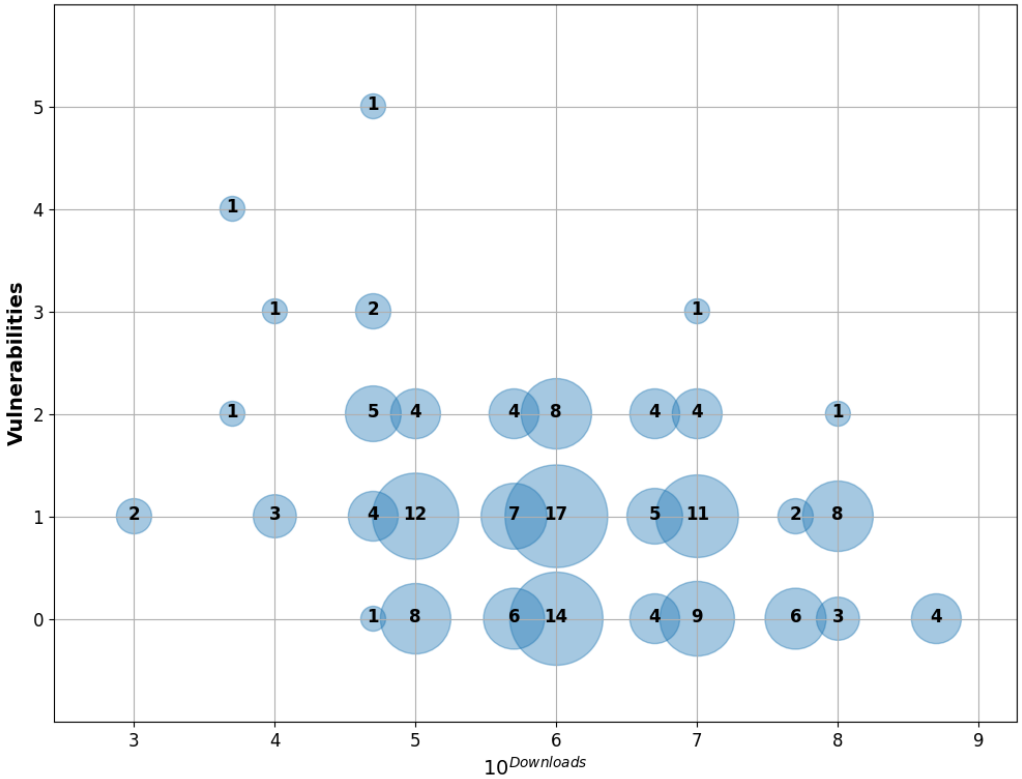

Figure 20 – Distribution of vulnerabilities per user perceived quality characteristics.

Source: Adapted from Junior *et al.* (2023).

5.4.3.2 Answer to RQ_2

According to Table 9, most of vulnerabilities were detected by MR_2 $(104/159)$, followed by MR_3 $(25/159)$, indicating the weaknesses related to an improper handling of AT are the most widespread among those considered in this study. Fourteen vulnerabilities were detected by MR_3, showing some apps still share sensible data over an insecure HTTP channel. MR_1 detected $15/159$ vulnerabilities regarding weaknesses on an incorrect use of digital certificates. Indeed, the results showed apps in the sample enable authentication and access to protected data also by devices with no trusted certificates. One vulnerability alone being found by MR_5 was already very dangerous, since one application enabled unauthenticated users to read private chats. Therefore, the answer to RQ_2 is:

> RQ_2 *Answer*: some MRs are able to find more vulnerabilities than others, which suggests some security weaknesses are much more widespread in mobile apps than others.

5.4.3.3 Answer to RQ_3

Figures 18, 19, and 20 provide, respectively, graphical representations of possible correlations between the vulnerabilities of the apps in the sample against user-perceived quality, rated stars, and number of downloads. Since the plots show no evidence of correlations, a statistical analysis was conducted for confirming that observation. Three correlation parameters, namely, Pearson, Spearman, and Kendall coefficients, were evaluated between different couples of variables towards the detection of presence or absence of correlations. Table 10 shows the correlation coefficients were evaluated between number of downloads, rated stars, and user perceived quality w.r.t. number of vulnerabilities and exposure, i.e., if the application is vulnerable or not. All coefficients are very low, thus evidencing no correlations among the variables. Such observations led to the following answer to RQ_3:

Table 10 – Correlation coefficients

	Pearson Coefficient		
	Downloads	**Stars**	**User perceived quality**
Vulnerabilities	-0.201	-0.113	-0.247
Is vulnerable	-0.225	-0.151	-0.183
	Spearman Coefficient		
	Downloads	**Stars**	**User perceived quality**
Vulnerabilities	-0.27	-0.13	-0.243
Is vulnerable	-0.225	-0.13	-0.193
	Kendall Coefficient		
	Downloads	**Stars**	**User perceived quality**
Vulnerabilities	-0.221	-0.107	0.215
Is vulnerable	-0.191	-0.114	-0.183

Source: Adapted from Junior *et al.* (2023).

RQ_3 Answer: applications with higher perceived quality, more downloads, or better ratings are not expected to show fewer vulnerabilities or no vulnerability.

5.5 Threats to Validity

The technique was evaluated with more than 150 objects; however, some threats to validity must be highlighted. This subsection discusses those to external, internal, conclusion, and construct validity.

5.5.1 External validity

A threat that may affect the external validity of this study is the way the objects of the experiment were sampled. Applications were downloaded only from the Brazilian and Italian official markets of Google Play and belong to only three categories. In an attempt to mitigate the threat, a sample of more than 150 apps diversified from the point of view of category and user-perceived quality was downloaded. However, the result may not be valid for other types of applications. The experiment aimed to show some vulnerability might be exposed with the use of our approach and not that it will always occur. Towards further mitigating it, the experimentation might be improved by extending the sample of tested apps and considering applications downloaded by other international markets and belonging to additional categories.

Another threat that may influence the generality of the results is, at first, a single version of Android was used in the experiment. Android 10.0, which is the newest and most worldwide diffused version, was exploited. Towards mitigating this threat, the same tests were applied on the previous version of Android as an additional confirmation of the results. The same testing technique can be used on different devices equipped with a different version of Android for further mitigating the threat.

Another threat considered may have stemmed from the selection of free applications rather than open-source or paid ones. Since the chosen applications are representative of real-world apps due to their millions of users, the threat can be further mitigated by the inclusion of open-source and paid applications in the experiment.

5.5.2 Internal validity

The technique can find vulnerabilities where the authentication method is self-implemented by the application, i.e., does not consider other authentication methods like *i)* those provided by third-party services such as Google or Facebook, *ii)* the so-called self pinned certificate, according to which the app is distributed along with its trusted digital certificate, and *iii)* two or three-step authentication methods. As a consequence, other types of vulnerabilities, apart from the ones found in this study, may be present in the analyzed sample of apps. Towards mitigating this threat, a modification in the testing technique would be required by 1) introducing further

MRs that take into account those authentication methods and 2) extending the Metamorphic Vulnerability Testing Environment for enabling their execution.

5.5.3 Conclusion validity

A threat that may affect the conclusion validity of this study is related to metric introduced for the evaluation of the user-perceived quality. It is based on the assumption such quality increases with the number of downloads and rated stars. Although meaningful due to its objective and measurable value of quality, the metric does not explicitly consider users' opinions about the app quality. The threat can be mitigated by introducing real subjects in the experiment, i.e., by involving real users to evaluate, or confirm, the user-perceived quality of the sampled apps.

5.5.4 Construct validity

The quality metrics of apps, such as the number of downloads and star ratings, have a very limited relationship with authentication and an even smaller one with authentication security. As a consequence, as shown in Table 10, no statistical significance was identified in the relationship between these app quality metrics. To mitigate this threat, various statistical tests were conducted to explore potential correlations from different angles. This comprehensive approach aimed to ensure that any subtle relationships between the quality metrics and authentication security were not overlooked.

5.6 Final Remarks

This chapter has presented a metamorphic-based testing technique for the identification of vulnerabilities in Android applications arising from flawed implementations of username and password authentication methods, particularly those that diverge from the OAuth 2.0 guidelines (OAUTH, 2012). Drawing inspiration from a prior study that introduced MT for uncovering vulnerabilities in web systems (MAI *et al.*, 2019), a tool-supported testing technique has been introduced. It focused on uncovering five of the most prevalent and emblematic real-world vulnerabilities associated with username and password authentication methods, as per the OWASP's classification.

A set of six MRs was devised and a dedicated Metamorphic Vulnerability Testing Environment was developed for facilitating their evaluation. Subsequently, a comprehensive experiment was conducted involving 163 real-world Android apps with a widespread usage in both Brazilian and Italian markets. The results were compelling, revealing 159 hitherto unknown vulnerabilities that had eluded the attention of developers. Surprisingly, 108 of the tested apps exhibited at least one vulnerability, underscoring the effectiveness of the metamorphic-based vulnerability testing technique.

The experiment yielded noteworthy insights. While it reaffirmed the expected correlation between higher user-perceived quality and lower vulnerability, it also presented an unexpected finding: even applications perceived as high-quality by users were not immune to vulnerabilities. This finding underscores the continuing need for robust security testing in the mobile app landscape.

CHAPTER

6

CONCLUSIONS

Mobile apps have been widely adopted in recent years, thus requiring new approaches in software engineering to ensure their quality. Common issues such as excessive power consumption, unexpected user interface behavior, and susceptibility to attacks during insecure server connections are intricately tied to NFRs and foundational aspects of mobile apps (ALOTAIBI; CLAUSE; HALFOND, 2020; YU *et al.*, 2021). In response, numerous studies have aimed at characterizing testing techniques specifically tailored for mobile apps.

MT, initially proposed as a testing technique to address the oracle problem, has been extensively employed for detecting flaws related to NFRs. Following the guidelines presented by Segura *et al.* (2018), which introduced a set of principles for applying MT to uncover faults related to NFRs (e.g., performance issues), subsequent studies have leveraged MT for detecting NFR-related flaws (AZIMIAN *et al.*, 2019; JOHNSTON *et al.*, 2019; MAI *et al.*, 2019; AYERDI *et al.*, 2022; RAHMAN; IZURIETA, 2023; CORRADINI; PASQUA; CECCATO, 2023). However, none of those studies have specifically offered a security testing technique that uses MT concepts to identify vulnerabilities in mobile apps.

Motivated by this gap, this PhD research involved two primary steps. Initially, comprehensive discussions ensued various aspects related to dynamic testing of NFRs in mobile apps, providing an overview of existing test techniques towards meeting different NFRs, as established by the quality standard from ISO (2011). The test techniques identified were characterized, focusing on collaboration between academia and industry, testing approaches and strategies, predominant mobile app types and platforms addressed, and tool support. Additionally, a broad discussion was held within the testing community on the main trends and research opportunities in this domain. The results were published in CSUR (JUNIOR *et al.*, 2022).

Inspired by a previous study that applied MT to discover vulnerabilities in web applications (MAI *et al.*, 2019; CHALESHTARI *et al.*, 2023b), a tool-supported testing technique was developed specifically for finding five of the most diffused and representative real world

vulnerabilities related to username and password authentication methods, according to OWASP. More precisely, MRs were designed and a Metamorphic Vulnerability Testing Environment was implemented for checking them. The technique was applied in an experiment for testing 163 real-world Android applications massively used from both Brazilian and Italian markets. 159 vulnerabilities unknown to developers were detected and 108 of the AUTs showed at least one vulnerability, demonstrating the effectiveness of the technique. Additionally, the correlation analysis provided evidence that, contrarily to expectations, it cannot be asserted applications with higher perceived quality, or more downloads, or even better rated have fewer vulnerabilities or are free from being vulnerable. A paper reporting such findings has been submitted to JSS (JUNIOR *et al.*, 2023).

6.1 Revisiting the Thesis Contribution

The general contribution of this PhD research is the definition of a security testing technique based on MT concepts for the detection of vulnerabilities in username and password authentication methods in Android applications. The specific contributions are summarized in what follows.

- **Dynamic testing techniques of NFRs in mobile applications:** an SMS conducted investigated dynamic testing techniques for NFRs for detecting faults in mobile applications. The present study followed the guidelines proposed by Petersen, Vakkalanka and Kuzniarz (2015). The SMS results provide an insightful overview of such techniques according to the quality standards set established by ISO (2011). Their characterization included discussions on several key aspects, namely, (1) collaboration between academia and industry, (2) testing approaches and strategies, (3) types of mobile applications and platforms most commonly addressed, and (4) tool support. Furthermore, a comprehensive discussion was held within the testing community on the primary trends and research opportunities in the domain. All artifacts generated during the SMS are publicly available and intended to serve as a valuable resource for the testing community, assisting researchers in exploring opportunities, identifying gaps, and envisioning future research directions (JUNIOR *et al.*, 2022). The SMS findings are provided in Section 4;
- **Characterization of representative real-world vulnerabilities in Android mobile applications:** towards the detection of genuine and illustrative vulnerabilities in Android mobile applications, an exhaustive manual analysis of vulnerabilities listed in three categories of OWASP (MUELLER; SCHLEIER; WILLEMSEN, 2019), including the *Top Ten 2016 Insecure Communication*, Insecure Communication (M3), Insecure Authentication (M4), and Insecure Authorization (M6) (OWASP, 2023b) was conducted. The five vulnerabilities identified can potentially appear if the guidelines outlined in Section 3.3 are not followed by developers. Weaknesses extensively documented in the community-driven CWE list

(CWE, 2023a) and associated with those five vulnerabilities were also deeply examined. A focus was placed on vulnerabilities related to an improper and insecure implementation of the Abstract Protocol Flow defined by OAuth 2.0. Their detailed descriptions are provided in Section 5.3;

- **Tool-supported MT-based vulnerability testing technique for Android mobile applications:** inspired by a previous study that introduced MT to find vulnerabilities in web applications (MAI *et al.*, 2019; CHALESHTARI *et al.*, 2023b), a tool-supported testing technique was formulated towards identifying five of the most prevalent and representative real-world vulnerabilities associated with username and password authentication methods, accoring to OWASP guidelines (OWASP, 2023b). Five MRs were designed and a dedicated Metamorphic Vulnerability Testing Environment was implemented for their evaluation. Section 5.2 provides a comprehensive exploration of the technique;
- **A wide experimentation on the most used Android mobile applications freely available on Brazilian and Italian Google Play:** a comprehensive experiment involving 163 popular Android applications of which 112 were sourced from Brazilian Google Play[1] and the remaining 51 were obtained from Italian Google Play[2] was conducted. The applications spanned three categories, namely, Food & Drink (57 out of 163), Shopping (65 out of 163), and Travel & Local (41 out of 163). According to the findings, 66.3% (108 out of 163) were affected by at least one vulnerability. Interestingly, the experiment revealed applications perceived as higher-quality ones by users tend to be less vulnerable and those with a high perceived quality are not immune to vulnerabilities. For a more in-depth exploration of the experiment and its results, please refer to Section 5.4;
- **Dissemination of scientific results related to the research field of this PhD research:**
  - Submitted Article: JUNIOR, M. C., AMALFITANO, D., VISONE, B., FASOLINO, A. R., DELAMARO, M.. A Metamorphic Testing Technique for Detecting Authentication Vulnerabilities in Android Mobile Apps. **Submitted to the Journal of Systems & Software (JSS)**, 2023. p. 1-28;
  - Published Article: JUNIOR, M. C., AMALFITANO, D., GARCES, L., FASOLINO, A. R., ANDRADE, S. A., DELAMARO, M.. Dynamic testing techniques of non-functional requirements in mobile apps: A systematic mapping study. **ACM Computing Surveys (CSUR)**, 54(10s), 2022. p. 1-38. Available: <https://doi.org/10.1145/3507903>;
  - Published Article: JUNIOR, M. C.. Automated verification of compliance of non-functional requirements on mobile applications through metamorphic testing. In: **Proceedings of the 13th International Conference on Software Testing, Validation and Verification (ICST) - Doctoral Symposium**. IEEE, 2020. p. 421-423. Available: <https://doi.org/10.1109/ICST46399.2020.00053>.
- **Dissemination of scientific results related to other research fields:**

- Published Article: ANDRADE, S. A., SANTOS, I., JUNIOR, C. B., JUNIOR, M. C., DE SOUZA, S. R., DELAMARO, M. E.. On applying metamorphic testing: an empirical study on academic search engines. **Proceedings of the 4th International Workshop on Metamorphic Testing (MET)**. IEEE, 2019. p. 9-16. Available: <https://doi.org/10.1109/MET.2019.00010>;
- Published Article: ANDRADE, S. A., SANTOS, I., JUNIOR, C. B., JUNIOR, M. C., MARCIEL, A. C., ABDALLA, G, DELAMARO, M. E.. Analyzing the effectiveness of One-Op Mutation against the minimal set of mutants. **Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing (SAST)**. ACM Press, 2019. p. 22-31. Available: <https://doi.org/10.1145/3356317.3356321>.

6.2 Limitations and Future Work

The limitations of this PhD research are related to primary aspects, namely, (*i*) targeted vulnerability class and (*ii*) number of elaborated MRs. Concerning the former, the study deliberately concentrated on vulnerabilities associated with username and password authentication methods in Android applications due to their classification as one of the most critical and prevalent issues in mobile applications since 2016, according to OWASP (2023b). Moreover, CWE (2023a) reported at least six types of vulnerabilities affecting username and password authentication methods, ranking among the Top 25 Most Dangerous Software Weaknesses in 2021[1]. Consequently, the selection of this vulnerability class was driven by its criticality, prevalence, and real-world manifestation in mobile applications.

Unlike studies such as those of Mai *et al.* (2019) and Chaleshtari *et al.* (2023b), which addressed a broad spectrum of vulnerabilities in Web systems, this PhD research refined a singular vulnerability class within Android applications, recognized as one of its primary limitations.

Concerning number of elaborated MRs, the research concentrated on a set of five primary vulnerabilities related to authentication and authorization methods in Android applications (see Table 6). Interestingly, the vulnerability associated with Certificate Improper was further detailed into two distinct vulnerabilities, namely, absence of a certificate and presence of a self-signed one, thus leading to five MRs addressed. However, five out of such MRs proposed in this PhD research were also considered by Mai *et al.* (2019) and Chaleshtari *et al.* (2023b), despite their focus on Web systems. In essence, the MRs proposed by them were adapted to the application domain of Android applications.

Whereas the set of elaborated MRs aligns with those of Mai *et al.* (2019) and Chaleshtari *et al.* (2023b), the strategy for their application in the respective domain differs, also being considered one of the main limitations of this research.

[1]Refer to the vulnerabilities ranked at (11, 14, 18, 20) in CWE (2021).

As part of future work, the applicability of the approach will be broadened through a diversification of the sample of objects (*i*) sourced from other official markets and (*ii*) spanning additional categories. The approach is expected to be adopted for the detection of other vulnerabilities in Android mobile apps. A more pragmatic future direction involves extending the implementation of MRs to encompassing other authentication methods, including (*i*) those provided by third-party services such as Google or Facebook, (*ii*) self-pinned certificate — i.e., a certificate distributed with the app, and (*iii*) two or three-step authentication methods. The integration of Machine Learning (ML) techniques can also be explored for enhancing both efficiency and accuracy of vulnerability detection and the development of automated tools for streamlining the testing process.

BIBLIOGRAPHY

ADAMS, K. **Non-functional Requirements in Systems Analysis and Design**. Cham: Springer, 2015. 264 p. ISBN 978-3-319-18343-5. Citation on page 35.

AFREEN, N.; KHATOON, A.; SADIQ, M. A taxonomy of software's non-functional requirements. In: **Proceedings of the 2th International conference on computer and communication technologies**. USA: Springer, 2016. v. 379, n. 1, p. 47–53. Citation on page 34.

AHMAD, A.; FENG, C.; LI, K.; ASIM, S. M.; SUN, T. Toward empirically investigating non-functional requirements of ios developers on stack overflow. **IEEE Access**, v. 7, n. 1, p. 61145–61169, 2019. Citations on pages 34, 37, and 51.

AL-AHMAD, A. S.; KAHTAN, H.; HUJAINAH, F.; JALAB, H. A. Systematic literature review on penetration testing for mobile cloud computing applications. **IEEE Access**, v. 7, n. 1, p. 173524–173540, 2019. Citations on pages 47 and 51.

AL-TEKREETI, M.; ABDRABOU, A.; NAIK, K. An end-user-centric test generation methodology for performance evaluation of mobile networked applications. **Software Testing, Verification and Reliability**, v. 29, n. 6-7, 2019. Citations on pages 43 and 51.

ALHANAHNAH, M.; YAN, Q.; BAGHERI, H.; ZHOU, H.; TSUTANO, Y.; SRISA-AN, W.; LUO, X. Dina: Detecting hidden android inter-app communication in dynamic loaded code. **IEEE Transactions on Information Forensics and Security**, IEEE, v. 15, n. 1, p. 2782–2797, 2020. Citation on page 58.

ALMEIDA, D. R.; MACHADO, P. D.; ANDRADE, W. L. Testing tools for android context-aware applications: a systematic mapping. **Journal of the Brazilian Computer Society**, v. 25, n. 12, p. 1–22, 2019. Citations on pages 47 and 65.

ALOTAIBI, A.; CLAUSE, J.; HALFOND, W. G. Mobile app energy consumption: A study of known energy issues in mobile applications and their classification schemes – summary plan. In: **Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)**. New York, NY, USA: IEEE, 2020. p. 854–854. Citations on pages 27 and 101.

AMALFITANO, D.; FASOLINO, A. R.; TRAMONTANA, P.; ROBBINS, B. Testing android mobile applications: Challenges, strategies, and approaches. In: MEMON, A. (Ed.). **Advances in Computers**. [S.l.]: Elsevier, 2013, (Advances in Computers, 1). p. 1–52. Citations on pages 44 and 45.

AMALFITANO, D.; RICCIO, V.; AMATUCCI, N.; SIMONE, V. D.; FASOLINO, A. R. Combining automated gui exploration of android apps with capture and replay through machine learning. **Information and Software Technology**, v. 105, p. 95–116, 2019. ISSN 0950-5849. Available: <https://www.sciencedirect.com/science/article/pii/S0950584918301708>. Citations on pages 85 and 86.

AMALFITANO, D.; RICCIO, V.; PAIVA, A. C. R.; FASOLINO, A. R. Why does the orientation change mess up my android application? from gui failures to code faults. **Software Testing,**

Verification and Reliability, v. 28, n. 1, p. e1654, 2018. E1654 stvr.1654. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1654>. Citation on page 80.

AMELLER, D.; AYALA, C.; CABOT, J.; FRANCH, X. How do software architects consider non-functional requirements: An exploratory study. In: **Proceedings of the 20th International Requirements Engineering Conference (RE)**. Chicago, IL, USA: IEEE, 2012. p. 41–50. Citation on page 37.

AMIN, A.; ELDESSOUKI, A.; MAGDY, M. T.; ABDEEN, N.; HINDY, H.; HEGAZY, I. Androshield: Automated android applications vulnerability detection, a hybrid static and dynamic analysis approach. **Information**, Multidisciplinary Digital Publishing Institute, v. 10, n. 10, p. 326, 2019. Citations on pages 69, 71, 73, and 77.

ANDRADE, S. A.; SANTOS, I.; JUNIOR, C. B.; JUNIOR, M. C.; SOUZA, S. R. S.; DELAMARO, M. E. On applying metamorphic testing: An empirical study on academic search engines. In: **Proceedings of the 4th International Workshop on Metamorphic Testing (MET)**. Montreal, QC, Canada: IEEE, 2019. p. 9–16. Citation on page 40.

ANDROID. **The getevent tool**. 2023. <https://source.android.com/docs/core/interaction/input/getevent>. [Online; accessed 19-December-2023]. Citation on page 84.

____. **uiautomator**. 2023. <https://stuff.mit.edu/afs/sipb/project/android/docs/tools/help/uiautomator/index.html>. [Online; accessed 19-December-2023]. Citation on page 84.

ARKIN, B.; STENDER, S.; MCGRAW, G. Software penetration testing. **IEEE Security & Privacy**, IEEE, v. 3, n. 1, p. 84–87, 2005. Citation on page 28.

ARZT, S.; RASTHOFER, S.; FRITZ, C.; BODDEN, E.; BARTEL, A.; KLEIN, J.; TRAON, Y. L.; OCTEAU, D.; MCDANIEL, P. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. **Acm Sigplan Notices**, ACM, v. 49, n. 6, p. 259–269, 2014. Citation on page 57.

AVANCINI, A.; CECCATO, M. Security testing of the communication among android applications. In: IEEE. **Proceedings of the 8th International Workshop on Automation of Software Test (AST)**. San Francisco, CA, USA, 2013. p. 57–63. Citations on pages 31 and 58.

____. Security testing of the communication among android applications. In: **Proceedings of the 8th International Workshop on Automation of Software Test (AST)**. New York, NY, USA: IEEE, 2013. p. 57–63. Citations on pages 69, 70, and 72.

AYERDI, J.; VALLE, P.; SEGURA, S.; ARRIETA, A.; SAGARDUI, G.; ARRATIBEL, M. Performance-driven metamorphic testing of cyber-physical systems. **Transactions on Reliability**, IEEE, 2022. Citations on pages 29, 30, and 101.

AZIMIAN, F.; FAGHIH, F.; KARGAHI, M.; MIRDEHGHAN, S. M. Energy metamorphic testing for android applications. In: **Proceedings of the 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)**. Istanbul, Turkey: IEEE, 2019. p. 1–6. Citations on pages 29 and 101.

BAGHERI, H.; WANG, J.; AERTS, J.; GHORBANI, N.; MALEK, S. Flair: efficient analysis of android inter-component vulnerabilities in response to incremental changes. **Empirical Software Engineering**, Springer, v. 26, p. 1–37, 2021. Citation on page 57.

BAJPAI, V.; GORTHI, R. P. On non-functional requirements: A survey. In: **Proceedings of the Students Conference on Electrical, Electronics and Computer Science (SCEECS)**. Bhopal, India: IEEE, 2012. p. 1–4. Citations on pages 38 and 53.

BARR, E.; HARMAN, M.; MCMINN, P.; SHAHBAZ, M.; YOO, S. The oracle problem in software testing: A survey. **IEEE Transactions on Software Engineering**, p. 507–525, 2015. Citations on pages 29 and 39.

BARUS, A. C.; CHEN, T. Y.; KUO, F.; LIU, H.; SCHMIDT, H. W. The impact of source test case selection on the effectiveness of metamorphic testing. In: **Proceedings of the International Workshop on Metamorphic Testing (MET)**. Austin, TX, USA: IEEE, 2016. p. 5–11. Citation on page 39.

BEYER, S.; MACHO, C.; PINZGER, M.; PENTA, M. D. Automatically classifying posts into question categories on stack overflow. In: **Proceedings of the 26th Conference on Program Comprehension**. Gothenburg, Sweden: IEEE, 2018. p. 211–221. Citation on page 37.

BHATNAGAR, S.; MALIK, Y.; BUTAKOV, S. Analysing data security requirements of android mobile banking application. In: **Proceedings of the International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments (ISDDC)**. Cham: Springer, 2018. p. 30–37. Citations on pages 69, 71, and 73.

BIANCHI, A.; GUSTAFSON, E.; FRATANTONIO, Y.; KRUEGEL, C.; VIGNA, G. Exploitation and mitigation of authentication schemes based on device-public information. In: **Proceedings of the 33rd Annual Computer Security Applications Conference**. Orlando, FL, USA: ACM, 2017. p. 16–27. Citation on page 60.

BOURQUE, P.; FAIRLEY, R. E.; SOCIETY, I. C. **Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0**. 3rd. ed. Washington, DC, USA: IEEE Computer Society Press, 2014. ISBN 0769551661. Citations on pages 72 and 75.

CALDIERA, V. R. B. G.; ROMBACH, H. D. The goal question metric approach. **Encyclopedia of software engineering**, John Wiley & Sons, v. 1, n. 1, p. 528–532, 1994. Citation on page 66.

CARACCIOLO, A.; LUNGU, M. F.; NIERSTRASZ, O. How do software architects specify and validate quality requirements? In: **European Conference on Software Architecture**. Vienna, Austria: Springer, 2014. p. 374–389. Citations on pages 37, 38, 51, 53, and 63.

CECI, L. **Annual number of app downloads from the Google Play Store**. 2022. <https://www.statista.com/statistics/734332/google-play-app-installs-per-year/>. [Online; accessed 18-December-2022]. Citations on pages 27 and 45.

CHALESHTARI, N. B.; PASTORE, F.; GOKNIL, A.; BRIAND, L. C. Metamorphic testing for web system security. **IEEE Transactions on Software Engineering**, IEEE, 2023. Citations on pages 31, 44, and 51.

____. Metamorphic testing for web system security. **IEEE Transactions on Software Engineering (TSE)**, IEEE, 2023. Citations on pages 31, 101, 103, and 104.

CHAN, W.; CHEN, T. Y.; CHEUNG, S. C.; TSE, T.; ZHANG, Z. Towards the testing of power-aware software applications for wireless sensor networks. In: **Proceedings of the International Conference on Reliable Software Technologies**. Geneva, Switzerland: Springer-Verlag, 2007. p. 84–99. Citations on pages 29, 42, 43, and 51.

CHAN, W.; CHEUNG, S. C.; LEUNG, K. R. Towards a metamorphic testing methodology for service-oriented software applications. In: **Proceedings of the 5th International Conference on Quality Software (QSIC)**. Melbourne, VIC, Australia: IEEE, 2005. p. 470–476. Citations on pages 29, 42, 43, and 51.

CHAN, W.; HO, J. C.; TSE, T. Piping classification to metamorphic testing: An empirical study towards better effectiveness for the identification of failures in mesh simplification programs. In: **Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC)**. Beijing, China: IEEE, 2007. p. 397–404. Citations on pages 29, 42, 43, and 51.

CHAN, W. K.; CHEUNG, S. C.; LEUNG, K. R. A metamorphic testing approach for online testing of service-oriented software applications. **International Journal of Web Services Research (IJWSR)**, p. 61–81, 2007. Citations on pages 29, 42, 43, and 51.

CHEN, T. Y.; CHEUNG, S. C.; YIU, S. M. **Metamorphic testing: a new approach for generating next test cases**. [S.l.], 1998. Citations on pages 29 and 39.

CHEN, T. Y.; KUO, F.-C.; LIU, H.; POON, P.-L.; TOWEY, D.; TSE, T. H.; ZHOU, Z. Q. Metamorphic testing: A review of challenges and opportunities. **ACM Computing Surveys (CSUR)**, p. 000:1–000:27, 2018. Citation on page 42.

CHEN, T. Y.; KUO, F.-C.; MA, W.; SUSILO, W.; TOWEY, D.; VOAS, J.; ZHOU, Z. Q. Metamorphic testing for cybersecurity. **Computer**, p. 48–55, 2016. Citations on pages 28, 43, and 51.

CHUNG, L.; NIXON, B. A.; YU, E.; MYLOPOULOS, J. **Non-functional requirements in software engineering**. [S.l.]: Springer Science & Business Media, 2012. Citations on pages 37 and 51.

COMMITTEE, I. S. C. *et al.* Ieee standard glossary of software engineering terminology (ieee std 610.12-1990). **CA: IEEE Computer Society**, 1990. Citation on page 34.

COPPOLA, R.; MORISIO, M.; TORCHIANO, M. Mobile gui testing fragility: A study on open-source android applications. **IEEE Transactions on Reliability**, p. 67–90, 2019. Citation on page 45.

CORRADINI, D.; PASQUA, M.; CECCATO, M. Automated black-box testing of mass assignment vulnerabilities in restful apis. In: **Proceedings of the 45th International Conference on Software Engineering (ICSE)**. Melbourne, Australia: IEEE, 2023. Citations on pages 29, 30, and 101.

CORTESI, A.; HILS, M.; KRIECHBAUMER, T.; CONTRIBUTORS. **mitmproxy: A free and open source interactive HTTPS proxy**. 2010. [Version 10.1]. Available: <https://mitmproxy.org/>. Citation on page 86.

CRUZ, L.; ABREU, R.; LO, D. To the attention of mobile software developers: guess what, test your app! **Empirical Software Engineering**, p. 1–31, 2019. Citation on page 48.

CRUZES, D.; MENDONCA, M.; BASILI, V.; SHULL, F.; JINO, M. Extracting information from experimental software engineering papers. In: **Proceedings of the 26th International Conference of the Chilean Society of Computer Science (SCCC)**. Iquique, Chile: IEEE, 2007. p. 105–114. Citation on page 90.

CWE. **CWE Top 25 Most Dangerous Software Weaknesses**. 2021. <https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html>. [Online; accessed 03-October-2023]. Citations on pages 29, 31, 60, and 104.

____. **Common Weakness Enumeration**. 2023. <https://cwe.mitre.org/>. [Online; accessed 03-October-2023]. Citations on pages 31, 103, and 104.

____. **CWE-288: Authentication Bypass Using an Alternate Path or Channel**. 2023. <https://cwe.mitre.org/data/definitions/288.html>. [Online; accessed 25-December-2023]. Citations on pages 87 and 89.

____. **CWE-295: Improper Certificate Validation**. 2023. <https://cwe.mitre.org/data/definitions/295.html>. [Online; accessed 25-December-2023]. Citations on pages 87 and 89.

____. **CWE-311: Missing Encryption of Sensitive Data**. 2023. <https://cwe.mitre.org/data/definitions/311.html>. [Online; accessed 25-December-2023]. Citations on pages 87 and 89.

____. **CWE-384: Session Fixation**. 2023. <https://cwe.mitre.org/data/definitions/384.html>. [Online; accessed 25-December-2023]. Citations on pages 87 and 89.

____. **CWE-613: Insufficient Session Expiration**. 2023. <https://cwe.mitre.org/data/definitions/613.html>. [Online; accessed 25-December-2023]. Citations on pages 87 and 89.

ECKHARDT, J.; VOGELSANG, A.; FERNÁNDEZ, D. M. Are "non-functional" requirements really non-functional? an investigation of non-functional requirements in practice. In: **Proceedings of the 38th International Conference on Software Engineering (ICSE)**. Austin, TX, USA: IEEE, 2016. p. 832–842. Citations on pages 28, 34, and 37.

FELDERER, M.; BÜCHLER, M.; JOHNS, M.; BRUCKER, A. D.; BREU, R.; PRETSCHNER, A. Security testing: A survey. In: **Advances in Computers**. [S.l.]: Elsevier, 2016. v. 101, p. 1–51. Citations on pages 28, 54, 55, 56, and 57.

GAMMA, E.; BECK, K. **JUnit**. 2017. [Online; accessed 06-February-2024]. Available: <https://junit.org/>. Citation on page 85.

GAO, J.; BAI, X.; TSAI, W.-T.; UEHARA, T. Mobile application testing: a tutorial. **Computer**, p. 46–55, 2014. Citation on page 45.

GLINZ, M. On non-functional requirements. In: **Proceedings of the 15th International Requirements Engineering Conference**. Delhi, India: IEEE, 2007. p. 21–26. Citations on pages 34 and 38.

GROEN, E. C.; KOPCZYŃSKA, S.; HAUER, M. P.; KRAFFT, T. D.; DOERR, J. Users—the hidden software product quality experts?: A study on how app users report quality aspects in online reviews. In: **Proceedings of the 25th International Requirements Engineering Conference (RE)**. Lisbon, Portugal: IEEE, 2017. p. 80–89. Citation on page 37.

GUO, C.; XU, J.; YANG, H.; ZENG, Y.; XING, S. An automated testing approach for inter-application security in android. In: **Proceedings of the 9th international workshop on automation of software test**. Hyderabad, India: ACM, 2014. p. 8–14. Citation on page 58.

____. An automated testing approach for inter-application security in android. In: **Proceedings of the 9th International Workshop on Automation of Software Test (AST)**. New York, NY, USA: ACM, 2014. p. 8–14. Citations on pages 69, 71, and 72.

HASSANSHAHI, B.; JIA, Y.; YAP, R. H.; SAXENA, P.; LIANG, Z. Web-to-application injection attacks on android: Characterization and detection. In: **Proceedings of the 20th European Symposium on Research in Computer Security**. Vienna, Austria: Springer, 2015. p. 577–598. Citations on pages 58 and 59.

HAY, R.; TRIPP, O.; PISTOIA, M. Dynamic detection of inter-application communication vulnerabilities in android. In: **Proceedings of the International Symposium on Software Testing and Analysis**. Baltimore, MD, USA: ACM, 2015. p. 118–128. Citation on page 58.

____. Dynamic detection of inter-application communication vulnerabilities in android. In: **Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)**. New York, NY, USA: ACM, 2015. p. 118–128. Citations on pages 69, 71, 73, and 77.

HENRY, G. Matt lacey on mobile app usability. **IEEE Softw.**, v. 38, n. 2, p. 134–136, 2021. Available: <https://doi.org/10.1109/MS.2020.3042424>. Citation on page 27.

HOFFMAN, D. Using oracles in test automation. **Proceedings of the 19th Annual Pacific Northwest Software Quality Conference (PNSQC)**, p. 90–117, 2001. Citation on page 39.

HUANG, Y.-W.; HUANG, S.-K.; LIN, T.-P.; TSAI, C.-H. Web application security assessment by fault injection and behavior monitoring. In: **Proceedings of the 12th International Conference on World Wide Web**. Budapest, Hungary: ACM, 2003. p. 148–159. Citations on pages 30 and 44.

ISO. **ISO/IEC 9126-1, Software engineering — Product quality**. Geneva, Switzerland: ISO, 2001. Citations on pages 35, 36, 37, and 54.

ISO. **ISO/IEC 25010:2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models**. Geneva, Switzerland: ISO, 2011. Citations on pages 35, 36, 37, 49, 54, 101, and 102.

ISO. **ISO/IEC/IEEE 29119-1: Software and systems engineering-Software testing-Part 1: Concepts and definitions**. [S.l.]: International Organization for Standardization Geneva, 2013. Citation on page 35.

JABBARVAND, R.; MALEK, S. Advancing energy testing of mobile applications. In: **Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C)**. Buenos Aires, Argentina: IEEE, 2017. p. 491–492. Citation on page 50.

JANICKI, M.; KATARA;, M.; PääKKöNEN, T. Obstacles and opportunities in deploying model-based gui testing of mobile software: A survey. **SOFTWARE TESTING, VERIFICATION AND RELIABILITY**, v. 22(5), p. 313–341, Aug. 2012. Citation on page 46.

JIANG, Y. Z. X.; XUXIAN, Z. Detecting passive content leaks and pollution in android applications. In: **Proceedings of the 20th Network and Distributed System Security Symposium (NDSS)**. San Diego, California: [s.n.], 2013. Citations on pages 29, 42, 43, 51, and 57.

JOHNSTON, O.; JARMAN, D.; BERRY, J.; ZHOU, Z. Q.; CHEN, T. Y. Metamorphic relations for detection of performance anomalies. In: **Proceedings of the 4th International Workshop on Metamorphic Testing (MET)**. Montreal, QC, Canada: IEEE, 2019. p. 63–69. Citations on pages 29, 30, and 101.

JOORABCHI, M. E.; MESBAH, A.; KRUCHTEN, P. Real challenges in mobile app development. In: **Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)**. Baltimore, MD, USA: IEEE, 2013. p. 15–24. Citations on pages 48 and 50.

JUNIOR, M. C. Automated verification of compliance of non-functional requirements on mobile applications through metamorphic testing. In: **Proceedings of the 13th International Conference on Software Testing, Validation and Verification (ICST)**. Porto, Portugal: IEEE, 2020. p. 421–423. Citation on page 33.

JUNIOR, M. C.; AMALFITANO, D.; GARCES, L.; FASOLINO, A. R.; ANDRADE, S. A.; DELAMARO, M. Dynamic testing techniques of non-functional requirements in mobile apps: A systematic mapping study. **ACM Computing Surveys (CSUR)**, ACM New York, NY, v. 54, n. 10s, p. 1–38, 2022. Citations on pages 28, 33, 35, 47, 50, 51, 63, 66, 67, 68, 69, 70, 71, 73, 76, 77, 78, 92, 101, 102, and 123.

JUNIOR, M. C.; AMALFITANO, D.; VISONE, B.; FASOLINO, A. R.; DELAMARO, M. A metamorphic testing technique for detecting authentication vulnerabilities in android mobile apps. **Journal of Systems & Software (JSS)**, p. 1–24, 2023. Citations on pages 53, 61, 79, 80, 82, 83, 84, 87, 89, 91, 94, 95, 96, and 102.

KAUR, A.; KAUR, K. Systematic literature review of mobile application development and testing effort estimation. **Journal of King Saud University - Computer and Information Sciences**, v. 1, n. 1, p. 1–22, 2018. ISSN 1319-1578. Citation on page 46.

KENG, J. C. J.; JIANG, L.; WEE, T. K.; BALAN, R. K. Graph-aided directed testing of android applications for checking runtime privacy behaviours. In: **Proceedings of the 11th International Workshop on Automation of Software Test**. Austin, TX, USA: IEEE, 2016. p. 57–63. Citations on pages 58 and 59.

____. Graph-aided directed testing of android applications for checking runtime privacy behaviours. In: **Proceedings of the 11th International Workshop on Automation of Software Test ((AST))**. New York, NY, USA: ACM, 2016. p. 57–63. Citations on pages 69, 71, and 72.

KIRUBAKARAN, B.; KARTHIKEYANI, V. Mobile application testing – challenges and solution approach through automation. In: **Proceedings of the International Conference on Pattern Recognition, Informatics and Mobile Engineering**. Salem, India: IEEE, 2013. p. 79–84. Citations on pages 45 and 49.

KITCHENHAM, B. Procedures for performing systematic reviews. **Keele University**, v. 33, n. 2004, p. 1–26, 2004. Citation on page 66.

KLUTH, W.; KREMPELS, K.-H.; SAMSEL, C. Automated usability testing for mobile applications. In: **Proceedings of the Web Information Systems and Technologies (WEBIST)**. Setúbal, Portugal: SciTePress, 2014. p. 149–156. Citation on page 50.

KNORR, K.; ASPINALL, D. Security testing for android mhealth apps. In: **Proceedings of the 8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. Graz, Austria: IEEE, 2015. p. 1–8. Citation on page 58.

____. Security testing for android mhealth apps. In: **Proceedings of the 8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. New York, NY, USA: IEEE, 2015. p. 1–8. Citations on pages 69, 71, and 73.

KOCHHAR, P. S.; THUNG, F.; NAGAPPAN, N.; ZIMMERMANN, T.; LO, D. Understanding the test automation culture of app developers. In: **Proceedings of the 8th International Conference on Software Testing, Verification and Validation (ICST)**. Graz, Austria: IEEE, 2015. p. 1–10. Citation on page 50.

KONG, P.; LI, L.; GAO, J.; LIU, K.; BISSYANDé, T. F.; KLEIN, J. Automated testing of android apps: A systematic literature review. **IEEE Transactions on Reliability**, v. 68, n. 1, p. 45–66, 2019. Citations on pages 47 and 65.

LARICCHIA, F. **Mobile operating systems' market share worldwide**. 2022. <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/>. [Online; accessed 18-December-2022]. Citations on pages 27 and 45.

LEE, W. van der; VERWER, S. Vulnerability detection on mobile applications using state machine inference. In: **Proceedings of the European Symposium on Security and Privacy Workshops (EuroS&PW)**. New York, NY, USA: IEEE, 2018. p. 1–10. Citations on pages 69, 71, 73, and 77.

LI, L.; BARTEL, A.; KLEIN, J.; TRAON, Y. L. Automatically exploiting potential component leaks in android applications. In: **Proceedings of the 13th International Conference on Trust, Security and Privacy in Computing and Communications**. Beijing, China: IEEE, 2014. p. 388–397. Citation on page 57.

LIANG, H.; WANG, Y.; YANG, T.; YU, Y. Applance: A lightweight approach to detect privacy leak for packed applications. In: **Proceedings of the Nordic Conference on Secure IT Systems**. Oslo, Norway: Springer, 2018. p. 54–70. Citation on page 57.

____. Applance: A lightweight approach to detect privacy leak for packed applications. In: **Proceedings of the Nordic Conference on Secure IT Systems**. Cham: Springer, 2018. p. 54–70. Citations on pages 69 and 73.

LIU, Y.; ZUO, C.; ZHANG, Z.; GUO, S.; XU, X. An automatically vetting mechanism for ssl error-handling vulnerability in android hybrid web apps. **World Wide Web**, Springer, v. 21, n. 1, p. 127–150, 2018. Citations on pages 31 and 58.

____. An automatically vetting mechanism for ssl error-handling vulnerability in android hybrid web apps. **World Wide Web**, Springer, v. 21, p. 127–150, 2018. Citations on pages 69, 71, and 73.

LÓPEZ, L.; PARTANEN, J.; RODRÍGUEZ, P.; MARTÍNEZ-FERNÁNDEZ, S. How practitioners manage quality requirements in rapid software development: A survey. In: **Proceedings of the 1st International Workshop on Quality Requirements in Agile Projects (QuaRAP)**. Banff, AB, Canada: IEEE, 2018. p. 14–17. Citations on pages 37, 38, 53, and 63.

LORTZ, S.; MANTEL, H.; STAROSTIN, A.; BÄHR, T.; SCHNEIDER, D.; WEBER, A. Cassandra: Towards a certifying app store for android. In: **Proceedings of the 4th Workshop on Security and Privacy in Smartphones & Mobile Devices**. Scottsdale, Arizona, USA: ACM, 2014. p. 93–104. Citation on page 57.

LUO, C.; GONCALVES, J.; VELLOSO, E.; KOSTAKOS, V. A survey of context simulation for testing mobile context-aware applications. **ACM Comput. Surv.**, Association for Computing

Machinery, New York, NY, USA, v. 53, n. 1, Feb. 2020. ISSN 0360-0300. Available: <https://doi.org/10.1145/3372788>. Citations on pages 47 and 65.

MA, S.; BERTINO, E.; NEPAL, S.; LI, J.; OSTRY, D.; DENG, R. H.; JHA, S. Finding flaws from password authentication code in android apps. In: **Proceedings of the European Symposium on Research in Computer Security**. Luxembourg, Luxembourg: Springer, 2019. p. 619–637. Citations on pages 60 and 61.

MAHMOOD, R.; ESFAHANI, N.; KACEM, T.; MIRZAEI, N.; MALEK, S.; STAVROU, A. A whitebox approach for automated security testing of android applications on the cloud. In: **Proceedings of the 7th International Workshop on Automation of Software Test (AST)**. Zurich, Switzerland: IEEE, 2012. p. 22–28. Citations on pages 58 and 59.

MAI, P. X.; PASTORE, F.; GOKNIL, A.; BRIAND, L. Metamorphic security testing for web systems. **arXiv preprint arXiv:1912.05278**, 2019. Citations on pages 29, 30, 31, 44, 51, 98, 101, 103, and 104.

MAIA, V.; GONÇALVES, T. G.; ROCHA, A. R. C. da. Quality characteristics of mobile applications: A survey in brazilian context. In: **Proceedings of the 18th Brazilian Symposium on Software Quality**. Fortaleza, Brazil: ACM, 2019. p. 109–118. Citations on pages 49 and 51.

MAIA, V.; ROCHA, A. R. C. da. **A Systematic Literature Mapping for Identifying Quality Characteristics in Mobile Applications**. [S.l.], 2019. Available: <https://maia.mobi/arq/tech_report_mapping.pdf>. Citations on pages 49 and 51.

MAIRIZA, D.; ZOWGHI, D.; NURMULIANI, N. An investigation into the notion of non-functional requirements. In: **Proceedings of the 2010 ACM Symposium on Applied Computing**. Sierre, Switzerland: ACM, 2010. p. 311–317. Citations on pages 37 and 51.

MAYER, J.; GUDERLEI, R. On random testing of image processing applications. In: **Proceedings of the 6th International Conference on Quality Software (QSIC)**. Beijing, China: IEEE, 2006. p. 85–92. Citations on pages 29, 42, 43, and 51.

MCAFEE, L. Mcafee labs 2018 threats predictions. **Mission College Boulevard**, 2017. Citations on pages 28, 53, and 54.

MUCCINI, H.; FRANCESCO, A. D.; ESPOSITO, P. Software testing of mobile applications: Challenges and future research directions. In: **Proceedings of the 7th International Workshop on Automation of Software Test**. Zurich, Switzerland: IEEE, 2012. p. 29–35. Citations on pages 44 and 49.

MUELLER, B.; SCHLEIER, S.; WILLEMSEN, J. **Mobile Security Testing Guide (MSTG)**. [S.l.]: OWASP, 2019. Citations on pages 31, 62, and 102.

MURPHY, C.; KAISER, G. E.; HU, L. Properties of machine learning applications for use in metamorphic testing. 2008. Citations on pages 29, 42, 43, and 51.

NAGAPPAN, M.; SHIHAB, E. Future trends in software engineering research for mobile apps. In: **Proceedings of the 23rd on Software Analysis, Evolution, and Reengineering (SANER)**. Osaka, Japan: IEEE, 2016. p. 21–32. Citations on pages 28, 45, 50, 51, and 63.

NAKAJIMA, S.; BUI, H. N. Dataset coverage for testing machine learning computer programs. In: **Proceedings of the Asia-Pacific Software Engineering Conference (APSEC)**. Hamilton, New Zealand: IEEE, 2016. p. 297–304. Citations on pages 29, 42, 43, and 51.

NETO, A. C. D.; SANTOS, J. S.; REIS, R. A. C.; LOBãO, L. M. A.; COLLINS, E. F. Capítulo 10 – teste em dispositivos móveis. In: **Introdução ao Teste de Software**. 2. ed. Rio de Janeiro: Campus, 2016. p. 297–329. Citations on pages 45, 48, and 51.

OAUTH. **OAuth 2.0**. 2012. <https://oauth.net/2/>. Online; accessed 10-December-2023. Citations on pages 61, 79, 87, and 98.

OCTEAU, D.; LUCHAUP, D.; DERING, M.; JHA, S.; MCDANIEL, P. Composite constant propagation: Application to android inter-component communication analysis. In: **Proceedings of the 37th IEEE International Conference on Software Engineering**. Florence, Italy: IEEE, 2015. v. 1, p. 77–88. Citation on page 57.

OCTEAU, D.; MCDANIEL, P.; JHA, S.; BARTEL, A.; BODDEN, E.; KLEIN, J.; TRAON, Y. L. Effective inter-component communication mapping in android: An essential step towards holistic security analysis. In: **Proceedings of the 22nd {USENIX} Security Symposium ({USENIX} Security)**. Washington, D.C.: USENIX Association, 2013. p. 543–558. Citation on page 57.

O'DEA, S. **Smartphone users worldwide 2016-2021**. 2020. <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>. [Online; accessed 10-January-2020]. Citations on pages 27 and 44.

OPASIAK, K.; MAZURCZYK, W. (in) secure android debugging: Security analysis and lessons learned. **Computers & Security**, v. 82, p. 80–98, 2019. Citations on pages 31, 50, and 53.

OWASP. **M2: Insecure Data Storage**. 2016. <https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage>. [Online; accessed 18-December-2023]. Citation on page 28.

____. **M3: Insecure Communication**. 2016. <https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication>. [Online; accessed 18-December-2023]. Citations on pages 28 and 87.

____. **M4: Insecure Authentication**. 2016. <https://owasp.org/www-project-mobile-top-10/2016-risks/m4-insecure-authentication>. [Online; accessed 18-December-2023]. Citations on pages 28 and 87.

____. **Mobile App Authentication Architectures – OWASP MASTG**. 2023. <https://mobile-security.gitbook.io/mobile-security-testing-guide/general-mobile-app-testing-guide/0x04e-testing-authentication-and-session-management>. Online; accessed 17-March-2024. Citations on pages 29 and 59.

____. **OWASP Mobile Top 10**. 2023. <https://owasp.org/www-project-mobile-top-10/>. [Online; accessed 02-November-2023]. Citations on pages 29, 31, 59, 60, 102, 103, and 104.

PAYET, É.; SPOTO, F. Static analysis of android programs. **Information and Software Technology**, Elsevier, v. 54, n. 11, p. 1192–1201, 2012. Citation on page 57.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. **Information and Software Technology**, Butterworth-Heinemann, v. 64, n. C, p. 1–18, 2015. Citations on pages 66, 67, and 102.

POORT, E. R.; MARTENS, N.; WEERD, I. V. D.; VLIET, H. V. How architects see non-functional requirements: beware of modifiability. In: **Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality**. Berlin, Heidelberg: Springer, 2012. p. 37–51. Citations on pages 37, 38, 53, and 63.

PORRAS, A. M.; LÓPEZ, C. U. Q.; CORONAS, M. J. Automated testing of mobile applications: A systematic map and review. In: **Proceedings of the 28th Ibero-American Conference on Software Engineering (CIBSE)**. Lima, Peru: URP,SPC,UCSP, 2015. p. 1–14. Citations on pages 46 and 65.

PORTSWIGGER. **Burp Suite**. 2016. <https://portswigger.net/burp>. [Online; accessed 19-December-2023]. Citation on page 86.

POTTER, B.; MCGRAW, G. Software security testing. **IEEE Security & Privacy**, IEEE, v. 2, n. 5, p. 81–85, 2004. Citations on pages 28 and 55.

PRESSMAN, R. **Software Engineering: A Practitioner's Approach**. 9. ed. USA: McGraw-Hill, Inc., 2016. ISBN 0073375977. Citation on page 34.

PULLUM, L. L.; OZMEN, O. Early results from metamorphic testing of epidemiological models. In: **Proceedings of the International Conference on BioMedical Computing (BioMedCom)**. Washington, DC, USA: IEEE, 2012. p. 62–67. Citations on pages 29, 42, 43, and 51.

RAHMAN, K.; IZURIETA, C. An approach to testing banking software using metamorphic relations. In: **Proceedings of the 24th International Conference on Information Reuse and Integration for Data Science (IRI)**. Bellevue, WA, USA: IEEE, 2023. p. 173–178. Citations on pages 29, 31, 44, and 101.

RASHID, M.; ARDITO, L.; TORCHIANO, M. Energy consumption analysis of algorithms implementations. In: **Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)**. Beijing, China: IEEE, 2015. p. 1–4. Citation on page 37.

RASTOGI, V.; CHEN, Y.; ENCK, W. Appsplayground: automatic security analysis of smartphone applications. In: **Proceedings of the 3rd Conference on Data and Application Security and Privacy (CODASPY)**. San Antonio, Texas, USA: ACM, 2013. p. 209–220. Citations on pages 69, 72, and 73.

RIBEIRO, V. V.; CRUZES, D. S.; TRAVASSOS, G. H. A perception of the practice of software security and performance verification. In: **Proceedings of the 25th Australasian Software Engineering Conference (ASWEC)**. Adelaide, SA, Australia: IEEE, 2018. p. 71–80. Citations on pages 28, 38, 39, 51, 53, 54, and 63.

RIBEIRO, V. V.; TRAVASSOS, G. H. Testing non-functional requirements: Lacking of technologies or researching opportunities? In: **Proceedings of the 15th Brazilian Symposium on Software Quality**. Porto Alegre, RS, Brasil: SBC, 2017. p. 226–240. Citations on pages 38 and 51.

ROMDHANA, A.; MERLO, A.; CECCATO, M.; TONELLA, P. Assessing the security of inter-app communications in android through reinforcement learning. **Computers & Security**, Elsevier, v. 131, p. 103311, 2023. Citation on page 58.

ROSEN, C.; SHIHAB, E. What are mobile developers asking about? a large scale study using stack overflow. **Empirical Software Engineering**, p. 1192–1223, 2016. Citation on page 37.

RUMEE, S. T. A.; LIU, D. Droidtest: Testing android applications for leakage of private information. In: **Proceedings of the 16th International Conference on Security (ISC)**. USA: Springer, 2015. p. 341–353. Citations on pages 58 and 59.

SAAD, E.; MITCHELL, R. **Web Security Testing Guide (WSTG)**. [S.l.]: OWASP, 2019. Citation on page 62.

SAHINOGLU, M.; INCKI, K.; AKTAS, M. S. Mobile application verification: A systematic mapping study. In: **Proceedings of the International Computational Science and Its Applications (ICCSA)**. Cham: Springer, 2015. p. 147–163. Citation on page 46.

SALVA, S.; ZAFIMIHARISOA, S. R. Apset, an android application security testing tool for detecting intent-based vulnerabilities. **International Journal on Software Tools for Technology Transfer**, Springer, v. 17, n. 2, p. 201–221, 2015. Citation on page 57.

____. Apset, an android application security testing tool for detecting intent-based vulnerabilities. **International Journal on Software Tools for Technology Transfer**, Springer, v. 17, n. 2, p. 201–221, 2015. Citations on pages 69, 70, 72, and 77.

SECURITY, S. **Vulnerability and Threat Trends Mid-Year Report 2020**. 2020. <https://www.skyboxsecurity.com/resources/report/vulnerability-threat-trends-mid-year-report-2020/>. [Online; accessed 03-October-2023]. Citation on page 28.

SEGURA, S.; CASTILLA, J. T.; TORO, A. D.; CORTÉS, A. R. Performance metamorphic testing: A proof of concept. **Information and Software Technology**, 2018. Citations on pages 29, 43, 51, and 101.

SEGURA, S.; DURÁN, A.; TROYA, J.; CORTÉS, A. R. A template-based approach to describing metamorphic relations. In: **Proceedings of the 2nd International Workshop on Metamorphic Testing (MET)**. Buenos Aires, Argentina: IEEE, 2017. p. 3–9. Citation on page 88.

SEGURA, S.; FRASER, G.; SANCHEZ, A. B.; RUIZ-CORTÉS, A. A survey on metamorphic testing. **IEEE Transactions on software engineering**, p. 805–824, 2016. Citations on pages 29, 39, 40, 41, and 42.

SEGURA, S.; TROYA, J.; DURÁN, A.; RUIZ-CORTÉS, A. Performance metamorphic testing: motivation and challenges. In: **Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)**. Buenos Aires, Argentina: IEEE, 2017. p. 7–10. Citation on page 43.

SEGURA, S.; ZHOU, Z. Q. Metamorphic testing 20 years later: A hands-on introduction. In: **Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings**. Gothenburg, Sweden: IEEE, 2018. p. 538–539. Citation on page 41.

SEQUEIROS, J. B. F.; CHIMUCO, F. T.; SAMAILA, M. G.; FREIRE, M. M.; INÁCIO, P. R. M. Attack and system modeling applied to iot, cloud, and mobile ecosystems: Embedding security by design. **ACM Computing Surveys (CSUR)**, Association for Computing Machinery, v. 53, n. 2, p. 25:1–25:32, 2020. Citation on page 27.

SHAHRIAR, H.; NORTH, S.; MAWANGI, E. Testing of memory leak in android applications. In: **Proceedings of the 15th International Symposium on High-Assurance Systems Engineering**. Miami Beach, FL, USA: IEEE, 2014. p. 176–183. Citation on page 57.

____. Testing of memory leak in android applications. In: **Proceedings of the 15th International Symposium on High-Assurance Systems Engineering**. New York, NY, USA: IEEE, 2014. p. 176–183. Citations on pages 69, 72, and 73.

SHI, S.; WANG, X.; LAU, W. C. Mossot: An automated blackbox tester for single sign-on vulnerabilities in mobile applications. In: **Proceedings of the Asia Conference on Computer and Communications Security (ASIACCS)**. New York, NY, USA: ACM, 2019. p. 269–282. Citations on pages 31, 69, 71, 72, and 77.

____. Mossot: An automated blackbox tester for single sign-on vulnerabilities in mobile applications. In: **Proceedings of the Asia Conference on Computer and Communications Security**. Auckland, New Zealand: ACM, 2019. p. 269–282. Citation on page 58.

SOMMERVILLE, I. **Software Engineering**. 9th. ed. USA: Addison-Wesley Publishing Company, 2011. ISBN 0137035152. Citation on page 34.

STAROV, O.; VILKOMIR, S.; GORBENKO, A.; KHARCHENKO, V. **Testing-as-a-Service for Mobile Applications: State-of-the-Art Survey**. Cham: Springer, 2015. Citation on page 46.

SUN, C.-A.; DAI, H.; GENG, N.; LIU, H.; CHEN, T. Y.; WU, P.; CAI, Y.; WANG, J. An interleaving guided metamorphic testing approach for concurrent programs. **ACM Transactions on Software Engineering and Methodology**, ACM New York, NY, v. 33, n. 1, p. 1–21, 2023. Citations on pages 42 and 43.

TIAN-YANG, G.; YIN-SHENG, S.; YOU-YUAN, F. Research on software security testing. **World Academy of science, engineering and Technology**, Citeseer, v. 70, p. 647–651, 2010. Citation on page 55.

TRAMONTANA, P.; AMALFITANO, D.; AMATUCCI, N.; FASOLINO, A. R. Automated functional testing of mobile applications: a systematic mapping study. **Software Quality Journal**, v. 27, n. 1, p. 149–201, 2019. Citations on pages 47 and 65.

TSE, T.; YAU, S. S. Testing context-sensitive middleware-based software applications. In: **Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.** Hong Kong, China: IEEE, 2004. p. 458–466. Citations on pages 29, 42, 43, and 51.

VARA, J. L. D. L.; WNUK, K.; BERNTSSON-SVENSSON, R.; SÁNCHEZ, J.; REGNELL, B. An empirical study on the importance of quality requirements in industry. In: **Proceedings of the Software Engineering and Knowledge Engineering (SEKE)**. Miami Beach, Florida, USA: ACM, 2011. p. 438–443. Citation on page 37.

VASQUEZ, M. L.; BERNAL-CÁRDENAS, C.; MORAN, K.; POSHYVANYK, D. How do developers test android applications? In: **Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)**. Shanghai, China: IEEE, 2017. Citation on page 48.

VASQUEZ, M. L.; MORAN, K.; POSHYVANYK, D. Continuous, evolutionary and large-scale: A new perspective for automated mobile app testing. In: **Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)**. Shanghai, China: IEEE, 2017. Citations on pages 48 and 49.

VILLAMIZAR, H.; KALINOWSKI, M.; GARCIA, A.; MENDEZ, D. An efficient approach for reviewing security-related aspects in agile requirements specifications of web applications. **Requirements Engineering**, Springer, v. 25, n. 4, p. 439–468, 2020. Citation on page 28.

WANG, B.; YANG, C.; MA, J. Iafdroid: Demystifying collusion attacks in android ecosystem via precise inter-app analysis. **IEEE Transactions on Information Forensics and Security**, IEEE, 2023. Citation on page 57.

WANG, T.; LIANG, P.; LU, M. What aspects do non-functional requirements in app user reviews describe? an exploratory and comparative study. In: **Proceedings of the 25th Asia-Pacific Software Engineering Conference (APSEC)**. Nara, Japan: IEEE, 2018. p. 494–503. Citation on page 50.

WANG, Y.; XU, G.; LIU, X.; MAO, W.; SI, C.; PEDRYCZ, W.; WANG, W. Identifying vulnerabilities of ssl/tls certificate verification in android apps with static and dynamic analysis. **Journal of Systems and Software**, Elsevier, v. 167, p. 110609, 2020. Citations on pages 31 and 58.

____. Identifying vulnerabilities of ssl/tls certificate verification in android apps with static and dynamic analysis. **Journal of Systems and Software**, Elsevier, v. 167, p. 110609, 2020. Citations on pages 62, 69, 71, 72, and 73.

WASSERMAN, T. Software engineering issues for mobile application development. In: **Proceedings of the FSE/SDP workshop on Future of software engineering research**. Santa Fe, New Mexico, USA: ACM, 2010. p. 397–400. Citation on page 45.

WEI, X.; WOLF, M. A survey on https implementation by android apps: issues and countermeasures. **Applied Computing and Informatics**, Elsevier, v. 13, n. 2, p. 101–117, 2017. Citation on page 62.

WEYUKER, E. J. On testing non-testable programs. **The Computer Journal**, p. 465–470, 1982. Citations on pages 29 and 39.

XIE, J.; LIPFORD, H. R.; CHU, B. Why do programmers make security errors? In: **Proceedings of the Symposium on Visual Languages and Human-Centric Computing (VL/HCC)**. Pittsburgh, PA, USA: IEEE, 2011. p. 161–164. Citation on page 28.

YANG, K.; ZHUGE, J.; WANG, Y.; ZHOU, L.; DUAN, H. Intentfuzzer: detecting capability leaks of android applications. In: **Proceedings of the 9th Symposium on Information, computer and communications security**. New York, NY, USA: ACM, 2014. p. 531–536. Citations on pages 69, 71, and 73.

YA'U, B. I.; SALLEH, N.; NORDIN, A.; IDRIS, N. B.; ABAS, H.; ALWAN, A. A. A systematic mapping study on cloud-based mobile application testing. **Journal of Information and Communication Technology**, v. 18, n. 4, p. 485–527, 2019. Citation on page 47.

YEH, C. C.; LU, H. L.; CHEN, C. Y.; KHOR, K. K.; HUANG, S. K. Craxdroid: Automatic android system testing by selective symbolic execution. In: **Proceedings of the 8th International Conference on Software Security and Reliability-Companion**. San Francisco, CA, USA: IEEE, 2014. p. 140–148. Citations on pages 58 and 59.

YU, S.; FANG, C.; YUN, Y.; FENG, Y. Layout and image recognition driving cross-platform automated mobile testing. In: **Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering (ICSE)**. New York, NY, USA: IEEE, 2021. p. 1561–1571. Citations on pages 27 and 101.

YUSOP, N.; KAMALRUDIN, M.; SIDEK, S.; GRUNDY, J. Automated support to capture and validate security requirements for mobile apps. In: **Proceedings of the Asia Pacific Requirements Engineering Conference (APSEC)**. Singapore: Springer, 2016. p. 97–112. Citations on pages 69, 70, and 72.

ZEIN, S.; SALLEH, N.; GRUNDY, J. A systematic mapping study of mobile application testing techniques. **Journal of Systems and Software**, Elsevier Science Inc., USA, v. 117, n. C, p. 334–356, Jul. 2016. ISSN 0164-1212. Available: <https://doi.org/10.1016/j.jss.2016.03.065>. Citations on pages 46 and 65.

ZHANG, J.; TIAN, C.; DUAN, Z. An efficient approach for taint analysis of android applications. **Computers & Security**, Elsevier, v. 104, p. 102161, 2021. Citation on page 57.

ZHOU, Z. Q.; TSE, T.; KUO, F.; CHEN, T. Automated functional testing of web search engines in the absence of an oracle. **Department of Computer Science, The University of Hong Kong, Tech. Rep. TR-2007-06**, 2007. Citations on pages 29, 42, 43, and 51.

ZHOU, Z. Q.; XIANG, S.; CHEN, T. Y. Metamorphic testing for software quality assessment: A study of search engines. **IEEE Transactions on Software Engineering**, p. 264–284, 2016. Citation on page 39.

ZOU, J.; XU, L.; YANG, M.; ZHANG, X.; YANG, D. Towards comprehending the non-functional requirements through developers' eyes: An exploration of stack overflow using topic analysis. **Information and Software Technology**, p. 19–32, 2017. Citation on page 37.

APPENDIX

A

SYSTEMATIC MAPPING PAPER

This Appendix includes the published article titled *Dynamic testing techniques of non-functional requirements in mobile apps: A systematic mapping study*, which provides a detailed description of the systematic mapping process, methodology, and findings as part of this PhD research. The article has been submitted, accepted, and published at the ACM Computing Surveys (CSUR) (JUNIOR *et al.*, 2022). Readers are encouraged to refer to the article for more detailed information on the systematic mapping and its results.

- JUNIOR, M. C., AMALFITANO, D., GARCES, L., FASOLINO, A. R., ANDRADE, S. A., DELAMARO, M. (2022). Dynamic testing techniques of non-functional requirements in mobile apps: A systematic mapping study. **ACM Computing Surveys (CSUR)**, 54(10s), 1-38. Available: <https://doi.org/10.1145/3507903>.

Dynamic Testing Techniques of Non-Functional Requirements in Mobile Apps: A Systematic Mapping Study

MISAEL C. JÚNIOR, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, ICMC-USP, Brazil
DOMENICO AMALFITANO, Department of Electrical Engineering and Information Technology, University of Naples Federico II, DIETI-UNINA, Italy
LINA GARCÉS, Instituto de Matemática e Computação, Universidade Federal de Itajubá, IMC-UNIFEI, Brazil
ANNA RITA FASOLINO, Department of Electrical Engineering and Information Technology, University of Naples Federico II, DIETI-UNINA, Italy
STEVÃO A. ANDRADE and MÁRCIO DELAMARO, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, ICMC-USP, Brazil

Context: The mobile app market is continually growing offering solutions to almost all aspects of people's lives, e.g., healthcare, business, entertainment, as well as the stakeholders' demand for apps that are more secure, portable, easy to use, among other Non-Functional Requirements (NFRs). Therefore, manufacturers should guarantee that their mobile apps achieve high-quality levels. A good strategy is to include software testing and quality assurance activities during the whole life cycle of such solutions.
Problem: Systematically warranting NFRs is not an easy task for any software product. Software engineers must take important decisions before adopting testing techniques and automation tools to support such endeavors.
Proposal: To provide to the software engineers with a broad overview of existing dynamic techniques and automation tools for testing mobile apps regarding NFRs.
Methods: We planned and conducted a Systematic Mapping Study (SMS) following well-established guidelines for executing secondary studies in software engineering.
Results: We found 56 primary studies and characterized their contributions based on testing strategies, testing approaches, explored mobile platforms, and the proposed tools.
Conclusions: The characterization allowed us to identify and discuss important trends and opportunities that can benefit both academics and practitioners.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Software and its engineering** → **Extra-functional properties**.

Authors' addresses: Misael C. Júnior, misaeljr@usp.br, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, ICMC-USP, Av. Trab. São Carlense, 400 - Centro., São Carlos, SP, Brazil, 13566-590; Domenico Amalfitano, domenico.amalfitano@unina.it, Department of Electrical Engineering and Information Technology, University of Naples Federico II, DIETI-UNINA, Via Claudio 21, Napoli, Campania, Italy, 80125; Lina Garcés, lina@unifei.edu.br, Instituto de Matemática e Computação, Universidade Federal de Itajubá, IMC-UNIFEI, Av. B P S, 1303 - Pinheirinho, Itajubá, MG, Brazil, 37500-903; Anna Rita Fasolino, fasolino@unina.it, Department of Electrical Engineering and Information Technology, University of Naples Federico II, DIETI-UNINA, Via Claudio 21, Napoli, Campania, Italy, 80125; Stevão A. Andrade, stevao@usp.br; Márcio Delamaro, delamaro@icmc.usp.br, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, ICMC-USP, Av. Trab. São Carlense, 400 - Centro., São Carlos, SP, Brazil, 13566-590.

0360-0300/2021/1-ART1 $15.00
https://doi.org/10.1145/3507903

Additional Key Words and Phrases: Software testing, dynamic testing techniques, mobile apps, non-functional requirements, systematic mapping

ACM Reference Format:
Misael C. Júnior, Domenico Amalfitano, Lina Garcés, Anna Rita Fasolino, Stevão A. Andrade, and Márcio Delamaro. 2021. Dynamic Testing Techniques of Non-Functional Requirements in Mobile Apps: A Systematic Mapping Study. *ACM Comput. Surv.* 1, 1, Article 1 (January 2021), 37 pages. https://doi.org/10.1145/3507903

1 INTRODUCTION

The mobile device market is constantly growing. In 2020, there were 3.5 billion smartphone users Worldwide [66]. Two of the most used mobile operating systems are: Google Android and Apple iOS, which have 84.8% and 15.2% of market users, respectively [17]. The diffusion of mobile devices is strictly related to an ever-increasing number of mobile apps that are of common use by helping users in various daily activities, from recreational and entertainment, such as playing Among Us, watching Netflix series, or chatting with Instagram friends, to the most critical ones such as paying for bills using bank accounts or controlling a home device. To have a better idea of the wide diffusion of mobile apps available for users, more than 13.8 million mobile apps are available on Google Play Store and the iOS App Store and more than 2.6 million publishers have distributed their apps on these online stores.

As more and more apps are being developed not only for entertainment purposes, but also to target safety and time critical domains, the quality of these apps has become crucial. Mobile apps need to be secure, as they commonly track our movements, follow our interests, and increasingly control our world, via IoT connected devices [73]. Usability is another key quality characteristic (q.c.) for a mobile app as users of an application, and their judgment, ultimately decide on its success or failure [30]. Mobile apps have to be efficient, be able to prevent excessive CPU, memory and energy consumption [5]. They should offer high performance in terms of throughput levels and response times. Another highly required q.c. of mobile apps is the capability to behave as expected across the combination of mobile devices and platforms their customers are using [87].

Testing is the most commonly used approach to assure the quality of software applications. Mobile app testing has attracted the interest of the software engineering community over the last decade and many methods, techniques and tools for testing both functional and non-functional requirements of mobile apps have been proposed in the literature. Non-functional requirement (NFR) testing stands out compared to functional testing, as for each specific q.c. different strategies need to be used for test generation, different techniques and infrastructures to execute tests and to assess their results, and different types of tools to support the testing process execution.

Several secondary studies in mobile app testing have recently tried to provide a systematic overview and analysis of the different contributions emerged in this wide field of research. This literature has mainly focused on specific concerns, such as automated testing techniques [4, 62], automated functional testing of mobile apps [48, 80], or testing of mobile context-aware applications [59]. Other secondary studies have investigated mobile app testing techniques in general [89]. Although the specific challenges and current solutions for NFR testing of mobile devices and applications are often mentioned in these reviews, little attention has been paid to providing an overview of the state of the art in this specific field, as shown in Section 3. At the best of our knowledge this is the first secondary study that is specific for non-functional requirements testing in the mobile apps context.

To fill this gap, we decided to perform a Systematic Mapping Study (SMS) that concentrates on the testing techniques that are specific to NFRs. The main goal of our SMS is to provide a broad discussion on the NFRs dynamic testing in mobile apps that focuses on addressed q.c. (s), adopted testing strategies, explored mobile platforms, and supporting tools.

We provide various discussions from 56 selected studies that clearly address NFRs testing in mobile apps. As a result, our SMS provides the following contributions:

1 an overview of the NFR testing techniques applied to mobile apps
2 a broad characterization of NFR testing techniques based on the addressed q.c. (s), the adopted testing approaches and strategies
3 a wide discussion on the explored mobile platforms and the tools supporting the NFR testing techniques
4 a discussion on the future research trends and opportunities in this field of research

It should be consistent with the content of the paper: This study is organized as follows. Section 2 introduces the theoretical background used in this mapping. Section 3 describes the related work and shows how our study contributes to the state of the art in mobile app testing. Section 4 shows in detail how this mapping was planned and executed. Section 5, 6, and 7 relate the results and discussions we obtained by answering, respectively, G1, G2, and G3. Section 8 provides our findings for each research goal and open research challenges that we identified. Section 9 discusses the main threats to validity of the study and the actions taken to mitigate them. Finally, the conclusions are drawn in Section 10.

2 BACKGROUND

In the following sections, the main concepts and term definitions we adopted in this systematic mapping are presented.

2.1 Description and Classification of Non-Functional Requirements

Differently from Functional Requirements (FRs) that specify what a software system has to do to solve stakeholders' needs, NFRs define the degree to which a product or system provides functions that meet these needs when it is used under specified conditions [68, 77]. NFRs define in detail constraints to the software system related to the organization, context, environment, government, market, and levels of quality (e.g., security, performance, or reliability) [68, 77]. An example of an NFR is the General Data Protection Regulations (GDPR) imposed by governments on any software system that operates in a specific region. Non-functional requirements have been addressed in international standards as part of the software and systems quality initiative, Adams [2]. Both the earlier ISO/IEC Standard 9126 (ISO/IEC 1991) [31] and its replacement ISO/IEC Std 25010 (ISO/IEC 2011) titled *"Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models"* [32], include NFRs, definitions, and how to measure them as part of a systems endeavor Adams [2]. The ISO/IEC/IEEE 29119-1 called *"Software and systems engineering-Software testing-Part 1: Concepts and definitions"* [33] strictly relates the NFRs to the quality characteristics of a software product. These software quality characteristics are the ones defined in the SQuaRE that proposes a model that categorizes software product quality properties into eight characteristics, namely, functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability. The ISO/IEC/IEEE 29119-1 classifies the requirements into two main categories, i.e. FRs and NFRs. FRs are aligned to the Functional Suitability quality characteristic outlined in ISO/IEC 25010, whereas the NFRs are linked to the remaining seven quality characteristics (a.k.a. quality attributes, q.a) outlined in SQuaRE. Henceforth, we will use the terms non-functional requirement and quality characteristic (q.c.) without distinctions in this paper. The SQuaRE [32] also defines for each q.c. a set of Sub-characteristics representing in more detail the different ways in which the quality of the software product can be assessed. In the following section, we report a brief description of the quality characteristics defined in SQuaRE [32]:

- The *Performance efficiency* q.c. represents the performance related to the number of resources exploited by the software when it runs under stated conditions
- The *Compatibility* q.c. describes the degree to which a software system can exchange information with other software products, software systems or software components. This characteristic also expresses the degree to which a software system performs its required functions, while sharing the same hardware or software environment with other software products, software systems or software components
- The *Usability* q.c. indicates the degree to which a software system can be used by specific users to achieve well defined goals with effectiveness, efficiency and satisfaction in a given context of use
- The *Reliability* q.c. reports the degree to which a software system executes given functions in specified conditions for a given time period
- The *Security* q.c. defines the degree to which a software system protects the access to information and data to persons or other software systems that have not the appropriate degree of data access according to their types and levels of authorization
- The *Maintainability* q.c. characterizes the degree of to which a software system can be modified by maintainers effectively and efficiently. Examples of modifications are corrections, improvements or adaptation of the software to changes in environment, requirements, and functional specifications. The installation of updates or upgrades is considered as a kind of maintenance
- The *Portability* q.c. expresses the degree of effectiveness and efficiency with which a software system can be transferred toward new hardware, software, operational or usage environment

2.2 Description and Classification of Test Techniques

Testing is a systematic process for revealing the presence of faults in software. This activity is of utmost importance for developing high-quality software systems by assessing its functional and non-functional requirements [22, 68, 77]. As defined in ISO/IEC/IEEE 29119-1-2013, the main purpose of test design techniques is to help testers find defects in test items as effectively and efficiently as possible. The ISO/IEC/IEEE 29119-1-2013 also classifies the testing techniques in static testing and dynamic testing. Static testing is typically exploited to identify apparent defects ("issues") in documentary test items or anomalies in source code. It includes various activities, such as static code analysis, cross document traceability analysis and reviews. On the other hand, dynamic testing aims to find defects by forcing failures of executable test items. The main goal of dynamic testing is to derive test cases that have to be executed on a running test item. Our interest lies only in dynamic testing techniques from now on in this paper and we will use the terms testing technique and dynamic testing technique without any distinctions. In practice, testers usually apply one or more test techniques to derive test cases and procedures with the main goal of achieving a given test completion criteria, typically described in terms of test coverage measures [27], and so, to detect as many failures as possible [14]. Figure 1 shows the two classifications we adopted in this paper to describe the testing techniques. Each testing technique can be characterized by the strategy and the approach it adopts.

According to Myers et al. [64] the testing techniques can be classified on the basis of the adopted testing strategy, i.e., *white box*, *black box*, or *hybrid*. The testing techniques, based on *white box* (also called *glass box*) strategies, derive the tests by exploiting information about how the software has been designed or coded. On the other hand, the testing techniques relying on *black box* strategies (also known as *data-driven* or *input/output-driven*) strategies, generating test cases by considering the input/output behavior of the software under testing. In this approach, test data are derived solely from the specifications of the program under testing, i.e., without taking advantage of the

knowledge of its internal structure. Testing techniques combining *white box* and *black box* strategies are defined as *hybrid*.

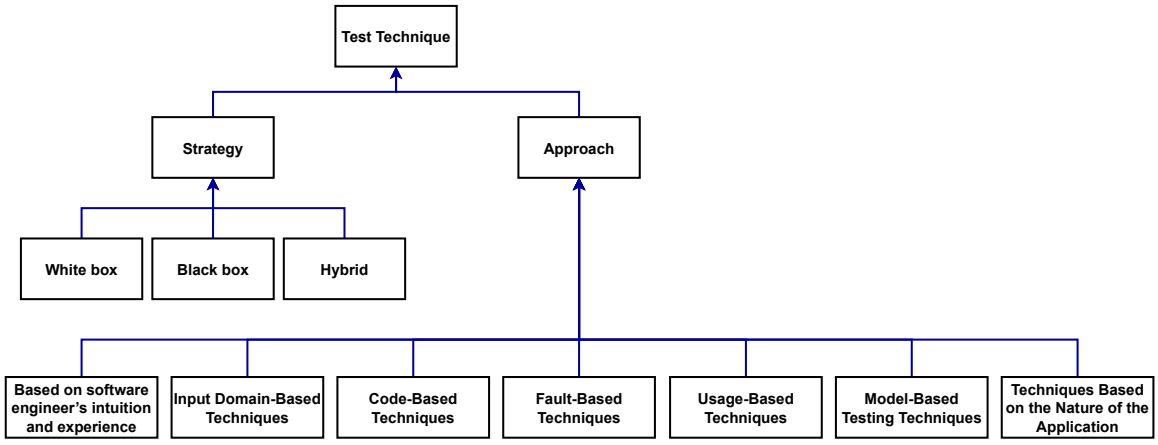

Fig. 1. Testing Techniques Classification, according to [14] and [64].

The other classification takes into account the testing approach used by the testing technique. The Software Engineering Body Of Knowledge (SWEBOK) [14] classifies the testing techniques in seven categories of approaches and a testing technique can belong to one or more approaches. Moreover, each category may be further specialized in sub categories of approaches. In the following section, we provide a brief description of the testing techniques and their sub categories as defined in [14]:

- The *Based on the Software Engineer's Intuition and Experience (SEIE)* category refers to approaches where tests are generated based on the skills and the knowledge of the software engineer involved in the test process. It comprises two sub-categories, i.e., *Ad Hoc* (*AH*) and *Exploratory Testing* (*ET*). The former refers to testing techniques where the tests are obtained only by relying on the expertise that the software engineer has about the software system under testing or in similar software systems. The latter indicates testing techniques where the tests are not preliminary defined in a test plan, but, rather they are dynamically designed, executed, or modified
- The *Input Domain Based (IDB)* category indicates approaches where tests are generated only by taking into account the properties of the inputs of the software system under testing. It is specialized in the following four sub-categories: *Equivalence Partitioning (EP)*, *Pairwise Testing (PT)*, *Boundary Value Analysis (BVA)*, and *Random Testing (RT)*. In Equivalence Partitioning, the input domain is preliminary partitioned into a collection of equivalency classes. Afterward, a set of tests is derived from each equivalency class. The Pairwise Testing refers to specific combinatorial approaches where the tests are derived by combining values of interest for every pair of a set of input variables. Boundary-Value Analysis denotes approaches where tests are derived by choosing values near the boundaries of the variables input domain. Random Testing represents approaches generating tests purely at random
- The *Code Based (CB)* category reports on testing approaches where tests are generated by taking into account the properties of code implementing the software system under testing. This category is specialized in two sub-categories, i.e., *Control Flow Based (CFB)*, and *Data Flow Based (DFB)*. The former refers to approaches generating tests aimed at covering all statements, blocks of statements, or specified combinations of statements in the source code of the software system under testing. The latter represents approaches that produce tests

aiming at execution for each variable in each segment of the control flow path, from the definition of that variable until its use

- The *Fault-Based (FB)* category denotes testing approaches where tests are generated for revealing specific types of faults, usually defined through a *fault model*. It is further specialized in two sub-categories that are *Error Guessing (EG)* and *Mutation Testing (MT)*. In Error Guessing, the tests are designed relying on the history of the faults in previous projects or based on the software engineer's expertise in anticipating the most plausible faults in the software system under testing. Mutation Testing refers to testing techniques where tests are executed on both the original program and on all its mutants, i.e., modified versions of the original program where changes, resembling common mistakes made by programmers, have been introduced
- The *Usage-Based (UB)* category characterizes to testing approaches aimed to generate tests that resemble as much as possible the behavior of the human user of the software system under testing. It comprises two sub-categories, i.e., *Operational Profile (OP)*, and *User Observation Heuristics (UOH)*. In the Operational Profile, the tests are derived by taking into account the operational profiles of the users. These approaches usually need a test environment reproducing as closely as possible the operational environment and the user profiles for emulating specific usages and criticality of the functions provided by the system under testing. The User Observation Heuristics class refers to testing approaches where tests are derived by taking into account usability principles
- The *Model-Based (MB)* category describes testing approaches exploiting models to derive the tests. A model is an abstract, formal in some cases, representative of the software system under testing. Model-based approaches can be further specified based on the models used to represent the system under testing. Hence, model-based testing approaches include the following four sub-categories: *Decision Tables (DT)*, *Finite-State Machines (FSM)*, *Formal Specifications (FS)*, and *Workflow Models (WM)*
- The *Based on the Nature of the Application (BNA)* category specifies testing techniques where the tests are designed and execution is based only on the specific nature of the software system under testing, such as object-oriented software, component-based software, concurrent programs, etc

2.3 Software Testing Tools

Testing processes require the executing a large number of labor-intensive, time-consuming and error-prone activities. To support software engineers in executing these tasks usually sophisticated tools are adopted [14]. These tools are distributed according to different licenses. *Free and Open-Source Software (FOSS)* tools are licensed to be freely used, copied, and changed by anyone. Moreover, the source code is available. *Proprietary software* tools are usually paid and are distributed under restrictive copyright licensing. The source code is usually unavailable to users. *Freeware* tools can be freely used by anyone, but the source code is not available.

We adopted the classification proposed by Bourque et al. [14] to characterize the testing tools based on the functionality they provide to the practitioners. Based on this classification, a testing tool can belong to one or more of the following categories:

- The *Test harnesses (drivers, stubs) (TH)* category refers to tools providing a complex controlled environment, made by driver and stubs, where tests can be generated and executed on the fly, or stored, planned, and executed. These tools are also able to log the results of the test executions

- The *Test generators (TG)* category indicates tools providing complex features for automatic test case generation
- The *Capture/replay (CR)* category describes tools able to record the interactions the user has with the application under test (usually with the user interface). Capture/replay tools also store the recorded interactions in a common shared repository and enables them all to be replayed or a selected subset of them
- The *Oracle/file comparators and assertion checking (OC)* category refers to tools that provide features for defining and executing complex oracles automatically. Oracles are used to check the presence of failure in the tested application, at run-time
- The *Coverage analyzers and instrumenters (CAI)* category specifies tools that are able to generate test cases by source code analysis. In this category, tools able to insert probes in the source code and the tools performing source code analysis and reverse engineering are also included
- The *Tracers (T)* category characterizes tools providing features for logging, in text files, execution data, such as: execution paths, energy consumption, memory occupancy, network traffic, etc
- The *Regression testing (RT)* category denotes tools that, in response to a code change, are able to select and execute automatically only the test cases needed to test the modified code
- The *Reliability evaluation (RE)* category expresses tools that provide rich graphical user interfaces to support the test engineers to perform complex statistical analysis of reliability-related measures

3 RELATED WORK

One of the first surveys in the research area of software testing for mobile applications was reported in 2012 by Janicki et al. [37]. The authors conducted a semi-supervised and group-administered survey with 49 mobile software engineers. They identified the expectations and challenges related to automatic test generation and execution, some obstacles of introducing automatic test tools in industry, and metrics and reports to help convince companies to try the technology.

In 2015, Janicki's work was preceded by three other literature reviews [62, 71, 78]. Starov et al. [78] provides an introduction to the main challenges in mobile app testing. They discussed some issues for cloud testing and investigated cloud services and testing-as-a-service resources that could improve mobile app testing, covering different types of mobile testing features. Furthermore, Sahinoglu et al. [71] focused on mobile app testing, relating it with test levels (e.g., system, acceptance, unit, component, integration) and q.c. (s) (i.e., compatibility, concurrent, conformance, performance, security, usability), and presented how different issues for validating those apps are addressed. Méndez Porras et al. [62] surveyed 83 primary studies. The authors provided an overview on automated testing approaches for mobile applications, testing techniques, and empirical assessment. They also investigated the major challenges in automated testing of mobile applications. As a result, they identified the following as main approaches, model-based testing, capture/replay, model-learning testing, systematic testing, fuzz testing, random testing and scripted based testing. By 2015, they observed an increase in proposals for automated mobile app testing.

One year later, Zein et al. [89] published a systematic mapping study that categorized and structured evidence obtained from 79 primary studies. The authors identified important needs while developing mobile software, such as eliciting test requirements early on in the development process, conducting research in real-world development environments, specifying testing techniques aimed at lifecycle compliance of applications and mobile service testing, as well as performing benchmarking studies for security and usability testing. In 2018, Kaur and Kaur [40] was the first secondary study aiming to identify and understand techniques to estimate testing effort in mobile applications.

Additionally, they summarize some characteristics that make mobile software/applications different from traditional software (e.g., desktop or web).

Secondary studies in mobile app testing have been predominant over the last two years [3, 4, 48, 59, 80, 86]. Luo et al. [59] aimed to give an overview of the simulation methods, i.e., data-driven and model-based, for testing mobile context-aware applications. This study also discusses how each method can be implemented and deployed by testing tool developers and mobile application testers. The systematic mapping conducted by Almeida et al. [4] identified and discussed the state of the art on tools that allow the automation of testing Android context-aware applications. The authors surveyed 68 primary studies and identified which of the 80 tools found are oriented to test context-aware characteristics in Android context-aware apps. As a result, they reported a lack of tools for these apps supporting neither the automatic generation or execution of test cases in high-level contexts, nor the asynchronous context variations. For the cloud domain, the mapping study executed by Ya'u et al. [86] surveyed 23 primary studies to identify how testing is conducted in mobile applications in this domain. The authors found diverse approaches to test GUI, compatibility, functionality, and security characteristics. However, few of these approaches are supported by automated testing tools. They also found that approaches lack portability among mobile platforms, as well as the lack of exploring TaaS (testing as a service).

Two recent studies [48, 80] investigated automatic testing for mobile applications. Tramontana et al. [80] analyzed 131 primary studies. This study investigated automatic testing tools based on the supported testing activities, the characteristics of the techniques they present, and the evaluation methodologies adopted to validate them. Kong et al. [48] offered an overview on existing Android testing approaches. They analyzed 103 primary studies. Their investigation was focused on identifying testing approaches, the test concerns and levels addressed by such approaches, and how they have been built and validated. Based on their findings, the authors proposed a taxonomy of android app testing, considering test objectives (e.g., non-functional requirements), test targets (i.e., parts of the App to be tested), test levels (e.g., integration, system, unit), and test techniques, such as methods (e.g., model-based, search-based), testing environments (e.g., emulator, real device), and testing types (e.g., white, black, or gray boxes).

Our systematic mapping contributes to the state of the art in the following ways: (i) offering a more complete overview than related work; (ii) focusing on existing testing techniques that deal with a broader set of q.c. (s), e.g., performance, security, usability, portability, reliability, and compatibility, and other q.c.; (iii) considering different mobile platforms, i.e., Android, iOS, Blackberry, LG, and Windows Phone; (iv) analyzing the involvement of the academic community and industry in this important topic; (v) characterizing the techniques and tools for NFRs testing in mobile applications; and (vi) identifying future trends.

4 SYSTEMATIC MAPPING PROCESS

The SMS was motivated by the lack and necessity of having an overview on the state of the art on platform-independent dynamic testing techniques for mobile apps focused on NFRs (and not only FRs).

The mapping study was designed following the guidelines proposed by Petersen et al. [67]. The protocol was defined through extensive discussions between all researchers involved in this study. In the remaining of this section, the SMS protocol is presented, detailing goals, research questions, search strategy, inclusion and exclusion criteria, strategies for data extraction and data synthesis, and the selection process.

4.1 Goals, Research Questions, and Metrics

s

The main goal of this SMS is to identify, evaluate and collect available and relevant research about dynamic NFR testing techniques on mobile applications. To design our research, we used the Goal–Question–Metric (GQM) approach [16]. By doing so, we refined the main goal into three research goals as listed in Table 1. For each goal we also defined a set of RQs and the rationale indicating what is expected from the answer (Table 2). The RQs were specified in a generic form to obtain, over time, topics already explored and research trends to be investigated, as suggested in Petersen et al. [67]. Table 3 presents the metrics planned to answer each RQ and the related data to be extracted from the primary studies.

Table 1. Goals of the Systematic Mapping.

Goal ID	Goal	Rationale
G1	To characterize the studies proposing techniques for testing NFRs in mobile apps	This provided to characterize the research works in the area of NFR testing techniques for mobile apps.
G2	To describe the techniques proposed in the literature for testing NFR in mobile apps.	This intended to characterize the NFR testing techniques for mobile apps.
G3	To analyze the tool support of NFR testing techniques for mobile apps.	The aim is to characterize the tools proposed in the literature to support the execution of NFR testing techniques for mobile apps.

Table 2. Proposed Research Questions.

Goal ID	RQ ID	Research Question	By answering this RQ, the researcher will find out
G1	$RQ1_1$	What is the articles count per year?	what efforts to consolidate this research area have been invested by the research community over the years.
	$RQ1_2$	What is the article count by venue type?	the venues in which relevant studies for this research area have been published.
	$RQ1_3$	What is the article count by collaboration type?	the collaboration type between academia and industry in proposing techniques to test NFRs for mobile apps.
	$RQ1_4$	Which are the most influential articles?	the most relevant publications in this research area.
	$RQ1_5$	Which is the article count by country?	which countries are leading this research area.
G2	$RQ2_1$	Which are the quality characteristics addressed for testing NFRs in mobile apps?	the quality characteristics that have received more interest (or are more critical) to be tested in mobile apps.
	$RQ2_2$	Which are the mobile platforms tackled by NFR testing?	the tendency of proposing testing techniques based on specific mobile platforms.
	$RQ2_3$	Which are the testing strategies adopted for NFRs testing?	the tendency of using specific strategies to test NFRs for mobile apps.
	$RQ2_4$	Which are the testing approaches used to test NFRs in mobile apps?	the direction of the testing approaches adopted to test NFRs in mobile apps.
G3	$RQ3_1$	How many of the identified NFRs testing techniques are tool supported?	which percentage of the existing NFRs testing techniques for mobile apps are tool supported.
	$RQ3_2$	What are the adoption levels (in %) of "Proprietary", "Freeware", "Free and Open" tools?	the software licenses of the tools used to support the execution of NFR testing in mobile apps[1].
	$RQ3_3$	Which are the tool types used to support the execution of NFR testing techniques for mobile apps?	the tendencies of the tool types used to support the execution of NFRs testing techniques for mobile apps.

4.2 Search Strategy

To perform a systematic selection of relevant primary studies for our research, we specified a search strategy by considering the guidelines proposed by Kitchenham and Charters [45] as described in the following section.

4.2.1 Construction of the Search String. To systematically identify the terms of the search string, we adopted the PICOC (Population, Intervention, Comparison, Outcome, and Context) criteria as suggested in [67], and described as follows:

- *Population* refers to the terms related to the main object being investigated, i.e., mobile applications. We observed that a wide number of papers are focused on apps implemented for a specific mobile operating system such as Android or iOS
- *Intervention* indicates the terms that are related to approaches or tools involving the population

Table 3. Proposed Metrics.

RQ ID	Metric	Extracted Data
$RQ1_1$	Count the number of papers grouped by year	The publication year
$RQ1_2$	Count the number of papers grouped by Venue	The publication venue ∈ {Workshop, Symposium, Conference, Journal}
$RQ1_3$	Count the number of papers grouped by Collaboration Type.	The collaboration type ∈ {Academic, Industrial, Crosscutting}
$RQ1_4$	Report the first 10 papers having more citations.	The citations count
$RQ1_5$	Count the number of papers grouped by country	The first author affiliation country
$RQ2_1$	Count the number of papers grouped by quality characteristics	The quality characteristics under test.
$RQ2_2$	Count the number of papers grouped by mobile platform.	The tackled mobile platforms.
$RQ2_3$	Count the number of papers grouped by testing approach	The adopted testing approaches.
$RQ2_4$	Count the number of papers grouped by testing strategy	The adopted testing strategy
$RQ3_1$	Count the number of papers grouped by "tool supported technique" and "no tool supported technique"	The testing technique is tool supported or not
$RQ3_2$	Count the number of tools grouped by license	For each tool extracted to answer $RQ3_1$, extract its license
$RQ3_3$	Count the number of tools grouped by type	For each tool extracted to answer $RQ3_1$, derive its type

- *Comparison* refers to the software engineering methodology/tool/technology/procedure with which the intervention should be compared. A comparison has not been made in our research as it is a more adequate for systematic literature reviews
- *Outcome* relates to factors of importance to practitioners' concerns, i.e., the q.c. presented in Section 2. Preliminary initial searches showed that performance efficiency testing techniques have been published in the literature as *energy consumption* or *energy bug detection* testing approaches. This justifies the additional term, i.e., Energy, we added to the q.c. proposed by the ISO/IEC 25010 International Standard
- *Context* refers to terms related to the testing approaches involving the considered population

The RQs were broken down into facets and a list of synonyms, abbreviations, and alternative spellings for each term of the PICOC was defined by Kitchenham and Charters [45]. Additional terms were obtained by considering subject headings used in journals and scientific data bases. The main terms and the synonyms we inferred for the PICOC components are shown in Table 4.

Table 4. PICOC main terms and their synonyms.

View point	Main terms	Synonym
Population	Mobile App	Mobile software, Mobile system, Android, iOS
Intervention	Approach	Technique, Method, Strategy, Activity, Methodology, Tool, Framework
Comparison	N.A.	N.A.
Outcome	Non Functional Requirement	Quality Attribute, Performance, Energy, Compatibility, Usability, Reliability, Security, Maintainability, Portability
Context	Test	Testing, Verification, Validation

Moreover, the search string was validated by considering a test set of seven control studies [21, 34, 46, 50, 72, 75, 91] previously identified by one of the authors. The test set was defined before we started to define the search string. Therefore, the quality of the search string was evaluated by verifying whether the control studies were returned by applying the string to the Scopus search engine. The final search string we defined is the following.

Search String:
("Mobile app" **OR** "Mobile software" **OR** "Mobile system" **OR** "Android" **OR** "iOS") **AND** ("Approach" **OR** "Technique" **OR** "Method" **OR** "Strategy" **OR** "Activity" **OR** "Methodology" **OR** "Tool" **OR** "Framework") **AND** ("Non Functional Requirement" **OR** "Quality Attribute" **OR** "Performance" **OR** "Energy" **OR** "Compatibility" **OR** "Usability" **OR** "Reliability" **OR** "Security" **OR** "Maintainability" **OR** "Portability") **AND** ("Test" **OR** "Testing" **OR** "Validation" **OR** "Verification")

4.2.2 Selection of the Information Sources. Table 5 shows the four scientific databases (selected as our search scope) to retrieve potentially relevant studies in computer science, as defined by Dyba et al. [23]. As each database has its own search interface, we needed to adapt the search string. Adaptations are fully described in Junior et al. [39].

Table 5. List of Electronic Databases.

Search Engine	Link
ACM Digital Library	http://dl.acm.org/
IEEE Xplore	http://ieeexplore.ieee.org/
Web of Science	https://www.webofknowledge.com/
Scopus	http://www.scopus.com/

4.3 Inclusion and Exclusion Criteria

In this section, we describe the exclusion and inclusion criteria used to filter the primary studies retrieved from the selected electronic databases. To be excluded from our SMS, a paper has to satisfy at least one of the twelve Exclusion Criteria (EC) listed below:

- **(EC-1):** studies that are duplicates
- **(EC-2):** studies that are not written in English
- **(EC-3);** studies reporting a summary, a conference call, a patent, or lecture notes
- **(EC-4):** studies that are in an initial stage, typically presenting an abstract, a summary of future steps, poster, panel, or conference short paper (i.e., paper having less than 6 pages)
- **(EC-5):** studies that cannot be downloaded
- **(EC-6):** studies that do not belong to the Computer Science research domain, such as, biology, finance, and digital forensics
- **(EC-7):** studies that are a preliminary or reduced version of a more extended work published by the same main author
- **(EC-8):** studies that do not clearly propose and describe a dynamic testing technique, according to the definition we pointed out in Section 2.2
- **(EC-9):** studies that do not present an evaluation of the proposed testing technique, such as: case study, controlled experiment, proof of concept, empirical study, etc
- **(EC-10):** studies that present a comparison of testing techniques proposed by other authors;
- **(EC-11):** studies that do not explicitly point out the q.c. or sub-characteristics under testing
- **(EC-12):** studies that do not have mobile apps as objects of their research

Four Inclusion Criteria (IC) were used to add studies to our sample. To be accepted in our SMS, the study must accomplish all the following IC:

- **(IC-1):** the study clearly describes and details the proposed testing technique for finding defects related to NFR in mobile apps
- **(IC-2):** the study clearly describes and details the methods performed to evaluate the proposed testing technique
- **(IC-3):** the study explicitly points out the q.c. (s) or sub-characteristic(s) under testing
- **(IC-4):** the study focuses on mobile app testing

4.4 Quality Assessment

For the aim of our mapping study, the quality assessment stage was not executed to avoid imposing high requirements on the primary studies, allowing us to obtain a broad overview of the topic area [44, 67].

4.5 Selecting primary studies

To select primary studies, we planned and followed the two-stage process shown in Figure 2, which is detailed as follows.

4.5.1 First Stage. It was executed to retrieve an initial set of primary studies relevant to our mapping. This stage was conducted from March 2020 to July 2020 by four authors who accumulated experience on mobile testing, NFR testing, and empirical studies. As shown in the Figure 2, this stage relies on the execution of the following five consecutive activities:

(1) *Database searches:* in this step, one researcher adapted and executed the search string in the four electronic databases. The search was filtered by title, abstract, and keywords. No time limits were considered in the search. As a result of this step, we retrieved a total of 8182 primary studies.
(2) *Removal of duplicates:* One author aggregated all the retrieved studies in a unique bibtex file. We used the JabRef tool to identify and remove duplicated studies. As a result, 2487 studies were excluded and 5695 studies proceeded to the first selection step.
(3) *First selection (title,abstract and keywords):* The title, abstract, keywords and, when necessary, conclusion sections of all primary studies were read by at least two researchers. They applied the inclusion and exclusion criteria for each study, excluding 4922 studies. Moreover, 773 studies, which included some of our criteria labeled as a "doubt" study, were read in full in the second selection step.
(4) *Second selection (full text):* Two researchers read the full text of all 773 primary studies applying again the inclusion and exclusion criteria. As a result, 517 studies were excluded, 35 studies were included. 221 studies were labeled as a "Doubt" study as no consensus between both researchers was achieved for those studies.
(5) *Solving "doubt" primary studies:* The 221 "doubt" studies were analyzed by a third senior researcher, who decided to include 10 studies and exclude 211 studies.

As a result of the first stage, a total of 45 primary studies were included.

4.5.2 Second Stage. This refers to a snowballing process that was conducted in April 2021 and, therefore, most of the analyzed studies were published in 2020. As shown in Figure 2 this stage relies on the execution of five consecutive activities, four of them are actually the same activities followed in the first stage:

(1) *Primary studies retrieval by backward and forward snowballing:* in this automatic step, 1903 primary studies were retrieved by applying backward and forward snowballing to the 45 papers selected in the first stage. More specifically, for each one of the 45 papers, in backward snowballing, all the studies it referenced were gathered, whereas, in forward snowballing, we collected from Google Scholar all the works citing it
(2) *Removal of duplicates:* duplicates primary studies were automatically removed from the set of 1903 papers obtained in the previous step and 1650 documents were filtered out
(3) *First selection (title,abstract and keywords):* 423 papers from the 1650 were included for full reading in the second selection. The remaining 1227 studies were excluded by the reviewers
(4) *Second selection (full text):* among the 423 primary studies included in the previous step, 401 studies were labeled as "No" and excluded. 9 primary studies were labeled as "Yes" and added to the final set. The remaining 13 papers were labeled as "doubt" and transferred to the following step
(5) *Solving "doubt" primary studies:* among the 13 doubt papers, 3 were added to the set of the 12 papers selected by snowballing

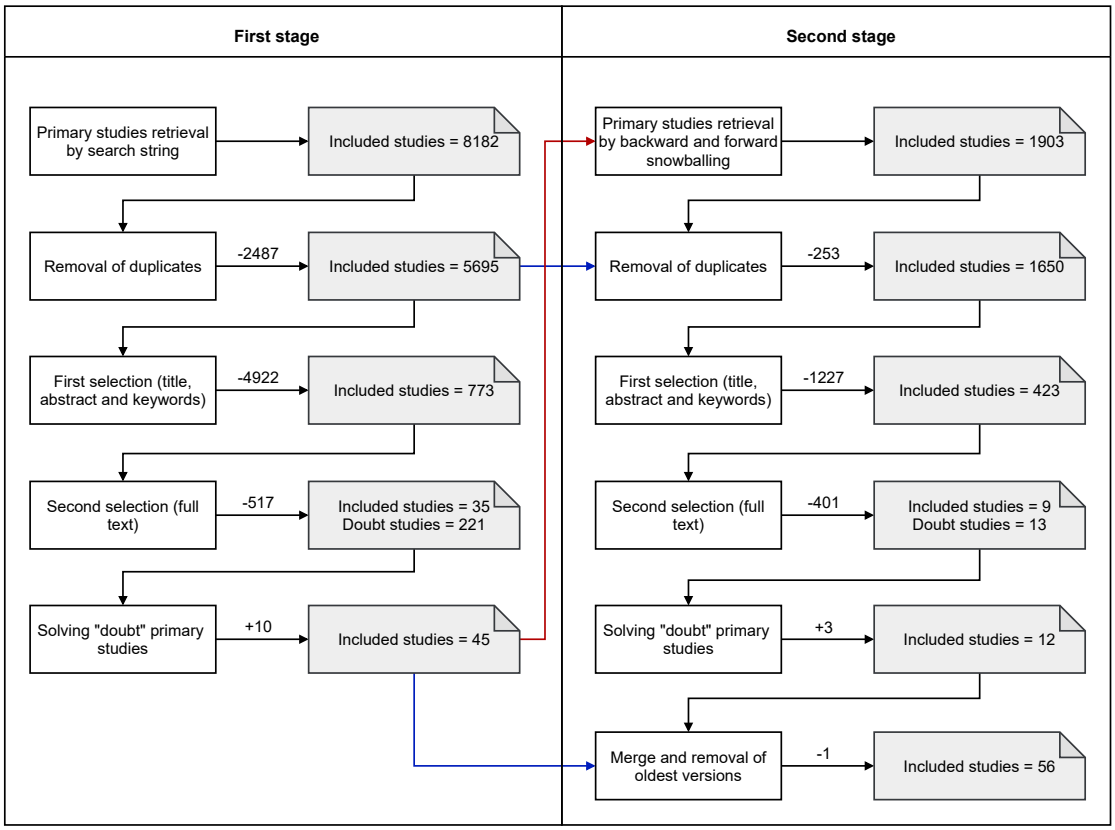

Fig. 2. Primary studies selection process.

(6) *Merge and removal of oldest versions:* the 45 papers retrieved in the first stage were merged to the 12 studies obtained by snowballing to build the final set of 56. We removed one of the studies found in the first stage as its updated and extended version was selected in the second stage through snowballing. Table 12, in Appendix A, shows the list of the 56 selected papers

4.5.3 Details on First and Second selection steps execution. In both steps, the IC and EC were applied following the process shown in Figure 3. The selection process was executed by two reviewers who worked independently. Each reviewer labeled each primary study as:

- "Yes", if the reviewer considered the paper to be included according to all IC
- "No", if the reviewer considered the paper to be excluded according to any EC
- "Doubt", if the reviewer was not able to include or exclude the paper

After, the results were merged. Studies labeled as "Yes" by both reviewers were added to the set of Included primary studies. Papers labeled as "No" by both reviewers were considered as Excluded. All the remaining papers were considered as "Doubt". In the First selection step both Included and Doubt papers passed to the Second step to be analyzed by full text reading. At the end of the Second selection step Included and Doubt primary studies were not merged together. Doubt papers were discussed and resolved by the two reviewers and a third one.

4.6 Data extraction

To extract data from the identified primary studies, we developed the extraction form shown in Table 6. Each data extraction field has a data item and one or more values depending on the research question it is related to. The extraction was performed by the first author and reviewed by the

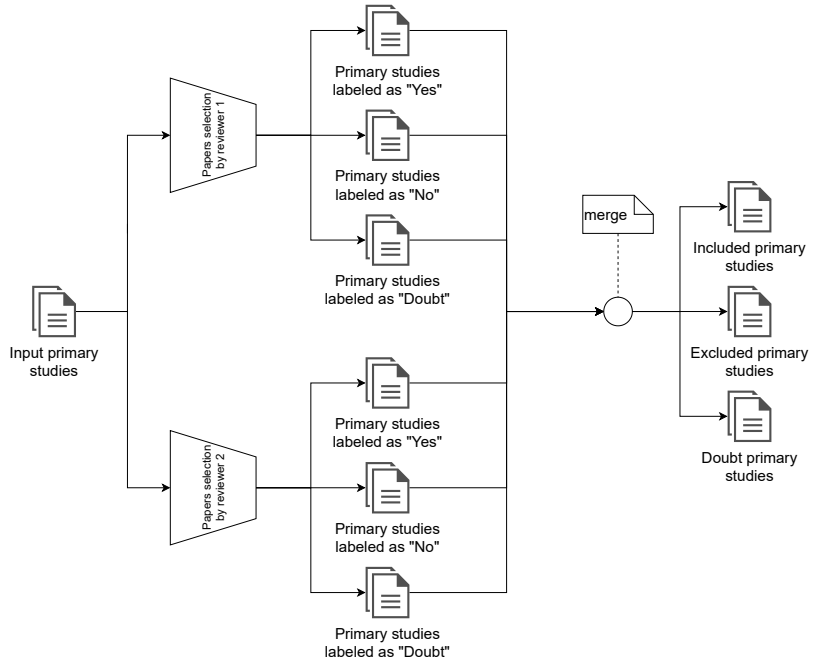

Fig. 3. First and Second selection processes applying Inclusion and Exclusion Criteria.

second one who tracked back the information in the extraction form to the statements in each paper and checked their correctness. Having another author check the extraction is common practice to reduce the bias as suggested by Petersen et al. [67]. To facilitate this step, we created an on-line free google form[2] to guide the researchers in collecting evidence from primary studies about (1) publication details (e.g., authors, year, title, source, abstract) and (2) context descriptions (e.g., NFRs, technologies, involved industry, empirical settings).

Table 6. Data Extraction Form.

ID	Field	Description	RQ
		Publication details	
1	Title	Title of the primary study	–
2	Abstract	Abstract of the primary study	–
3	Keywords	Keywords of the primary study	–
4	Authors	Authors of the primary study	–
5	Source	Name of the search engine source where the primary study was returned	–
6	Year	Publication year of the primary study	RQ1
7	Venue	Name of publication where the primary study was published	RQ1
8	Country	Country of affiliation of the first author of the study	RQ1
9	Citation	Number of citations of the primary study	RQ1
10	First Author	Name of the first author of the primary study	RQ1
11	Collaboration	Collaboration type (e.g., Academy, Industry, and Crosscutting) of the primary study	RQ1
12	Company	Company name involved in the primary study	RQ1
		Context Descriptions	
13	Software Quality Standard	Software Quality Standards (9126, 25010) referenced by the primary study	RQ2
14	Quality attribute	List of quality attributes analyzed in the study defined according ISO/IEC 25010	RQ2
15	Reasons to address the quality attribute	Reasons raised by the authors to address the quality attribute.	RQ2
16	Technique description	A brief description of Non-functional testing approach proposed in the study	RQ2
17	Mobile platform	List of mobile platforms cited in the study (e.g., Android, iOS, etc.)	RQ2
18	Testing strategy	Testing strategy reported in the study according to the SWEBOK	RQ2
19	Testing approach	List of testing approaches reported in the study from according to the SWEBOK	RQ2
20	Tool name	Name of the used tool	RQ2
21	Tool URL	URL to download the tool	RQ2
22	Tool License	Licenses of the supporting tools, i.e., FOSS, Freeware, or Proprietary	RQ3
23	Tool Type	List of tool types from those proposed in the SWEBOK	RQ3
		Additional Information	
24	BibTeX reference	Unique reference for the primary study	–
25	Researcher(s)	Researcher(s) who extracted the data from the study	–

[2] https://docs.google.com/spreadsheets/d/1dAEnYkfzgXlpFI8TuYPp8tjsYVu3aAkrJBU_ppOXkHA/edit?usp=sharing

5 G1: PRIMARY STUDIES ON NFR TESTING OF MOBILE APPS

In this section, we discuss the distribution of primary studies on NFR testing over time (Subsection 5.1), the distribution of the primary studies by the publication venue type (Subsection 5.2), the distribution of companies involved in NFR testing of mobile apps (Subsection 5.3), the list of the most cited studies in the addressed research topic (Subsection 5.4), and the list of the countries leading in this research topic (Subsection 5.5).

5.1 $RQ1_1$ Studies over time

Figure 4 shows the distribution of publication years of the 56 primary studies we selected. The list of selected works is reported in Table 12 showing the ID, title, reference, year of publication, publication venue, and the NFRs for each study addressed in the study. Figure 4 shows the growing interest of the community in this research area starting from 2009 when the first primary study was published. Moreover, we can observe that the topic is quite new with approximately 13 years of contributions, whereby the 58.9%(33/56) of primary studies were published during the last four years, as shown in Table 7. The growing interest of the community matches the raising number of smartphones sold from 2007[3].

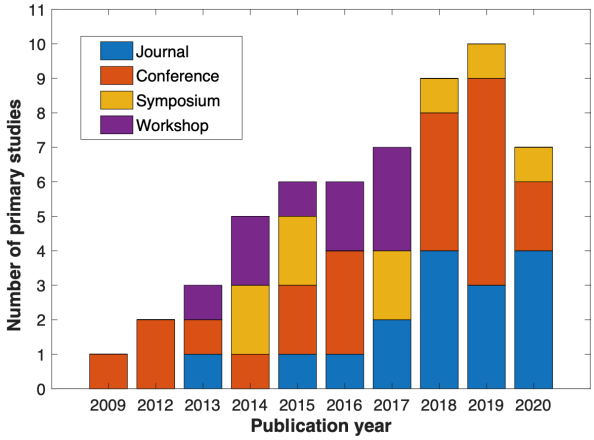

Fig. 4. Distribution of publications by year.

Table 7. Number of studies grouped by 4 years.

4-year interval	#
2009 – 2012	3
2013 – 2016	20
2017 – 2020	33
Total	56

5.2 $RQ1_2$ Publication venues

Figure 4 also renders the distribution of the primary studies by the publication venue type. As the figure shows, 44.7% (25/56) of the studies were published in conferences, 28.6% (16/56) in journals, 16.0% (9/56) in symposiums, and 10.7% (6/56) in workshops. We can observe a growing trend of works published in journals, showing that the quality of the works has reached a sufficient quality. These works were published in a wide range of computer science venues. Moreover, the venues that published more studies are the top ones, with very high rank and impact factor, i.e., IEEE Access among the journals, ICSE for the conferences, ISSTA for the symposiums, and AST among the workshops. Table 15 shows the full list of venues where selected papers have been published.

5.3 $RQ1_3$ Industry-Academy collaboration

We found that 85.7% (48/56) of the primary studies were fully developed in academia, 3.6% (2/56) were fully conducted in industry, and 10.7% (6/56) resulted from collaborations between the academic community and industry. Table 8 reports the eight primary studies with industry involvement,

[3]https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/

listing the companies and its respective countries. The results show that there is not either a leading country or an industry that has been more involved than others. Although we found a low percentage of studies (i.e., 14.3%) reporting companies involved in NFRs testing for mobile apps, it seems that this gap started to be slightly filled in the last 6 years by studies involving companies.

Table 8. Companies involved in NFR testing of mobile apps.

ID	Year	Company	Collaboration	Country
S31 [9]	2013	Fondazione Bruno Kessler	Industry	Italy
S23 [91]	2015	Fujitsu Laboratories	Crosscutting	China
S25 [29]	2015	IBM	Industry	Israel
S55 [26]	2015	Nokia Technology Institute	Crosscutting	Brazil
S34 [21]	2017	Google Inc	Crosscutting	United States
S37 [1]	2018	Technologie Sanstream	Crosscutting	Canada
S1 [50]	2019	Red Ant Technology	Crosscutting	Malaysia
S47 [84]	2020	National Computer Network Emergency Response Technical Team/Coordination Center of China (CNCERT/CC)	Crosscutting	China

5.4 $RQ1_4$ Most cited studies

We identified the most relevant studies in the addressed research topic. Google Scholar[4] was used to evaluate the number of citations for each primary study. Table 9 shows the 10 most cited papers ranked based on the number of the citations evaluated in July 2021.

Table 9. Top 10 papers ranked on Google Scholar citations.

ID	Year	Venue type	Venue	Number of citations
S4 [70]	2013	Conference	Conference on Data and Application Security and Privacy	442
S53 [85]	2014	Symposium	Symposium on Information, Computer and Communications Security	86
S35 [52]	2012	Conference	Conference on Advances in Mobile Computing & Multimedia	74
S25 [29]	2015	Symposium	International Symposium on Software Testing and Analysis	71
S42 [43]	2009	Conference	Conference on Secure Software Integration and Reliability Improvement	55
S24 [83]	2015	Conference	International Conference on Software Testing, Verification and Validation	51
S15 [35]	2015	Symposium	Symposium on the Foundations of Software Engineering	50
S41 [19]	2016	Conference	Working Conference on Mining Software Repositories	46
S38 [60]	2013	Journal	Mobile Networks and Applications	42
S26 [10]	2017	Journal	IEEE Transactions on Software Engineering	42

5.5 $RQ1_5$ Contributions at Worldwide

Figure 5 reports a map chart showing how the main researchers on NFR testing techniques for mobile apps are spread all over the world. A country is colored in blue if there is at least one study having the main author affiliated with either a university or a company placed there. Table 10 shows the countries that have more than one study. As the data show, China, United States, and Canada have led the research area with 21.4%(12/56), 19.6%(11/56), and 7.1%(4/56) of contributions, respectively.

[4] https://scholar.google.com

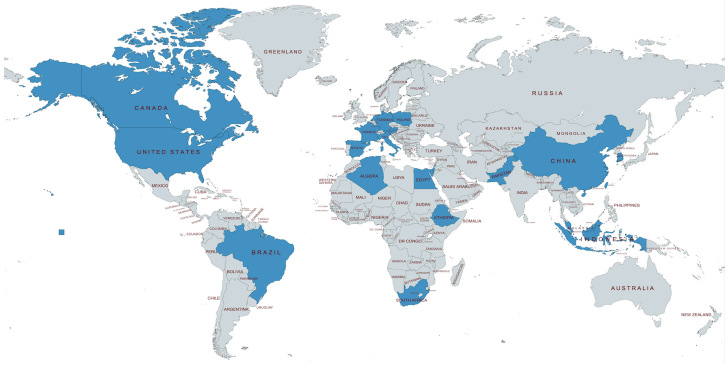

Fig. 5. Map chart of the countries.

Table 10. Number of publications by country.

Country	#
China	12
USA	11
Canada	4
Brazil	4
Singapore	3
Italy	3
Republic of Korea	3
Germany	3
Malaysia	2
Others	11

6 G2: NFR TESTING TECHNIQUES FOR MOBILE APPS

In this section, we discuss the distribution of primary studies per q.c. and per year (Subsection 6.1), the types of mobile platform considered in the studies (Subsection 6.2), the NFR testing strategies (Subsection 6.3) and the NFR testing approaches (Subsection 6.4) that have been proposed per quality characteristic.

6.1 $RQ2_1$ Distribution of quality characteristics addressed by primary studies

Figures 6 and 7 show the set of q.c. found in primary studies, namely, performance, security, usability, portability, reliability, and compatibility, and their distribution over the years.

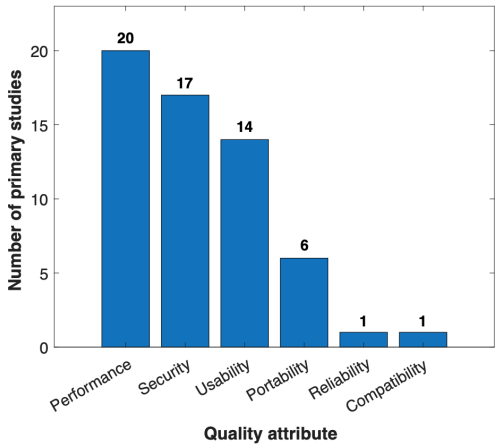

Fig. 6. Quality attributes reported in the primary studies.

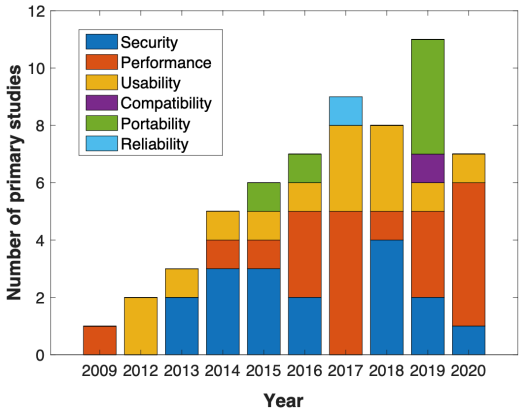

Fig. 7. Quality attributes reported in the primary studies by year.

Distribution of q.c. (s): From Figure 6, we observe that Performance (35.7%, 20/56), Security (30.3%, 17/56), and Usability (25.0%, 14/56) are the most addressed q.c. (s), followed by Portability (10.7%, 6/56). Finally, Compatibility and Reliability q.c. (s) were addressed in 1.8% (1/56) of the selected studies.

We performed an extra analysis of considered primary studies to understand the motivations that led the researchers to address the testing of specific q.c.. Table 11 shows, for each q.c., the reported reasons and the references to the primary studies where they were declared. Related to the most common motivations, we note that for performance efficiency q.c., *there is a need to test if the apps negatively impact battery/power consumption.* Furthermore, *there is a lack of support tools*

Table 11. Motivations for addressing a specific NFR testing.

NFR	Motivation	References
PE	The need to test how apps impact battery/power consumption	[34], [35], [65], [83], [10], [1], [92], [19], [43], [76], [53]
	Performance bugs are very difficult to be detected and reproduced	[57], [24], [51]
	Performance testing is a very time-consuming task	[43]
	Lack of tools or methodologies for performance testing	[34], [35], [83], [1], [36]
	The need to detect memory leaks	[6]
	Poor performance apps negatively impact the user experience	[6], [51], [90]
	App performance may vary different mobile platforms	[81], [76]
	Developers do not have proper knowledge about performance, therefore they are not careful with this q.c	[81], [6], [21], [19]
S	Developers do not have proper knowledge about security, therefore they are not careful with this q.c	[7], [70], [75], [28], [58], [12], [72], [88], [29], [41], [47], [9], [82], [84], [54], [74], [85]
	Security testing is not a simple task and it is worth being investigated	[9], [82], [54]
	Programmers developing mobile apps that communicate with a server usually do not follow security guidelines to implement SSL/TLS protocols	[75], [58], [84]
	Security testing is neglected with respect to other q.c.(s), hence there is a need for automated tools to detect vulnerabilities	[28]
	Apps are not tested against known vulnerabilities	[12]
	The usage of third-party apps and frameworks may introduce vulnerabilities	[72]
U	Usability may impact on the success of the mobile apps in the market	[79], [61], [49], [52], [25]
	App usability should be tested in devices that have different display characteristics	[49], [46], [20], [13]
	Usability testing usually requires extra cost, specific expertise, and time	[38], [79], [18], [52], [25], [26]
	Developers do not have proper knowledge about usability, therefore they are not careful with this q.c	[38], [26]
P	Mobile apps have to work correctly in mobile devices and platforms with very different characteristics.	[50], [42], [91], [56], [15]
	Portability testing is a very time-consuming task	[91]
	Existing solutions are not applicable in portability testing	[50]
	Lack of solutions to support UI portability	[42]
C	Huge diversity of mobile devices and platforms where mobile apps should work properly	[50]
R	Need to assess the correct behavior of large-scale mobile systems that have plenty of users and handling many business operations and transactions	[69]

for testing this q.c. All the primary studies addressing the security testing claimed that *developers do not care about security when they implement mobile apps, as a consequence specific techniques and tools are needed for this q.c.*. Regarding the usability q.c. the most common motivations are related to the proposal of cost-effective techniques since *usability testing usually requires extra costs, specific expertise, and time.* Moreover the usability q.c. has to be properly tested since *it may impact on the success of the mobile apps on the market.* As for the portability q.c., all the authors addressed this topic as techniques and tools are needed to assess that *mobile apps work correctly when they are deployed on devices and platforms having very different characteristics.*

Distribution of the q.c. (s) over the years: As Figure 7 shows, the NFR testing topic was addressed for the first time in 2009 in Kim et al. [43] by presenting a performance testing technique. In 2012, the Usability started becoming interesting for the community. Hereafter, each year, except for the 2015, at least one study about Usability testing has been proposed. Primary studies on security testing, from 2013 and on performance testing, since 2014, have been continuously published each year. Only in 2017, no primary studies about security were published. The first primary study on portability testing appeared in 2015 and the next one in 2016, 4 works about this topic were published in 2019. The only two primary studies on reliability and compatibility testing have been respectively published in 2017 and 2019.

6.2 $RQ2_2$ Mobile Platforms

Figure 8 shows the distribution of the mobile platforms considered in the primary studies. As reported in the figure, Android has been the most exploited platform as it was considered in 91.1%

(51/56) of the selected works, followed by iOS, 4.4% (2/56). The other platforms, i.e. Windows Phone, Blackberry and LG were taken into account just in one study, 1.8%. Two works did not consider a specific mobile platform. They presented an independent testing platform. It is worth pointing out that 5 of the considered studies presented a testing technique for hybrid mobile apps, which are apps combining elements of both native and web apps. They can be developed using frameworks, such as React Native[5] or Flutter[6]. The widespread use of Android as a case study can be justified by the fact that it is the most widespread mobile platform worldwide and it is also open source. Moreover, Android has a huge community of developers that provide both the versioning and issue tracking repositories for the applications they implement. As a consequence, researchers have easy access to a great number of applications that are at the same time open source (facilitating in-depth analyses of the experimental results) and representative of real daily usage mobile apps.

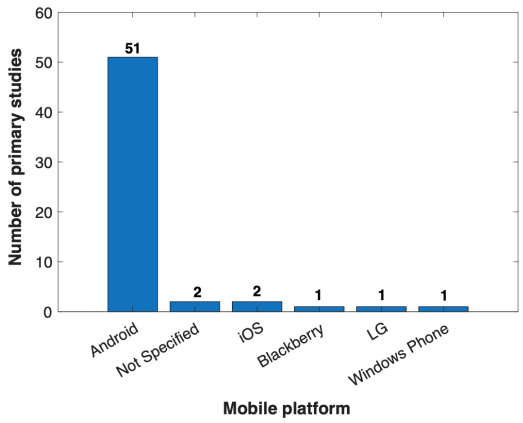

Fig. 8. Mobile platforms reported in primary studies.

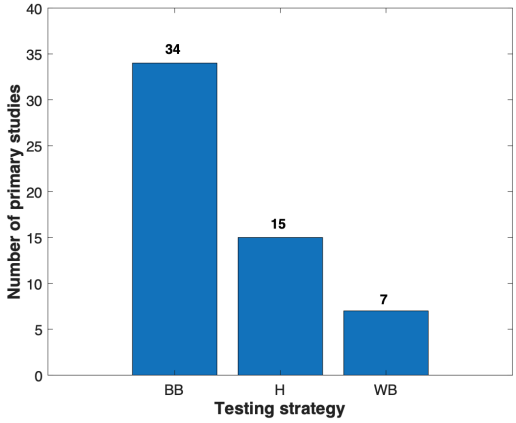

Fig. 9. Testing strategies reported in the primary studies.

6.3 $RQ2_3$ NFR Testing Strategies.

Figure 9 shows the distribution of strategies adopted for NFR testing. As shown in the figure, 60.7% (34/56) of the selected primary studies exploited a black box strategy, 12.5% (7/56) introduced a white box strategy, and 26.8% (15/56) adopted a hybrid strategy.

Figure 10 shows the quantity of studies reporting a given strategy for each specific q.c.. As we can see, the black box strategy is the most used one and has been applied for all q.c. found in this SMS. Both hybrid and white box strategies have been utilized only for testing security or performance q.c.. Table 13 reports the testing strategy adopted by each work.

White box strategies. Figure 10 shows that white box strategies have been adopted only for performance efficiency (3/7) and for security (4/7) testing. We inferred two types of white box strategy.

*WB*1: *Strategy generating test cases from app models.* Different types of models have been considered, including Finite state machines (FSMs) modeling the behavior of the entire app [72], combinations of use cases and user interface models [88], FSMs modeling UI behavior [24], models or metamodels capable of characterizing performance aspect [76, 81].

*WB*2: *Strategy injecting mutants into the source code of the tested app.* What varies is the kind of injected mutant. Energy mutants are exploited to test the performance [35], whereas mutants

[5]https://reactnative.dev/

[6]https://flutter.dev/, that allow to create code once and use it across various platforms.

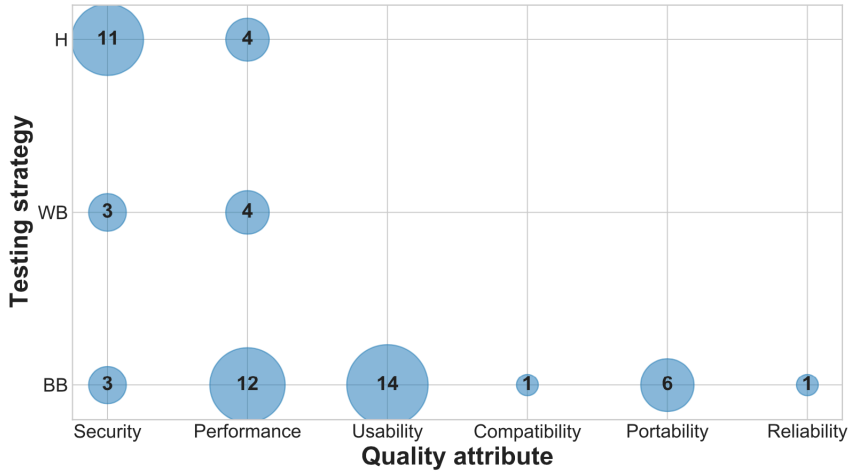

Fig. 10. Testing strategies adopted in NFR testing.

implementing intents that may introduce potential vulnerabilities are used in security testing [9].

Hybrid strategies. The bubble chart shown in Figure 10 shows that techniques exploiting a hybrid strategy have been presented only for testing the security (11/15) and the performance efficiency (4/15). We inferred two types of hybrid strategies.

*H*1: *Strategy generating test cases from inferred models* The strategy exploits artifacts and models inferred by app reverse engineering. Finite State Machines (FSM) [34, 75, 82, 90] and Control Flow Graph (CFG) models of two types, i.e., Control-Flow Graph of Call-Backs (CCFG) [41] and Window Transition Graph (WTG) [55] are exploited by this type of strategy.

*H*2: *Strategy based on app exploration and code scanning.* The strategy combines two steps of app dynamic exploration and code scanning. In the former step, the app is tested on the fly by exploring its UI [12, 58], launching specific APIs and intents [7], or executing tests that simulate attacks [28] or tests designed to find known errors [47]. The exploration can be also be driven by models such as Event Flow Graph(EFG) [10], or CFGs [29] or by source code knowledge [84, 85]. In the second step, the code of the app under test is scanned for finding quality issues.

Black box strategies. As Figure 10 shows, black box strategies have been used for testing all the quality characteristics of mobile apps. The usability, compatibility, portability, and reliability quality characteristics are tested only by techniques exploiting this strategy. From the analysis of the testing techniques we inferred five types of black box strategies.

*BB*1: *Strategy based on the analysis of recorded user interactions.* The user is free to use the app or to execute specific scenarios. Meanwhile specific logs are collected, saved, and then analyzed. This is the main strategy used for usability testing, and is proposed in 13 of the 14 studies addressing this q.c. [11, 13, 18, 20, 25, 26, 38, 46, 49, 52, 60, 61, 79]. The analysis of energy and memory usage logs are also used in performance efficiency testing [1, 43].

*BB*2: *Strategy based on app exploration.* The application is explored through the UI and tested on the fly to find vulnerabilities [70, 74, 84], performance or energy inefficiencies [51, 57], or portability issues [15, 51].

*BB*3: *Strategy based on ML.* Machine learning is used to find energy inefficiencies [19, 36, 92].

*BB*4: *Strategy based on execution of predefined sequences of user events.* Tests are made by sequences of user events used to request the UI of the apps. Test cases are executed on multiple devices

to test the portability [42, 50, 56, 91] and the compatibility [50] of the apps. Sequences of events resembling specific usage scenarios are executed while the performances and energy consumption of the apps are monitored to test their performance efficiency [6, 21, 53, 83].

*BB*5: *Strategy based on input partitioning*. The test input parameters are partitioned in two equivalence classes, i.e. fixed parameters, parameters that can be set by the hardware components, and parameters that can be set by the user. Based on this classification, the test cases are designed and implemented in performance efficiency testing [65].

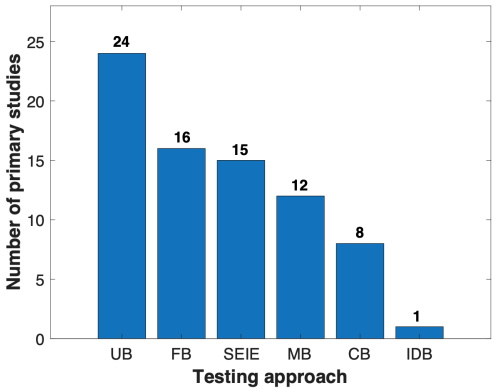

Fig. 11. Testing approaches reported in primary studies.

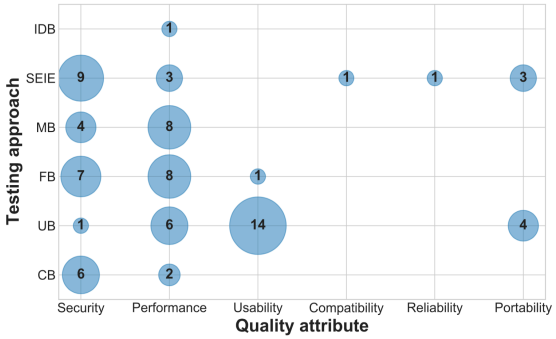

Fig. 12. Relationship between testing approaches and quality attributes.

6.4 $RQ2_4$ Approaches for NFR testing

According to the SWEBOK classification of testing approaches reported in Subsection 2.2, we found that NFR testing has been addressed by all the proposed categories of approaches, except one, the *Based on the Nature of the Application (BNA)* category. Figure 11 shows the distribution of the testing approaches in the analyzed primary studies. The *Usage Based (UB)* approach was the most used one as it was presented in 42.8% (24/56) of the analyzed papers, followed by *Fault Based (FB)* 28.6% (16/56), *Based on Software Engineer's Intuition and Experience (SEIE)* 26.8% (15/56), *Model Based (MB)* 21.4% (12/56), *Code Based (CB)* 14.3% (8/56), and *Input Domain Based (IDB)* 1.8% (1/56). The bubble chart in Figure 12 indicates how many times a given approach has been applied for testing a specific q.c.. Moreover, Table 13 points out the subcategory it adopts for each testing approach. In the following section, we discuss in more detail the characteristics of the proposed testing techniques and present a mapping for each q.c. that intends to provide an overview of the SWEBOK categories of adopted approaches and the respective subcategories. Moreover, the mapping indicates the studies that presented approaches belonging to each specific subcategory.

6.4.1 Testing approaches for performance efficiency testing. Performance testing is defined as a type of testing conducted to evaluate the degree to which a test item accomplishes its designated functions within given constraints of time and other resources [33]. Twenty different performance testing techniques emerged from our study that adopt the approaches summarized by the mapping shown in Figure 13. This Figure reports both single approaches and combinations of two approaches, which are represented by two incoming red edges entering a connector node. The most used approaches are *UB* and *MB*, which have been each presented in 4 studies. All the *UB* approaches are of *OP* subcategory and execute test cases comprising sequences of user events to generate log files reporting performance and consumption of the apps [6, 21, 43, 83]. *MB* approaches generate test cases either from *FSM* describing the app behavior [24, 34] or *FS* able to characterize performance

aspects [76, 81]. The other approaches either belong to the *ET* subcategory of *SEIE*, which test the app performance by exploring the UI on the fly [51], or are of *MT* subcategory of *FB* injecting energy mutants into the app source code [35]. Eventually, a *CFB* approach of *CB* category has been proposed to generate test cases starting from the CCFG [55], and an *EP* approach of category *IDB* has been presented where the test input parameters are partitioned in equivalence classes based on the consumption they may cause [65].

We also found eight testing techniques that exploit five combinations of two different approaches. *MB* and *FB* have been used together in two ways, i.e., *EG + FS* and *EG + FSM*. The former one is used to implement a machine learning testing technique [19, 36, 92] where tests are generated from a mathematical formal specification of an objective function aimed at guessing energy consumption. The second combination generates test cases to cover specific paths of an FSM modeling the app behavior. The approach covers paths that may expose potential memory leaks. *FB* has also been combined with *UB* and *SEIE* in *EG + OP* and *EG + ET* approaches, respectively. The former one requests the app using tests made by user events sequences covering specific usage scenarios. Tests are executed logs and are traced and stored. These logs are automatically analyzed to guess inefficiency errors [1, 53]. The latter approach explores the app to discover known inefficiencies [57]. *SEIE* was used with *CB* as *CFB + ET* that tests the app while the UI is explored. The exploration is driven by the EFG model of the app under test [10].

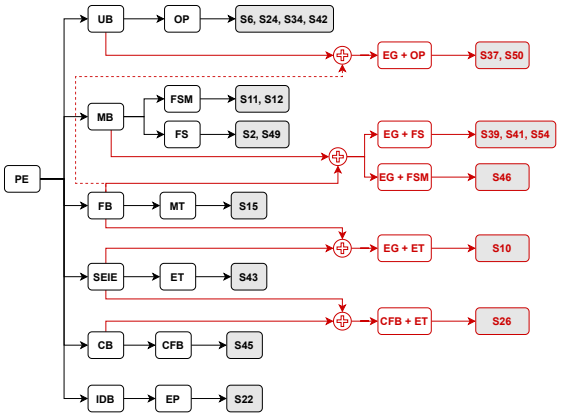

Fig. 13. Mapping of approaches for performance efficiency testing.

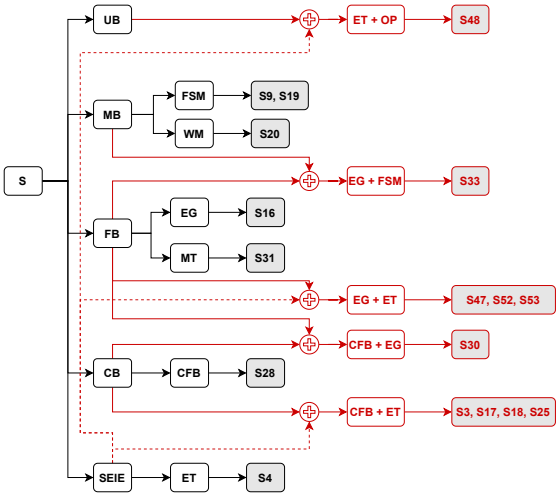

Fig. 14. Mapping of approaches for security testing.

6.4.2 Testing approaches for security testing. This type of testing is conducted to evaluate the degree to which a test item, and associated data and information, are protected so that unauthorized persons or systems cannot use, read, or modify them, and authorized persons or systems are not denied access to them [33]. As the Figure 14 shows, 7/17 techniques rely on a single approach. Three *MB* approaches generate tests from *FSM* [72, 75] or *WM*, such as UML use case and user interface models [88]. The two *FB* approaches belong to the *EG* and *MT* subcategories. The former requests the app by tests simulating known attacks [28], the latter injects into the app source code specific vulnerability mutants [9]. The *CB* approach of subcategory *CFB* generates the test cases starting from the CFG [41], whereas the *SEIE*, classified as *ET*, tests the app on the fly while the UI is explored searching for vulnerabilities [70].

Five different combinations of two approaches have been presented in 10/17 primary studies. *FB* was combined with the *SEIE* in *EG + ET* subcategory approaches that explore the app with the

main aim of triggering specific vulnerabilities [74, 84, 85]. *SEIE* and *CB* have been used jointly as *CFB + ET*. The approaches belonging to this subcategory find vulnerability both by exploring the app on the fly and by scanning the source code [7, 12, 29, 58]. *SEIE* was also combined with *UB* as *ET + OP* that explores the app on the fly for tracing log files that are analyzed to find vulnerabilities [54]. The integration of *FB* and *CB* in *CFG + EG* tests the app both using test cases designed for finding known errors and code scanning [47]. Finally, the combination of *FB* and *MB* as *EG + FSM* generates test cases covering paths of the FSM modeling the app behavior that may reveal potential vulnerabilities [82].

6.4.3 Testing approaches for usability testing. This type of testing is conducted to evaluate the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use (see ISO 9241-11). As Figure 15 shows, all the 14 techniques proposed to test this q.c. rely on *UB*. 13 of them are based on a single *UB* approach. Among these latter approaches, 11 belong to the *UOH* subcategory, leaving the users free to use the app under testing through the UI and the interactions are logged and analyzed to find usability issues [13, 18, 20, 26, 38, 46, 49, 52, 60, 61, 79]. Two approaches are instead classified as *OP*, where the users are asked to execute tasks covering specific requirements and the interactions are logged and analyzed to find usability issues [21, 25]. *UB* was also combined with *FB* in the *EG + UOH* approach, where the users have to request the application under testing with specific sequences of events that may cause potential usability errors.

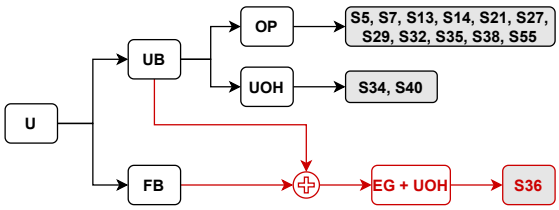

Fig. 15. Mapping of approaches for usability testing.

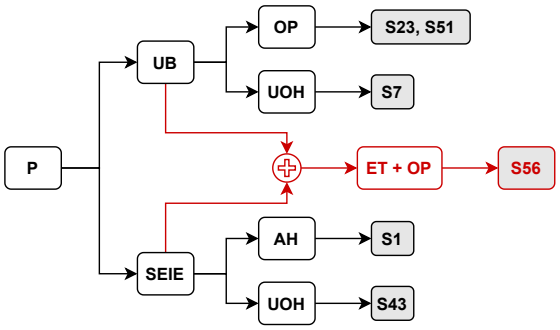

Fig. 16. Mapping of approaches for portability testing.

6.4.4 Testing approaches for portability testing. This type of testing is conducted to evaluate the ease with which a test item can be transferred from one hardware or software environment to another, including the level of modification needed for it to be executed in various types of environment [33]. As Figure 16 shows, among the 6 approaches used to test this q.c., 3 are classified as *UB*, 2 as *SEIE*, and 1 as combination of *UB* and *SEIE*. Among the three *UB* approaches, 2 belong to the *OP* subcategory and rely on the execution of test cases, made by sequences of user events, on multiple devices [56, 91]. The other *UB* is classified as *UOH* and is based on the analysis of record user interactions performed on multiple devices [79]. As for the two *SEIE* approaches, one belongs to *AH* since it executes test cases, generated ad-hoc, on multiple devices [50], the other one, classified as *ET*, explores the same app on the fly on multiple devices [51]. The combination of *UB* and *SEIE* falls in the *ET + OP* subcategory as it explores the app and the execution logs are analyzed to find portability issues [15].

6.4.5 Testing approaches for reliability and compatibility testing. Reliability testing is a type of testing conducted to evaluate the ability of a test item to perform its required functions, including evaluating the frequency with which failures occur, when it is used under stated conditions for a

specified period of time [33]. Compatibility testing instead measures the degree to which a test item can function satisfactorily alongside other independent products in a shared environment (co-existence), and where necessary, exchanges information with other systems or components (interoperabilility) [33]. *SEIE* approaches classified as *AH* have been exploited for both reliability and compatibility testing of mobile apps. These approaches generate ad hoc test cases implementing specific usage scenarios to test the reliability of the apps [69], or executed on multiple devices having different characteristics to check the absence of compatibility issues [50].

7 G3: - TOOLS FOR SUPPORTING NFR TESTING FOR MOBILE APPS

In this Section, we discuss which of the proposed NFR testing techniques are tool supported (Subsection 7.1), how the software of such tools is licensed (Subsection 7.2), and what type of functionality, according to the SWEBOK classification, is provided by the tools for each specific q.c. testing (Subsection 7.3).

7.1 $RQ3_1$ Tool support to NFR testing of mobile apps

The 69.6% (39/56) of the selected studies introduced or exploited a tool, the remaining 30.4% (17/56) did not explain whether and how the presented technique was tool supported. Table 14 lists the studies that introduced or exploited a tool. For each study, the Table reports the tool name (if available), tool type according to the SWEBOK classification we adopted, the the URL where it can be reached (if available). As the Table shows, 13/39 studies described a tool but did not name it. We observed that none of the tools was described or used in more than one study.

7.2 $RQ3_2$ Testing tool licensing

Among the tools that have been made available by the authors, 69% (27/39) of the them were distributed without any license, 23.0% (9/39) were distributed as Free Open-Source Software (FOOS), 2.6% (1/39) as Freeware, and 5.2% (2/39) is Proprietary software. Proprietary tools are IntentDroid developed by IBM, and ZIPT (Zero-Integration Performance Testing) developed by the University of Illinois in collaboration with Google. We observed that supporting tools that have a Proprietary license were usually proposed in primary studies fully or partially conducted by Industry. We believe that the Industry collaboration in these primary studies influences how the developed tools are made available. Supporting tools with FOSS and Freeware license are usually proposed in primary studies fully conducted by Academics, and their executable file or code is normally publicly available.

7.3 $RQ3_3$ Types of NFR testing tools

The tools presented in literature to support the execution of testing processes for NFRs have been classified on the basis of the functionality they offer, according to the classification provided by the *SWEBOK* [14] and described in Subsection 2.3. Figure 17 shows the distribution of tool types, whereas Figure 18 reports for each q.c. the number of studies where a given tool type has been adopted. As Table 14 shows, 30 tools provide more than one feature. The map reported in Figure 19 shows the types of features and feature combinations of the tools used to testing each q.c.. Moreover, the ID of the paper, where the tool has been presented, is traced to each type or combination of types. Tools for performance efficiency and security have a wider variety of feature combinations. As expected, tools for performance efficiency mostly offer *TH*, *OC*, and *TG* features, since they have to provide a complex test harness to execute the test cases or explore the app, to collect the consumption logs, and to analyze them trough complex oracles for checking the presence of performance issues. Specific test generators are needed to produce performance test cases from the models. The tools used for security have to provide *TH*, *TG*, *CAI*, and *OC* features. Indeed they

have to provide a complex test harness to explore the app with the goal of finding vulnerabilities as well as the feature for generating security tests from models. These tools should also provide the functionality to inspect or instrument the code in order to guide the navigation of the app or to find known vulnerabilities. Usability tools mainly provide the *CR* feature for enabling the capture of user events and for replaying them in order to find usability issues on the user interface. The *T* and *OC* features are respectively used to trace the user interactions, e.g. for collecting the encountered user interfaces or eyes tracking, and to implement complex oracles able to check usability issues automatically. Portability tools mainly offer a *TH* feature as they have to offer a complex software infrastructure to execute test cases on multiple devices.

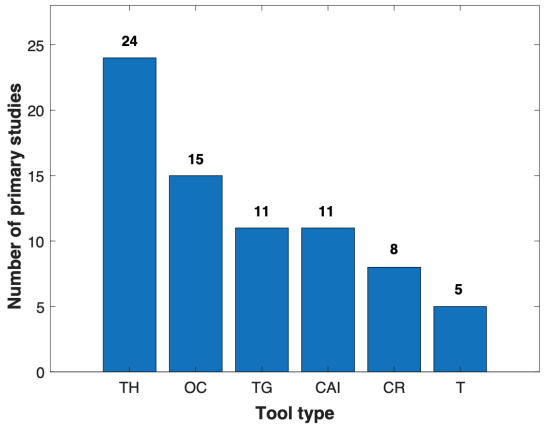

Fig. 17. Types of supporting tools.

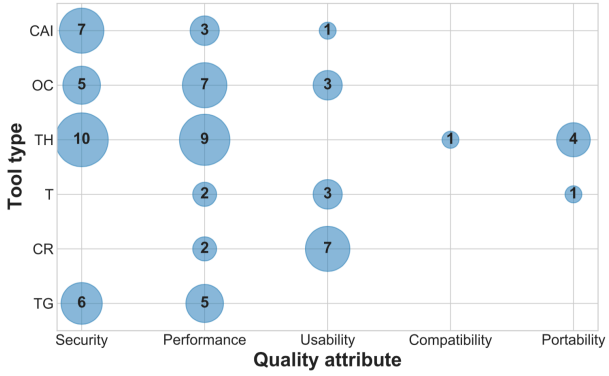

Fig. 18. Relationship between types of supporting tools and quality attributes.

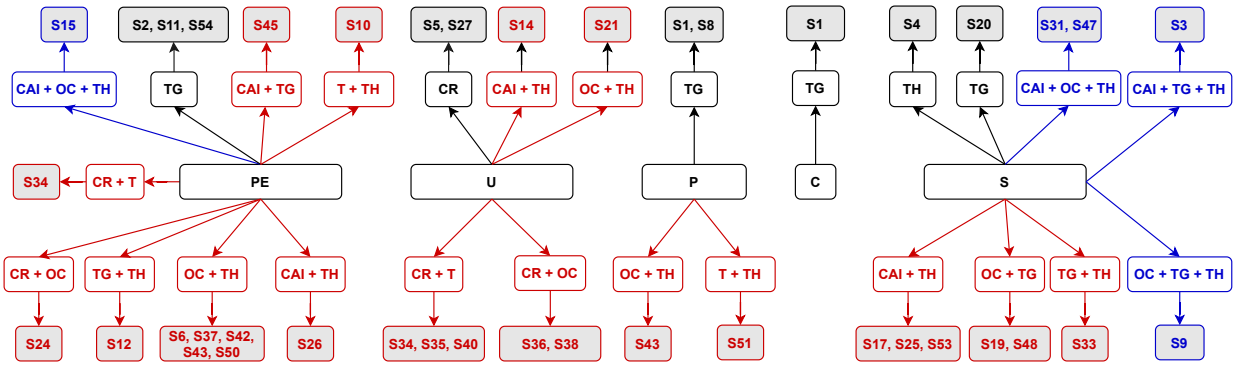

Fig. 19. Tool types for each quality characteristic.

8 DISCUSSION

In the following section, we summarize our findings for each research goal. Then, we discuss the open research challenges that we identified.

8.1 Findings about the G1 research goal

G1 aimed at characterizing the research works in the area of NFRs testing of mobile apps. The results of our study show a growing interest from the research community in this topic, likely depending on a growing awareness of the relevance of NFR testing for the success of mobile

apps. This topic has been addressed in a variety of relevant publication venues of the software engineering community. Most contributions were fully developed in academia, but in the last 6 years more industrial involvement in this research field has been recorded. Concerning the geographic distribution of the contributors, the most prolific countries included China, USA, Canada and Brazil, with at least 4 contributions each.

8.2 Findings about the G2 research goal

G2 aimed at characterizing the NFRs techniques for mobile apps from the perspectives of the addressed quality characteristic, the tackled mobile platforms, the adopted testing strategies and testing approaches. Security, performance, and usability testing of mobile apps were of interest to the software engineering community since 2009 and are still being investigated with much interest. More recently, portability testing is gaining growing attention, likely due to the wide spread of cross-platform mobile app development frameworks. Android is the most considered mobile platform, being addressed in more than 91% of the considered studies. As to the strategies adopted for NFRs testing, we found that White Box and Hybrid strategies have been exclusively used for performance efficiency and security testing, indicating that these types of testing require knowledge of how the app has been designed and implemented. Black box strategies have been instead widely used to test all the considered q.c., especially for performance and usability testing. As regards the NFRs testing approaches, different categories have emerged for testing each q.c.. Using the taxonomy proposed by the SWEBOK, we were able to classify and map them in six main categories, and deduce which are the most frequently used ones for each q.c.

8.3 Findings about the G3 research goal

G3 aimed to analyze the tool support offered to NFRs testing techniques. Most of the techniques proposed for NFR testing of mobile apps are tool supported. The vast majority of tools have been developed in academic contexts, being distributed without any license or with open-source ones, whereas the minority of tools developed in industry is proprietary software. Most of the presented tools belong to the Test Harness (TH) category, followed by the Oracle comparator and assertion checking (OC) one. Surprisingly, we did not find tools able to support Regression Testing (RT). TH tools are usually exploited for Security, Performance and Portability testing. Capture/Replay (CR) tools are exclusively used for Usability and Performance testing. Many of the tools offer a variety of functionality, able to support the different testing process activities. We provided a mapping that offers the reader an overview of the existing tools for each q.c. and the type of support each one provides. Moreover, we have checked if any of the support tools provide functional testing. In this case, we have identified that one study [21] provides a support tool that addresses performance testing (e.g., completion rate, time on task) and reports a summary of app functionality issues.

8.4 Open challenges for researchers

Based on the evidence we obtained from the data discussion, we describe some of the identified several open research challenges:

Addressing specific q.c. (s). Testing of specific quality characteristics in mobile apps is still an open issue. Our analysis showed that testing q.c. like Compatibility and Reliability, is still underdeveloped. Significant effort should be spent to develop suitable techniques and tools implementing this specific type of testing.

NFRs testing of mobile hybrid apps. In the last years, Hybrid mobile apps are becoming increasingly common on mobile devices. Testing this specific type of mobile apps is still an open issue, and a few research contributions have emerged about this topic. Since these apps are developed by combining web and mobile technologies, or using specific frameworks, some testing techniques

may need to be properly adapted for taking into account the characteristics of such new generation of mobile apps [63]. As an example, white box or gray box strategies for NFR testing should be readjusted for taking into account the specific technologies used for implementing them.

Future extensions of the Mapping Study. To the best of our knowledge, our secondary study was the first one addressing the NFR testing for mobile apps. According to our research methodology, we defined inclusion and exclusion criteria for selecting relevant articles about dynamic testing techniques. However, during our study we made some observations that allowed us to identify some possible interesting extensions of this mapping study. As an example, due to the exclusion criterion EC-8 reported in Subsection 4.3, we could not consider 14 papers presenting a static testing technique mainly in the context of security. Extending our secondary study to static analysis techniques may be a relevant research topic that could lead interesting results for the community. Moreover, according to EC-11, we discarded 26 papers proposing techniques aimed to find generic errors (defects, failures, or crashes), without clearly declaring the tested quality characteristic. Extending the mapping to include such types of contributions would certainly require an extra effort for inferring and classifying the q.c. addressed by them. However, it could provide a wider overview about the state of the art in this field and open novel discussions.

Making supporting testing tools available. Practitioners are interested in accessing the existing tools for NFRs testing, in order introduce such tools in their testing processes. Analogously, researchers are interested in accessing the existing tools, in order to evaluate and possibly improve them. Our study showed that 69.2% (27/39) of the tools proposed in literature have not been made publicly available. This is a significant obstacle when researcher or practitioners intend to use a given NFRs testing technique. More attention to this aspect should be given. Making testing tools available can be of great benefit to both the researcher and practitioner community.

Sharing industrial experiences with the academic community. We observed that only 3.6% of the primary studies were fully conducted by industry and 10.7% come from crosscutting studies. It could be very interesting and useful for the cultural growth of the community to share, in the next future, the experiences and the results obtained by a more and more extensive collaboration between the academia and the industry.

Using a consolidate terminology for q.c. (s). We observed that 52 of the 56 selected primary studies did not use a consolidate and acknowledged terminology for indicating the addressed q.c.. We had to infer the NFR by full reading the papers and applying the definition of the quality models proposed by the standard [32]. The lack of using a unified terminology may lead the authors to make mistakes in the term definitions and conduction of their studies. For instance, some studies confounded compatibility and portability quality attributes since they were related to the adaptability sub characteristic of the portability. Therefore, we believe that the use of a consolidate and well established terminology may improve the quality of the works and make more clear for which q.c. they have been proposed.

9 THREATS TO VALIDITY

In this section, we discuss some threats to validity of the study and the actions taken in order to mitigate them. Some decisions made during systematic mapping review activities can be characterized by subjectivity, mainly influenced by the researchers' knowledge and viewpoints. We tried to mitigate the following threats to validity by obeying to the actions proposed by Ampatzoglou et al. [8].

Study selection: this threat to validity regards to aspects related to the first activities carried out during the initial stages of the research protocol. Its main objective is to warn about risks related to the study selection process. To mitigate this threat, we highlight that the research was conducted using strict protocols for performing systematic mappings Petersen et al. [67] and to define the

objectives, research questions and metrics to be evaluated in the study [16]. Another aspect related to this threat regards the procedure to identify the studies. We used a set of well-known search engines to collect the primary studies. As suggested by Dyba et al. [23] and Kitchenham and Charters [45], to mitigate the risk of losing important studies we used a broad search engine and indexation database (SCOPUS) and computer science-specific ones (IEEE, ACM, and Web of Science). Moreover, we also performed backward and forward snowballing process to identify missed studies that were not returned by search engines. Another threat regards the application of the inclusion/exclusion criteria for studies selection. This step is threatened by the possibility of excluding relevant works. To mitigate this threat, two researchers have been involved in this process. Studies were included/excluded if they were accepted/discarded by both. Doubt papers were discussed by the two researchers and a third one. An additional threat regards the strategy used to build the search string. The keywords of the search string were chosen on the basis of the experience of the authors, as well as insights from related works and well known terms and definitions. Moreover we followed a process to define and calibrate the search string that relies on the adoption of control studies to measure the adequacy of the selected terms.

Data Validity: the main threat is related to bias during the process of data extraction. To mitigate this threat data were extracted and recorded manually by two authors. A third author, having the best background in the research topic, analyzed the data and inferred the results. Finally, the results were discussed by all authors to settle errors and inconsistencies.

Research Validity: this threat is related to the generalization of the results and to the coverage of research questions. To mitigate this threat, we followed a planned protocol where the scientific methods, goals, and research questions, all decisions were based on well-established guidelines for conducting systematic mapping studies in the software engineering area. All the steps we followed are well documented and all the produced material (list of selected papers, extracted data and plots) is publicly available [39]. This allows the replications of the study and the check of the obtained results by other researchers.

10 CONCLUSIONS

Mobile apps have gained wide popularity in recent years and new approaches from software engineering are required to ensure their quality. For instance, a mobile app can consume too much power to perform a task, the user interface can behave unexpectedly, or the user data can be susceptible to attacks during an insecure connection with a server. These problems are related to NFRs and they are fundamental features in a mobile app. Because of this, relevant studies have been conducted aiming to characterize test techniques on mobile apps.

Inspired by this scenario, we conducted a set of discussions on the main aspects related to NFRs dynamic testing on mobile apps. In this perspective, we provide an overview of the existing test techniques that address the different NFRs defined according to the quality standard from ISO [32]. Then, we characterized the test techniques identified mainly discussing the following topics: (1) collaboration between academia and industry; (2) testing approaches and strategies; (3) most addressed mobile app types and platforms; and (4) tool support. Moreover, we provided a broad discussion to the testing community on the main trends and research opportunities in this field of research. All artifacts generated from this SMS have been made publicly available, and they should be widely used by the testing community as a support to explore research opportunities, gaps, and future directions.

As future work, we intend to increase our investigation by searching from the gray literature. Moreover, in this SMS we focus on dynamic testing techniques, excluding the possibility of discussing on static testing commonly used in security testing. Therefore, we intend to include a specific investigation from static testing.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. Stevão A. Andrade research was funded by Fundação de Amparo a Pesquisa do Estado de São Paulo - FAPESP (São Paulo Research Foundation), process number 2017/19492-1.

REFERENCES

[1] Abdul Muqtadir Abbasi, Mustafa Al-Tekreeti, Kshirasagar Naik, Amiya Nayak, Pradeep Srivastava, and Marzia Zaman. 2018. Characterization and Detection of Tail Energy Bugs in Smartphones. *IEEE Access* 6 (2018), 65098–65108. https://doi.org/10.1109/access.2018.2877395

[2] Kevin Adams. 2015. *Non-functional Requirements in Systems Analysis and Design*. Vol. 28. Springer, Cham. 264 pages. https://doi.org/10.1007/978-3-319-18344-2

[3] A. S. Al-Ahmad, H. Kahtan, F. Hujainah, and H. A. Jalab. 2019. Systematic Literature Review on Penetration Testing for Mobile Cloud Computing Applications. *IEEE Access* 7 (2019), 173524–173540. https://doi.org/10.1109/ACCESS.2019.2956770.

[4] Diego R Almeida, Patrícia DL Machado, and Wilkerson L Andrade. 2019. Testing tools for Android context-aware applications: a systematic mapping. *Journal of the Brazilian Computer Society* 25, 12 (2019), 1–22. https://doi.org/10.1186/s13173-019-0093-7

[5] Ali Alotaibi, James Clause, and William G.J. Halfond. 2020. Mobile App Energy Consumption: A Study of Known Energy Issues in Mobile Applications and their Classification Schemes – Summary Plan. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, New York, NY, USA, 854–854. https://doi.org/10.1109/ICSME46990.2020.00109

[6] Domenico Amalfitano, Vincenzo Riccio, Porfirio Tramontana, and Anna Rita Fasolino. 2020. Do Memories Haunt You? An Automated Black Box Testing Approach for Detecting Memory Leaks in Android Apps. *IEEE Access* 8 (2020), 12217–12231. https://doi.org/10.1109/access.2020.2966522

[7] Amr Amin, Amgad Eldessouki, Menna Tullah Magdy, Nouran Abdeen, Hanan Hindy, and Islam Hegazy. 2019. AndroShield: Automated Android Applications Vulnerability Detection, a Hybrid Static and Dynamic Analysis Approach. *Information* 10, 10 (2019), 326. https://doi.org/10.3390/info10100326

[8] Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. 2019. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and Software Technology* 106 (2019), 201–230. https://doi.org/10.1016/j.infsof.2018.10.006

[9] Andrea Avancini and Mariano Ceccato. 2013. Security testing of the communication among Android applications. In *Proceedings of the 8th International Workshop on Automation of Software Test (AST)*. IEEE, New York, NY, USA, 57–63. https://doi.org/10.1109/iwast.2013.6595792

[10] Abhijeet Banerjee, Lee Kee Chong, Clément Ballabriga, and Abhik Roychoudhury. 2017. Energypatch: Repairing resource leaks to improve energy-efficiency of android apps. *IEEE Transactions on Software Engineering* 44, 5 (2017), 470–490. https://doi.org/10.1109/tse.2017.2689012

[11] Silvio Barra, Rita Francese, and Michele Risi. 2019. Automating Mockup-Based Usability Testing on the Mobile Device. In *Proceedings of the International Conference on Green, Pervasive, and Cloud Computing (GPC)*. Springer, Cham, 128–143. https://doi.org/10.1007/978-3-030-19223-5_10

[12] Shikhar Bhatnagar, Yasir Malik, and Sergey Butakov. 2018. Analysing Data Security Requirements of Android Mobile Banking Application. In *Proceedings of the International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments (ISDDC)*. Springer, Cham, 30–37. https://doi.org/10.1007/978-3-030-03712-3_3

[13] Magdalena Borys and Marek Milosz. 2018. Mobile Application Usability Testing in Quasi-Real Conditions-the Synergy of Using Different Methods. In *Proceedings of the 11th International Conference on Human System Interaction (HSI)*. IEEE, New York, NY, USA, 362–368. https://doi.org/10.1109/hsi.2015.7170698

[14] Pierre Bourque, Richard E. Fairley, and IEEE Computer Society. 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0* (3rd ed.). IEEE Computer Society Press, Washington, DC, USA.

[15] Haipeng Cai, Ziyi Zhang, Li Li, and Xiaoqin Fu. 2019. A large-scale study of application incompatibilities in android. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, New York, NY, USA, 216–227. https://doi.org/10.1145/3293882.3330564

[16] Victor R Basili1 Gianluigi Caldiera and H Dieter Rombach. 1994. The goal question metric approach. *Encyclopedia of software engineering* 2 (1994), 528–532.

[17] Melissa Chau and Ryan Reith. 2020. Smartphone Market Share. https://www.idc.com/promo/smartphone-market-share/os. [Online; accessed 10-January-2020].

[18] Lin Chou Cheng. 2016. The mobile app usability inspection (maui) framework as a guide for minimal viable product (mvp) testing in lean development cycle. In *Proceedings of the 2nd International Conference in HCI and UX*. ACM, New York, NY, USA, 1–11. https://doi.org/10.1145/2898459.2898460

[19] Shaiful Alam Chowdhury and Abram Hindle. 2016. Greenoracle: Estimating software energy consumption with energy measurement corpora. In *Proceedings of the 13th Working Conference on Mining Software Repositories (MSR)*. ACM, New York, NY, USA, 49–60. https://doi.org/10.1145/2901739.2901763

[20] Thiago Adriano Coleti, Leticia da Silva Souza, Marcelo Morandini, Suzie Allard, and Pedro Luiz Pizzigatti Correa. 2017. ErgoMobile: A Software to Support Usability Evaluations in Mobile Devices Using Observation Techniques. In *Proceedings of the International Conference of Design, User Experience, and Usability (DUXU)*. Springer, Cham, 363–378. https://doi.org/10.1007/978-3-319-58634-2_27

[21] Biplab Deka, Zifeng Huang, Chad Franzen, Jeffrey Nichols, Yang Li, and Ranjitha Kumar. 2017. ZIPT: Zero-Integration Performance Testing of Mobile App Designs. In *Proceedings of the 30th Annual Symposium on User Interface Software and Technology (UIST)*. ACM, New York, NY, USA, 727–736. https://doi.org/10.1145/3126594.3126647

[22] Marcio Delamaro, Mario Jino, and Jose Maldonado. 2016. *Introduction to Software Testing* (2nd ed.). Elsevier Brasil, USA.

[23] T. Dyba, T. Dingsoyr, and G. K. Hanssen. 2007. Applying Systematic Reviews to Diverse Study Types: An Experience Report. In *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, New York, NY, USA, 225–234. https://doi.org/10.1109/ESEM.2007.59

[24] Ana Rosario Espada, María del Mar Gallardo, Alberto Salmerón, and Pedro Merino. 2017. Performance analysis of Spotify® for Android with model-based testing. *Mobile Information Systems* 2017 (2017), 1–14. https://doi.org/10.1155/2017/2012696

[25] Xavier Ferre, Elena Villalba, Héctor Julio, and Hongming Zhu. 2017. Extending mobile app analytics for usability test logging. In *Proceedings of the International Conference on Human-Computer Interaction (HCI)*. Springer, Cham, 114–131. https://doi.org/10.1007/978-3-319-67687-6_9

[26] Jackson Feijó Filho, Thiago Valle, and Wilson Prata. 2015. Automated usability tests for mobile devices through live emotions logging. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*. ACM, New York, NY, USA, 636–643. https://doi.org/10.1145/2786567.2792902

[27] International Organization for Standardization. 2013. ISO/IEC/IEEE International Standard - Software and systems engineering – Software testing –Part 2:Test processes. , 68 pages.

[28] Chenkai Guo, Jing Xu, Hongji Yang, Ying Zeng, and Shuang Xing. 2014. An automated testing approach for inter-application security in Android. In *Proceedings of the 9th International Workshop on Automation of Software Test ((AST))*. ACM, New York, NY, USA, 8–14. https://doi.org/10.1145/3106237.3106244

[29] Roee Hay, Omer Tripp, and Marco Pistoia. 2015. Dynamic detection of inter-application communication vulnerabilities in Android. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*. ACM, New York, NY, USA, 118–128. https://doi.org/10.1145/2771783.2771800

[30] Gavin Henry. 2021. Matt Lacey on Mobile App Usability. *IEEE Softw.* 38, 2 (2021), 134–136. https://doi.org/10.1109/MS.2020.3042424

[31] ISO. 2001. *ISO/IEC 9126-1, Software engineering — Product quality*. ISO, Geneva, Switzerland.

[32] ISO. 2011. *ISO/IEC 25010:2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. ISO, Geneva, Switzerland.

[33] ISO. 2013. ISO/IEC/IEEE 29119-1: Software and systems engineering-Software testing-Part 1: Concepts and definitions.

[34] Reyhaneh Jabbarvand, Jun-Wei Lin, and Sam Malek. 2019. Search-based energy testing of Android. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. IEEE, New York, NY, USA, 1119–1130. https://doi.org/10.1109/icse.2019.00115

[35] Reyhaneh Jabbarvand and Sam Malek. 2017. μDroid: an energy-aware mutation testing framework for Android. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, New York, NY, USA, 208–219. https://doi.org/10.1145/3106237.3106244

[36] Reyhaneh Jabbarvand, Forough Mehralian, and Sam Malek. 2020. Automated construction of energy test oracles for Android. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, NY, USA, 927–938. https://doi.org/10.1145/3368089.3409677

[37] Marek Janicki, Mika Katara;, and Tuula Pääkkönen. 2012. Obstacles and opportunities in deploying model-based GUI testing of mobile software: A survey. *SOFTWARE TESTING, VERIFICATION AND RELIABILITY* 22(5) (Aug. 2012), 313–341. https://doi.org/10.1002/stvr.460

[38] JongWook Jeong, NeungHoe Kim, and Hoh Peter In. 2020. Detecting usability problems in mobile applications on the basis of dissimilarity in user behavior. *International Journal of Human-Computer Studies* 139 (2020), 102364. https://doi.org/10.1016/j.ijhcs.2019.10.001

[39] Misael Junior, Domenico Amalfitano, Lina Garcés, Stevão Andrade, and Márcio Delamaro. 2021. Dataset on Dynamic Testing Techniques of Non-Functional Requirements on Mobile Applications. https://data.mendeley.com/datasets/gswvb2s2ht/3. Accessed: 2021-12-29.

[40] Anureet Kaur and Kulwant Kaur. 2018. Systematic literature review of mobile application development and testing effort estimation. *Journal of King Saud University - Computer and Information Sciences* 1, 1 (2018), 1–22. https://doi.org/10.1016/j.jksuci.2018.11.002

[41] Joseph Chan Joo Keng, Lingxiao Jiang, Tan Kiat Wee, and Rajesh Krishna Balan. 2016. Graph-aided directed testing of Android applications for checking runtime privacy behaviours. In *Proceedings of the 11th International Workshop on Automation of Software Test ((AST))*. ACM, New York, NY, USA, 57–63. https://doi.org/10.1145/2896921.2896930

[42] Taeyeon Ki, Chang Min Park, Karthik Dantu, Steven Y Ko, and Lukasz Ziarek. 2019. Mimic: UI compatibility testing system for Android apps. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. IEEE, New York, NY, USA, 246–256. https://doi.org/10.1109/icse.2019.00040

[43] Heejin Kim, Byoungju Choi, and W Eric Wong. 2009. Performance testing of mobile applications at the unit test level. In *Proceedings of the 3rd International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*. Springer, Cham, 171–180. https://doi.org/10.1109/ssiri.2009.28

[44] B. Kitchenham, D. Budgen, and O.P. Brereton. 2010. The value of mapping studies: A participantobserver case study. In *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. BCS Learning and Development Ltd, Swindon, Wiltshire, 25–33. https://doi.org/10.14236/ewic/EASE2010.4

[45] B. Kitchenham and S Charters. 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering.

[46] Wolfgang Kluth, Karl-Heinz Krempels, and Christian Samsel. 2014. Automated Usability Testing for Mobile Applications. In *Proceedings of the Web Information Systems and Technologies (WEBIST)*. SciTePress, Setúbal, Portugal, 149–156. https://doi.org/10.5220/0004985101490156

[47] Konstantin Knorr and David Aspinall. 2015. Security testing for Android mHealth apps. In *Proceedings of the 8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, New York, NY, USA, 1–8. https://doi.org/10.1109/icstw.2015.7107459

[48] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, and J. Klein. 2019. Automated Testing of Android Apps: A Systematic Literature Review. *IEEE Transactions on Reliability* 68, 1 (2019), 45–66.

[49] Artur H Kronbauer, Celso AS Santos, and Vaninha Vieira. 2012. Smartphone applications usability evaluation: a hybrid model and its implementation. In *Proceedings of the International Conference on Human-Centred Software Engineering (HCSE)*. Springer, Berlin, Heidelberg, 146–163. https://doi.org/10.1007/978-3-642-34347-6_9

[50] Ammar Lanui and Thiam Kian Chiew. 2019. A Cloud-Based Solution for Testing Applications Compatibility and Portability on Fragmented Android Platform. In *Proceedings of the 26th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, New York, NY, USA, 158–164. https://doi.org/10.1109/apsec48747.2019.00030

[51] Jemin Lee and Hyungshin Kim. 2016. QDroid: Mobile Application Quality Analyzer for App Market Curators. *Mobile Information Systems* 2016 (2016), 1–11. https://doi.org/10.1155/2016/1740129

[52] Florian Lettner and Clemens Holzmann. 2012. Automated and unsupervised user interaction logging as basis for usability evaluation of mobile applications. In *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia (MoMM)*. ACM, New York, NY, USA, 118–127. https://doi.org/10.1145/2428955.2428983

[53] Xueliang Li, Yuming Yang, Yepang Liu, John P Gallagher, and Kaishun Wu. 2020. Detecting and diagnosing energy issues for mobile applications. In *Proceedings of the 29th International Symposium on Software Testing and Analysis*. ACM, New York, NY, USA, 115–127. https://doi.org/10.1145/3395363.3397350

[54] Hongliang Liang, Yudong Wang, Tianqi Yang, and Yue Yu. 2018. AppLance: A Lightweight Approach to Detect Privacy Leak for Packed Applications. In *Proceedings of the Nordic Conference on Secure IT Systems*. Springer, Cham, 54–70. https://doi.org/10.1007/978-3-030-03638-6_4

[55] Ao Liu, Jing Xu, Weijing Wang, Jiawei Yu, and Hongcan Gao. 2019. Automated Testing of Energy Hotspots and Defects for Android Applications. In *Proceedings of the International Conference on Energy Internet (ICEI)*. IEEE, New York, NY, USA, 374–379. https://doi.org/10.1109/ICEI.2019.00072

[56] Chien-Hung Liu. 2019. A compatibility testing platform for android multimedia applications. *Multimedia Tools and Applications* 78, 4 (2019), 4885–4904. https://doi.org/10.1007/s11042-018-6268-y

[57] Yi Liu, Jue Wang, Chang Xu, Xiaoxing Ma, and Jian L¨u. 2018. NavyDroid: an efficient tool of energy inefficiency problem diagnosis for Android applications. *Science China Information Sciences* 61, 5 (2018), 1–20. https://doi.org/10.1007/s11432-017-9400-y

[58] Yang Liu, Chaoshun Zuo, Zonghua Zhang, Shanqing Guo, and Xinshun Xu. 2018. An automatically vetting mechanism for SSL error-handling vulnerability in android hybrid Web apps. *World Wide Web* 21 (2018), 127–150. https://doi.org/10.1007/s11280-017-0458-9

[59] Chu Luo, Jorge Goncalves, Eduardo Velloso, and Vassilis Kostakos. 2020. A Survey of Context Simulation for Testing Mobile Context-Aware Applications. *ACM Comput. Surv.* 53, 1, Article 21 (Feb. 2020), 39 pages. https://doi.org/10.

1145/3372788

[60] Xiaoxiao Ma, Bo Yan, Guanling Chen, Chunhui Zhang, Ke Huang, Jill Drury, and Linzhang Wang. 2013. Design and implementation of a toolkit for usability testing of mobile apps. *Mobile Networks and Applications* 18 (2013), 81–97. https://doi.org/10.1007/s11036-012-0421-z

[61] Katherine M Malan, Jan HP Eloff, and Jhani A de Bruin. 2018. Semi-automated usability analysis through eye tracking. *South African Computer Journal* 30 (2018), 66–84. https://doi.org/10.18489/sacj.v30i1.511

[62] Abel Méndez Porras, Christian Ulises Quesada López, and Marcelo Jenkins Coronas. 2015. Automated testing of mobile applications: A systematic map and review. In *Proceedings of the 28th Ibero-American Conference on Software Engineering (CIBSE)*. URP,SPC,UCSP, Lima, Peru, 1–14.

[63] H. Muccini, A. Di Francesco, and P. Esposito. 2012. Software testing of mobile applications: Challenges and future research directions. In *Proceedings of the 7th International Workshop on Automation of Software Test (AST)*. IEEE, New York, NY, USA, 29–35.

[64] Glenford J. Myers, Corey Sandler, and Tom Badgett. 2011. *The Art of Software Testing* (3rd ed.). Wiley Publishing, New York, NY, USA.

[65] Kshirasagar Naik, Yasir Ali, Veluppillai Mahinthan, Ajit Singh, and Abdulhakim Abogharaf. 2014. Categorizing configuration parameters of smartphones for energy performance testing. In *Proceedings of the 9th International Workshop on Automation of Software Test (AST)*. ACM, New York, NY, USA, 15–21. https://doi.org/10.1145/2593501.2593504

[66] Simon O'Dea. 2020. Smartphone users worldwide 2016-2021. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Online; accessed 10-January-2020].

[67] K. Petersen, S. Vakkalanka, and L. Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

[68] Roger Pressman. 2016. *Software Engineering: A Practitioner's Approach* (9 ed.). McGraw-Hill, Inc., USA.

[69] Titis Sari Putri and Fatwa Ramdani. 2017. Reliability testing using hybrid exploratory basis of tour and fuzzy Inference System Tsukamoto. In *Proceedings of the International Conference on Sustainable Information Engineering and Technology (SIET)*. IEEE, New York, NY, USA, 176–183. https://doi.org/10.1109/siet.2017.8304131

[70] Vaibhav Rastogi, Yan Chen, and William Enck. 2013. AppsPlayground: automatic security analysis of smartphone applications. In *Proceedings of the 3rd Conference on Data and Application Security and Privacy (CODASPY)*. ACM, New York, NY, USA, 209–220. https://doi.org/10.1145/2435349.2435379

[71] Mehmet Sahinoglu, Koray Incki, and Mehmet S. Aktas. 2015. Mobile Application Verification: A Systematic Mapping Study. In *Proceedings of the International Computational Science and Its Applications (ICCSA)*. Springer, Cham, 147–163. https://doi.org/10.1007/978-3-319-21413-9_11

[72] Sébastien Salva and Stassia R Zafimiharisoa. 2015. APSET, an Android aPplication SEcurity Testing tool for detecting intent-based vulnerabilities. *International Journal on Software Tools for Technology Transfer* 17, 2 (2015), 201–221. https://doi.org/10.1007/s10009-014-0303-8

[73] João B. F. Sequeiros, Francisco T. Chimuco, Musa G. Samaila, Mário M. Freire, and Pedro R. M. Inácio. 2020. Attack and System Modeling Applied to IoT, Cloud, and Mobile Ecosystems: Embedding Security by Design. *ACM Comput. Surv.* 53, 2 (2020), 25:1–25:32. https://doi.org/10.1145/3376123

[74] Hossain Shahriar, Sarah North, and Edward Mawangi. 2014. Testing of memory leak in Android applications. In *Proceedings of the 15th International Symposium on High-Assurance Systems Engineering*. IEEE, New York, NY, USA, 176–183. https://doi.org/10.1109/hase.2014.32

[75] Shangcheng Shi, Xianbo Wang, and Wing Cheong Lau. 2019. MoSSOT: An Automated Blackbox Tester for Single Sign-On Vulnerabilities in Mobile Applications. In *Proceedings of the Asia Conference on Computer and Communications Security (ASIACCS)*. ACM, New York, NY, USA, 269–282. https://doi.org/10.1145/3321705.3329801

[76] Lady Silva and Denivaldo Lopes. 2020. Model Driven Engineering for Performance Testing in Mobile Applications. In *Proceedings of the 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE, New York, NY, USA, 1–7. https://doi.org/10.1109/seeda-cecnsm49515.2020.9221828

[77] Ian Sommerville. 2011. *Software Engineering* (9th ed.). Addison-Wesley Publishing Company, USA.

[78] Oleksii Starov, Sergiy Vilkomir, Anatoliy Gorbenko, and Vyacheslav Kharchenko. 2015. *Testing-as-a-Service for Mobile Applications: State-of-the-Art Survey*. Vol. 307. Springer, Cham. https://doi.org/10.1007/978-3-319-08964-5_4

[79] Carola Trahms, Sebastian Möller, and Jan-Niklas Voigt-Antons. 2018. Estimating Quality Ratings from Touch Interactions in Mobile Games. In *Proceedings of the 10th International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, New York, NY, USA, 1–6. https://doi.org/10.1109/qomex.2018.8463419

[80] Porfirio Tramontana, Domenico Amalfitano, Nicola Amatucci, and Anna Rita Fasolino. 2019. Automated functional testing of mobile applications: a systematic mapping study. *Software Quality Journal* 27, 1 (2019), 149–201. https://doi.org/10.1007/s11219-018-9418-6

[81] Muhammad Usman, Muhammad Zohaib Iqbal, and Muhammad Uzair Khan. 2020. An automated model-based approach for unit-level performance test generation of mobile applications. *Journal of Software: Evolution and Process* 32 (2020), e2215. https://doi.org/10.1002/smr.2215

[82] Wesley van der Lee and Sicco Verwer. 2018. Vulnerability Detection on Mobile Applications Using State Machine Inference. In *Proceedings of the European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, New York, NY, USA, 1–10. https://doi.org/10.1109/eurospw.2018.00008

[83] Mian Wan, Yuchen Jin, Ding Li, and William GJ Halfond. 2015. Detecting display energy hotspots in Android apps. In *Proceedings of the 8^{th} International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, New York, NY, USA, 1–10. https://doi.org/10.1109/ICST.2015.7102585

[84] Yingjie Wang, Guangquan Xu, Xing Liu, Weixuan Mao, Chengxiang Si, Witold Pedrycz, and Wei Wang. 2020. Identifying vulnerabilities of SSL/TLS certificate verification in Android apps with static and dynamic analysis. *Journal of Systems and Software* 167 (2020), 110609. https://doi.org/10.1016/j.jss.2020.110609

[85] Kun Yang, Jianwei Zhuge, Yongke Wang, Lujue Zhou, and Haixin Duan. 2014. IntentFuzzer: detecting capability leaks of android applications. In *Proceedings of the 9^{th} Symposium on Information, computer and communications security*. ACM, New York, NY, USA, 531–536. https://doi.org/10.1145/2590296.2590316

[86] Badamasi Imam Ya'u, Norsaremah Salleh, Azlin Nordin, Norbik Bashah Idris, Hafiza Abas, and Ali Amer Alwan. 2019. A systematic mapping study on cloud-based mobile application testing. *Journal of Information and Communication Technology* 18, 4 (2019), 485–527. https://doi.org/10.32890/jict2019.18.4.5

[87] Shengcheng Yu, Chunrong Fang, Yexiao Yun, and Yang Feng. 2021. Layout and Image Recognition Driving Cross-Platform Automated Mobile Testing. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, New York, NY, USA, 1561–1571. https://doi.org/10.1109/ICSE43902.2021.00139

[88] Noorrezam Yusop, Massila Kamalrudin, Safiah Sidek, and John Grundy. 2016. Automated support to capture and validate security requirements for mobile apps. In *Proceedings of the Asia Pacific Requirements Engineering Conference (APSEC)*. Springer, Singapore, 97–112. https://doi.org/10.1007/978-981-10-3256-1_7

[89] Samer Zein, Norsaremah Salleh, and John Grundy. 2016. A Systematic Mapping Study of Mobile Application Testing Techniques. *Journal of Systems and Software* 117, C (July 2016), 334–356. https://doi.org/10.1016/j.jss.2016.03.065

[90] Hailong Zhang, Haowei Wu, and Atanas Rountev. 2016. Automated test generation for detection of leaks in Android applications. In *Proceedings of the 11^{th} International Workshop on Automation of Software Test*. ACM, New York, NY, USA, 64–70. https://doi.org/10.1145/2896921.2896932

[91] Tao Zhang, Jerry Gao, Jing Cheng, and Tadahiro Uehara. 2015. Compatibility testing service for mobile applications. In *Proceedings of the Symposium on Service-Oriented System Engineering (SOSE)*. IEEE, New York, NY, USA, 179–186. https://doi.org/10.1109/sose.2015.35

[92] Chenyang Zhu, Zhengwei Zhu, Yunxin Xie, Wei Jiang, and Guiling Zhang. 2019. Evaluation of Machine Learning Approaches for Android Energy Bugs Detection With Revision Commits. *IEEE Access* 7 (2019), 85241–85252. https://doi.org/10.1109/access.2019.2925350

A MAPPING TABLES

Table 12. Final set of selected studies.

ID (ref)	Title	PE	S	U	P	R	C	Year	Venue
S1 [50]	A cloud-based solution for testing applications compatibility and portability on fragmented android platform				✓		✓	2019	Conference
S2 [81]	An automated model-based approach for unit-level performance test generation of mobile applications	✓						2020	Journal
S3 [7]	Androshield: automated android applications vulnerability detection, a hybrid static and dynamic analysis approach		✓					2019	Journal
S4 [70]	Appsplayground: automatic security analysis of smartphone applications		✓					2013	Conference
S5 [38]	Detecting usability problems in mobile applications on the basis of dissimilarity in user behavior			✓				2020	Journal
S6 [6]	Do memories haunt you? an automated black box testing approach for detecting memory leaks in android apps	✓						2020	Journal
S7 [79]	Estimating quality ratings from touch interactions in mobile games			✓				2018	Conference
S8 [42]	Mimic: ui compatibility testing system for android apps				✓			2019	Conference
S9 [75]	Mossot: an automated blackbox tester for single sign-on vulnerabilities in mobile applications		✓					2019	Conference
S10 [57]	NavyDroid: an efficient tool of energy inefficiency problem diagnosis for Android applications	✓						2018	Journal
S11 [24]	Performance analysis of spotify (r) for android with model-based testing	✓						2017	Journal
S12 [34]	Search-based energy testing of android	✓						2019	Conference
S13 [61]	Semi-automated usability analysis through eye tracking			✓				2018	Journal
S14 [49]	Smartphone applications usability evaluation: a hybrid model and its implementation			✓				2012	Conference
S15 [35]	μDroid: an energy-aware mutation testing framework for Android	✓						2017	Symposium
S16 [28]	An automated testing approach for inter-application security in android		✓					2014	Workshop
S17 [58]	An automatically vetting mechanism for SSL error-handling vulnerability in android hybrid Web apps		✓					2018	Journal
S18 [12]	Analysing data security requirements of android mobile banking application		✓					2018	Conference
S19 [72]	APSET, an Android aPplication SEcurity Testing tool for detecting intent-based vulnerabilities		✓					2015	Journal
S20 [88]	Automated support to capture and validate security requirements for mobile apps		✓					2016	Conference
S21 [46]	Automated usability testing for mobile applications			✓				2014	Conference
S22 [65]	Categorizing Configuration Parameters of Smartphones for Energy Performance Testing	✓						2014	Workshop
S23 [91]	Compatibility testing service for mobile applications				✓			2015	Symposium
S24 [83]	Detecting Display Energy Hotspots in Android Apps	✓						2015	Conference
S25 [29]	Dynamic detection of inter-application communication vulnerabilities in android		✓					2015	Symposium
S26 [10]	EnergyPatch: Repairing Resource Leaks to Improve Energy-Efficiency of Android Apps	✓						2017	Journal
S27 [20]	ErgoMobile: A software to support usability evaluations in mobile devices using observation techniques			✓				2017	Conference
S28 [41]	Graph-aided Directed Testing of Android Applications for Checking Runtime Privacy Behaviours		✓					2016	Workshop
S29 [13]	Mobile application usability testing in quasi-real conditions-the synergy of using different methods			✓				2018	Conference
S30 [47]	Security testing for Android mHealth apps		✓					2015	Workshop
S31 [9]	Security testing of the communication among Android applications		✓					2013	Workshop
S32 [18]	The mobile app usability inspection (MAUi) framework as a guide for minimal viable product (MVP) testing in lean development cycle			✓				2016	Conference
S33 [82]	Vulnerability Detection on Mobile Applications Using State Machine Inference		✓					2018	Symposium
S34 [21]	ZIPT: Zero-integration performance testing of mobile app designs	✓		✓				2017	Symposium
S35 [52]	Automated and unsupervised user interaction logging as basis for usability evaluation of mobile applications			✓				2012	Conference
S36 [11]	Automating Mockup-Based Usability Testing on the Mobile Device			✓				2019	Conference
S37 [1]	Characterization and Detection of Tail Energy Bugs in Smartphones	✓						2018	Journal
S38 [60]	Design and Implementation of a Toolkit for Usability Testing of Mobile Apps			✓				2013	Journal
S39 [92]	Evaluation of Machine Learning Approaches for Android Energy Bugs Detection with Revision Commits	✓						2019	Journal
S40 [25]	Extending mobile app analytics for usability test logging			✓				2017	Conference
S41 [19]	GreenOracle: Estimating Software Energy Consumption with Energy Measurement Corpora	✓						2016	Conference
S42 [43]	Performance testing of mobile applications at the unit test level	✓						2009	Conference
S43 [51]	QDroid: Mobile Application Quality Analyzer for App Market Curators	✓			✓			2016	Journal
S44 [69]	Reliability Testing using Hybrid Exploratory Basis of Tour and Fuzzy Inference System Tsukamoto					✓		2017	Conference
S45 [55]	Automated Testing of Energy Hotspots and Defects for Android Applications	✓						2019	Conference
S46 [90]	Automated Test Generation for Detection of Leaks in Android Applications	✓						2016	Workshop
S47 [84]	Identifying vulnerabilities of SSL/TLS certificate verification in Android apps with static and dynamic analysis		✓					2020	Journal
S48 [54]	AppLance: A Lightweight Approach to Detect Privacy Leak for Packed Applications		✓					2018	Conference
S49 [76]	Model Driven Engineering for Performance Testing in Mobile Applications	✓						2020	Conference
S50 [53]	Detecting and diagnosing energy issues for mobile applications	✓						2020	Symposium
S51 [56]	A compatibility testing platform for android multimedia applications				✓			2019	Journal
S52 [74]	Testing of Memory Leak in Android Applications		✓					2014	Symposium
S53 [85]	IntentFuzzer: Detecting capability leaks of android applications		✓					2014	Symposium
S54 [36]	Automated construction of energy test oracles for Android	✓						2020	Conference
S55 [26]	Automated usability tests for mobile devices through live emotions logging			✓				2015	Conference
S56 [15]	A Large-Scale Study of Application Incompatibilities in Android				✓			2019	Symposium

Table 13. Strategies and approaches adopted for NFRs testing.

ID (ref)	Black Box	White Box	Hybrid	UB	FB	SEIE	CB	IDB	MB	Technique description
S1 [50]	✓					AH				Tests are already implemented and stored. They are executed in cloud on multiple mobile devices.
S2 [81]		✓							FS	Tests are automatically generated starting from a performance model of the application under test.
S3 [7]			✓			ET	CFB			The app is tested on the fly by launching intents and API calls. Scan of reverse engineered code.
S4 [70]	✓					ET				The app is tested on the fly by exploring its UI.
S5 [38]	✓			UOH						Based on the analysis of recorded user interactions.
S6 [6]	✓			OP						Tests are composed by user events that may generate memory leaks.
S7 [79]	✓			UOH						Based on the analysis of recorded user interactions.
S8 [42]	✓			OP						Tests are composed by user events and executed on different devices.
S9 [75]			✓						FSM	Tests are generated from a FSM modeling the application UI. The FSM is inferred automatically.
S10 [57]	✓				EG	ET				The app is explored through its UI, meanwhile the performance is monitored to find inefficiencies.
S11 [24]		✓							FSM	Tests are generated from a FSM modeling the application UI.
S12 [34]			✓						FSM	Tests are generated from multiple FSMs modeling the app behavior. Models are automatically inferred.
S13 [61]	✓			UOH						Based on the analysis of recorded user interactions and eye tracking
S14 [49]	✓			UOH						Based on the analysis of recorded user interactions
S15 [35]		✓			MT					Based on a mutation testing process involving specific energy mutants
S16 [28]			✓		EG					The app is solicited by tests simulating attack behaviors. Scan of reverse engineered code.
S17 [58]			✓			ET	CFB			The app is tested on the fly by exploring its UI. Scan of reverse engineered code.
S18 [12]			✓			ET	CFB			The app is tested on the fly by exploring its UI. Scan of reverse engineered code.
S19 [72]		✓							FSM	Tests are generated from a Finite State Machine modeling the behavior of the application.
S20 [88]		✓							WM	Tests are generated from use cases and user interface models.
S21 [46]	✓			UOH						Based on the analysis of recorded user interactions.
S22 [65]	✓							EP		Test parameters are partitioned in primary parameters, causing most of the device's consumes, and stand-alone parameters.
S23 [91]	✓			OP						Tests are composed by user events and executed on different devices.
S24 [83]	✓			OP						Based on the execution of specific user events meanwhile performances and consumes are monitored, traced, logged, and analyzed.
S25 [29]			✓			ET	CFB			The app is tested on the fly meanwhile it is explored. The exploration is guided by a specific CFG.
S26 [10]			✓			ET	CFB			The app is tested on the fly meanwhile it is explored. The exploration is guided by a specific CFG.
S27 [20]	✓			UOH						Based on the analysis of recorded user interactions.
S28 [41]			✓				CFB			Tests are generated starting from a specific CFG obtained by reverse engineering.
S29 [13]	✓			UOH						Based on the analysis of recorded user interactions and eye tracking.
S30 [47]			✓		EG		CFB			The app is tested on the fly by test cases designed for finding known errors. Scan of reverse engineered code.
S31 [9]		✓			MT					Based on a mutation testing process involving specific intent mutants.
S32 [18]	✓			UOH						Based on the analysis of recorded user interactions.
S33 [82]			✓		EG				FSM	Tests are generated from a FSM modeling the application UI. The FSM is inferred automatically. Tests covers path of the FSM that may expose known vulnerabilities.
S34 [21]	✓			OP						Based on the execution of predefined user events that may cause usability errors. Performances and consumes are monitored, traced, logged, and analyzed.
S35 [52]	✓			UOH						Based on the analysis of recorded user interactions
S36 [11]	✓			UOH	EG					Based on the analysis of recorded interactions made by user events that may cause usability errors.
S37 [1]	✓			OP	EG					Based on the analysis of recorded interactions made by user events that may cause usability errors. Meanwhile, performances and consumes are monitored, traced, logged, and analyzed.
S38 [60]	✓			UOH						Based on the analysis of recorded interactions made by user events that may cause usability errors.
S39 [92]	✓				EG				FS	Based on machine learning approaches for testing energy consumption.
S40 [25]	✓			OP						Based on the analysis of recorded interactions made by known user events that may cause usability errors.
S41 [19]	✓				EG				FS	Based on machine learning approaches for testing energy consumption.
S42 [43]	✓			OP						Based on the analysis of recorded interactions made by user events that may cause usability errors. Meanwhile, performances and consumes are monitored, traced, logged, and analyzed.
S43 [51]	✓					ET				The app is tested on the fly by exploring its UI.
S44 [69]	✓					AH				Specific exploratory testing scenarios are designed and executed.
S45 [55]			✓				CFB			Tests are generated starting from a specific CFG obtained by reverse engineering.
S46 [90]			✓		EG				FSM	Tests are generated from a FSM modeling the application UI. The FSM is inferred automatically. Tests covers path of the FSM that may expose known leaks.
S47 [84]			✓		EG	ET				Potential vulnerable activities are automatically explored. The exploration is guided by the code.
S48 [54]	✓			OP		ET				The app is explored through its UI. Execution data are logged and analyzed to find vulnerabilities.
S49 [76]		✓							FS	Test cases are generated starting from Metamodels modeling performance aspects.
S50 [53]	✓			OP	EG					Based on the execution of specific user events that may cause performance issues. Performances and consumes are monitored, traced, logged, and analyzed.
S51 [56]	✓			OP						Test cases made by specific user events are designed and executed on multiple devices.
S52 [74]	✓				EG	ET				The app is tested on the fly by exploring its UI. During the exploration events that may cause known vulnerabilities are executed.
S53 [85]			✓		EG	ET				The app is tested on the fly by launching intents that may cause known vulnerabilities. Code static analysis is needed to construct the intents to be executed.
S54 [36]	✓				EG				FS	Based on machine learning approaches for testing energy consumption
S55 [26]	✓			UOH						Based on the analysis of recorded interactions made by user events that may cause usability errors.
S56 [15]	✓			OP		ET				The app is explored through its UI. Execution logs are analyzed to find portability issues.

Table 14. Supporting tools for NFRs testing of mobile apps.

ID (ref)	Tool name	TH	TG	CR	OC	CAI	T	URL (if available)
S1 [50]	NoName	✓						N/A
S2 [81]	NoName		✓					N/A
S3 [7]	AndroShield	✓	✓			✓		https://github.com/AmrAshraf/AndroShield
S4 [70]	AppsPlayground	✓						N/A
S5 [38]	NoName			✓				N/A
S6 [6]	FunesDroid	✓			✓			https://github.com/reverse-unina/FunesDroid
S8 [42]	Mimic	✓						N/A
S9 [75]	MoSSOT	✓	✓		✓			https://github.com/cuhk-mobitec/MoSSOT
S10 [57]	NavyDroid	✓					✓	N/A
S11 [24]	NoName		✓					N/A
S12 [34]	COBWEB	✓	✓					N/A
S14 [49]	UEProject	✓				✓		N/A
S15 [35]	μDroid	✓			✓	✓		https://www.ics.uci.edu/~seal/projects/mu_droid/tool.html
S17 [58]	NoName	✓				✓		N/A
S19 [72]	APSET		✓		✓			https://github.com/statops/apset
S20 [88]	MobiMEReq		✓					N/A
S21 [46]	NoName	✓			✓			N/A
S24 [83]	dLens			✓	✓			https://sites.google.com/site/dlensproject/home
S25 [29]	IntentDroid	✓				✓		https://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=an&subtype=ca&appname=gpateam&supplier=897&letternum=ENUS214-533
S26 [10]	EnergyPatch	✓				✓		https://www.comp.nus.edu.sg/~rpembed/epatch/home.html
S27 [20]	ErgoMobile			✓				N/A
S28 [41]	MAMBA		✓			✓		N/A
S31 [9]	NoName	✓			✓	✓		N/A
S33 [82]	MAT	✓	✓					https://github.com/wesleyvanderlee/AppSecurity
S34 [21]	ZIPT			✓			✓	http://www.usertesting.com/
S35 [52]	ATEBs detector			✓			✓	N/A
S36 [11]	NoName			✓	✓			N/A
S37 [1]	NoName	✓			✓			N/A
S38 [60]	NoName			✓	✓			N/A
S40 [25]	NoName			✓			✓	N/A
S42 [43]	NoName	✓			✓			N/A
S43 [51]	QDroid	✓			✓			https://github.com/leejaymin/QDroid
S45 [55]	EHDetector		✓			✓		N/A
S47 [84]	DCDroid	✓			✓	✓		N/A
S48 [54]	AppLance	✓			✓			N/A
S50 [53]	NoName	✓			✓			N/A
S51 [56]	CTP	✓					✓	N/A
S53 [85]	IntentFuzzer	✓				✓		N/A
S54 [36]	ACETON		✓					https://github.com/seal-hub/ACETON

Table 15. Publication Venues.

Venue name	**#**
Journal	
IEEE Access	3
Mobile Information Systems	2
IEEE Transactions on Software Engineering	1
Journal of Software: Evolution and Process	1
Information	1
International Journal of Human-Computer Studies	1
Science China Information Sciences	1
South African Computer Journal	1
World Wide Web	1
Journal on Software Tools for Technology Transfer	1
Mobile Networks and Applications	1
Journal of Systems and Software	1
Multimedia Tools and Applications	1
Conference	
International Conference on Software Engineering	2
Asia-Pacific Software Engineering Conference	2
Conference on Human-Computer Interaction	2
Conference on Data and application security and privacy	1
Conference on Quality of Multimedia Experience	1
Conference on Computer and Communications Security	1
Conference on Human-Centred Software Engineering	1
Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments	1
Conference on Web Information Systems and Technologies	1
Conference on Software Testing, Verification and Validation	1
Conference of Design, User Experience, and Usability	1
Conference on Human System Interaction	1
Conference in HCI and UX Indonesia	1
Conference on Advances in Mobile Computing & Multimedia	1
Conference on Green, Pervasive, and Cloud Computing	1
Working Conference on Mining Software Repositories	1
Conference on Secure Software Integration and Reliability Improvement	1
Conference on Sustainable Information Engineering and Technology	1
Conference on Energy Internet	1
Nordic Conference on Secure IT Systems	1
South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference	1
Joint European Software Engineering	1
Symposium	
International Symposium on Software Testing and Analysis	3
Joint Meeting on Foundations of Software Engineering	1
Symposium on Service-Oriented System Engineering	1
Symposium on Security and Privacy	1
Symposium on User Interface Software and Technology	1
International Symposium on High-Assurance Systems Engineering	1
Symposium on Information, Computer and Communications Security	1
Workshop	
Workshop on Automation of Software Test	5
Conference on Software Testing, Verification and Validation Workshops	1


ICMC
USP
SÃO CARLOS