



Mobile Application Testing: A Tutorial

Jerry Gao, *San Jose State University*

Xiaoying Bai, *Tsinghua University*

Wei-Tek Tsai, *Arizona State University*

Tadahiro Uehara, *Fujitsu Laboratories*

To cope with frequent upgrades of mobile devices and technologies, engineers need a reusable and cost-effective environment for testing mobile applications and an elastic infrastructure to support large-scale test automation.

According to a recent Clearwater Technology Team Report, mobile computing industry profits are expected to top US \$330 billion by 2015 (www.clearwatercf.com/documents/library/Mobile_Report_FINAL.pdf). The explosive demand for mobile devices—primarily in the form of smartphones and tablets—and both native and Web applications for such devices has been accompanied by a need for more and better mobile application testing tools.¹ ABI Research predicts that revenue from such tools, which today are primarily manual but are becoming increasingly automated, will rise from \$200 million in 2012 to \$800 million by 2017 (<https://www.abiresearch.com/press/200-million-mobile-application-testing-market-boos>).

Most research on mobile application testing has focused on solutions to specific technical problems in the following areas:

- white-box and unit testing of mobile apps,
- black-box and GUI testing of mobile apps,

- validation of mobile app quality-of-service (QoS) requirements,
- wireless network testing,
- mobile usability testing, and
- mobile test automation frameworks.

See the sidebar, “Related Work in Mobile Application Testing” for more information.

In contrast, our goal here is to provide a general tutorial on mobile application testing that first examines testing requirements and then looks at current approaches for both native and Web apps for mobile devices.

MOBILE APPLICATION TESTING

The term *mobile testing* refers to different types of testing, such as native mobile app testing, mobile device testing, and mobile Web app testing. We use *mobile app testing* to refer to “testing activities for native and Web applications on mobile devices using well-defined software test methods and tools to ensure quality in functions, behaviors, performance, and quality of service, as well as features, such as mobility, usability, interoperability, connectivity, security, and privacy.”

REQUIREMENTS

Several unique requirements distinguish mobile application testing from conventional software testing. First, mobile apps must function correctly anytime, anywhere. In addition, because mobile services are often developed for a set of select devices, apps must work properly across

Related Work in Mobile Application Testing

Many studies address different issues and topics in mobile application testing. Due to space constraints, we provide a high-level view of recent mobile testing work.

White-box testing

Existing white-box testing methods are still applicable to mobile apps. For example, Java Pathfinder¹ is a mobile program verification tool supporting white-box mobile Java program testing. Engineers can use this tool to detect race conditions and deadlocks based on UML state charts and symbolic execution. Riyadh Mahmood and his colleagues² use a white-box approach to generate test cases with two program-based models (call graph and architectural) to achieve mobile program code coverage.

Black-box testing

Many black-box testing techniques are useful in mobile app testing. Random testing and the scenario-based testing method³ are good examples. GUI-based testing has been discussed in numerous papers. For instance, Saswat Anand and his colleagues⁴ introduced an automated testing approach to validating mobile GUI event sequences for smartphone apps. Similarly, Domenico Amalfitano and his colleagues⁵ presented a tool called AndroidRipper that uses an automated GUI-based technique to test Android apps in a structured manner.

Usability testing

Usability testing helps enhance the quality of the user experience on mobile devices. Anne Kaikkonen and her colleagues⁶ present a usability testing study and comparative findings in the laboratory as well as in the field.

Quality-of-service testing

The QoS requirements for mobile apps include software performance, reliability, availability, scalability, and loading speed. Rabeb Mizouni and his colleagues⁷ evaluated the Web service performance of handheld resource-constrained clients using SOAP and RESTful technologies. Their focused QoS parameters included response time, availability, throughput, and scalability.

Wireless connectivity testing

Today, mobile devices support diverse wireless connectivity options, so mobile apps must be validated with specified wireless connectivity and contexts. Tapani Puhakka and Marko Palola⁸ addressed this need for B3G applications and presented an experimental system for automated testing of B3G mobile apps on multiple mobile phones at the same time. Ichiro Satoh⁹ presented a new approach, called the flying emulator, to test the software executed on mobile terminals. Unlike existing approaches, it lets engineers construct emulators as mobile agents that can travel between computers.

Mobile test automation frameworks

Certain research efforts are dedicated to developing tools or frameworks to address limitations in current commercial tools. For example, JPF-ANDROID¹ is a verification tool that supports white-box testing for mobile apps, and JeBUTi/ME² is a tool that offers white-box test coverage analysis based on a conventional CFG-based test model.

A few recent research papers focus on GUI-based testing using scripts and GUI event-flow models.^{4,5} In addition, other work on integrated test automation frameworks¹⁰ has proposed research tools at the system level that support mobile testing on multiple heterogeneous platforms. Another example is MoViT,¹¹ a distributed software suite for emulating mobile wireless networks. MoViT provides users with a virtualized environment that helps researchers develop and test mobile apps and protocols for any hardware and software platform that can be virtualized.

References

1. H. van der Merwe et al., "Verifying Android Applications Using Java Pathfinder," *ACM SIGSOFT Software Eng. Notes*, vol. 37, no. 6, 2012, pp. 1–5.
2. R. Mahmood et al., "A Whitebox Approach for Automated Security Testing of Android Applications on the Cloud," *Proc. 7th Int'l Workshop Automation of Software Test (AST 12)*, 2012, pp. 22–28.
3. J. Bo et al., "MobileTest: A Tool Supporting Automatic Black Box Test for Software on Smart Mobile Devices," *Proc. 2nd Int'l Workshop on Automation of Software Test (AST 07)*, 2007, pp. 8–14.
4. S. Anand et al., "Automated Concolic Testing of Smartphone Apps," *Proc. ACM SIGSOFT 20th Int'l Symp. Foundations of Software Eng. (FSE 12)*, 2012, pp. 1–11.
5. D. Amalfitano et al., "Using GUI Ripping for Automated Testing of Android Applications," *Proc. 27th IEEE/ACM Int'l Conf. Automated Software Eng. (ASE 12)*, 2012, pp. 258–261.
6. T. Kallio and A. Kaikkonen, "Usability Testing of Mobile Applications: A Comparison between Laboratory and Field Testing," *J. Usability Studies*, vol. 1, no. 1, 2005, pp. 4–16.
7. R. Mizouni et al., "Performance Evaluation of Mobile Web Services," *Proc. 9th IEEE European Conf. Web Services (ECOWS 11)*, 2011, pp. 184–191.
8. T. Puhakka and M. Palola, "Towards Automating Testing of Communicational B3G Applications," *Proc. 3rd Int'l Conf. Mobile Technology, Applications & Systems*, 2006, article no. 27, pp. 1–6.
9. I. Satoh, "Software Testing for Wireless Mobile Computing," *IEEE Wireless Comm.*, vol. 11, no. 5, 2004, pp. 58–64.
10. H. Song et al., "An Integrated Test Automation Framework for Testing on Heterogeneous Mobile Platforms," *Proc. 1st ACIS Int'l Symp. Software and Network Eng.*, 2011, pp. 141–145.
11. E. Giordano et al., "MoViT: The Mobile Network Virtualized Testbed," *Proc. 9th ACM Int'l Workshop Vehicular Inter-networking, Systems, and Applications*, 2012, pp. 3–12.

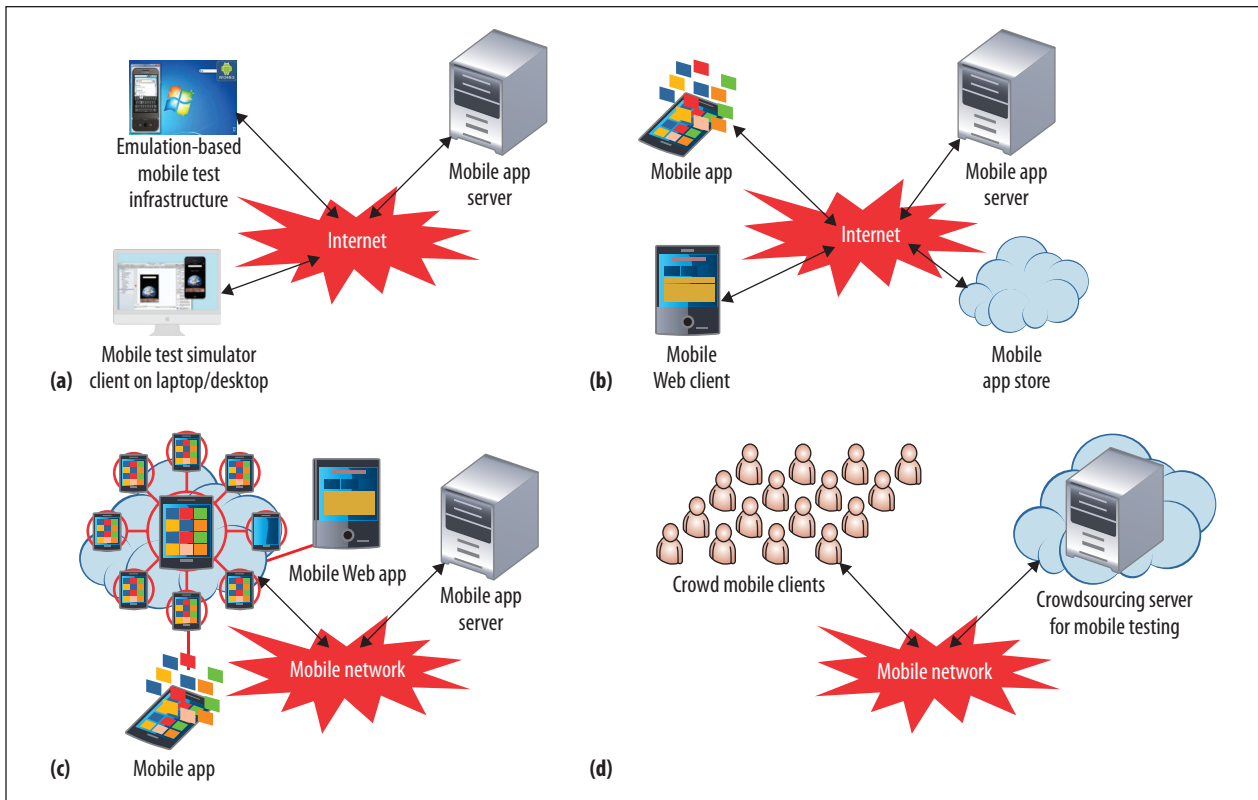


Figure 1. Different mobile test infrastructures: (a) emulation, (b) cloud, (c) device, and (d) crowd based.

platforms that have, for example, different operating systems, display sizes, compute power resources, and battery life. Moreover, to provide the rich experience mobile users have come to expect, mobile apps must include multiple input channels (keyboard, voice, and gestures), multimedia support, and other enhanced usability features. Also, large-scale simulation and virtualization are required to keep hardware costs low. Finally, because most mobile service plans support a range of wireless networks (2G, 3G, 4G, Wi-Fi, WiMax), mobile apps must function in diverse network connectivity contexts.

Testing activities and goals

Given these requirements, mobile app testing tends to focus on the following activities and goals:

- *functionality and behavior testing*—activities that validate service functions, mobile Web APIs, external system behaviors, system-based intelligence, and user interfaces (UIs);
- *QoS testing*—activities that evaluate system load, performance, reliability/availability, scalability, and throughput;
- *interoperability testing*—activities that check system interoperability across different devices, platforms, browsers, and wireless networks;

- *usability and internationalization testing*—activities that assess UI content and alerts, user operation flows and scenarios, media richness, and gesture interaction support;
- *security and privacy testing*—activities that check user authentication, device security, session security, system/network penetration, peer-to-peer (P2P) mobile communications security, end-to-end transaction security, and user privacy;
- *mobility testing*—activities that validate location-based functions, user profiles, system data, and user data;
- *compatibility and connectivity testing*—activities that assess mobile browser and platform compatibility and diverse wireless network connectivity; and
- *multitenancy testing*—activities that validate tenant-based functions, system behaviors, system data, and UIs.

Several strategies have evolved to accommodate this mobile app testing environment.

Testing approaches

We have identified four popular mobile app testing approaches, based on the underlying client-server infrastructure. Figure 1 illustrates the different infrastructures. Table 1 compares these approaches for different testing activities.

Table 1. Mobile application testing approaches.

Testing		Emulation-based testing	Device-based testing	Cloud testing	Crowd-based testing
Functionality and behavior	Function features	Single emulator or simulator-based	Device-based single client	Large-scale device-based testing	Ad hoc or managed testing
	Mobile user operations	Yes	Yes	Yes	Yes
	Mobile gestures	Limited	Yes	Yes	Yes
Quality of service	Load testing	Limited scale	Limited scale	Large scale	Ad hoc scale
	Performance testing	Function-based	Single client	Large scale	Ad hoc scale
	Reliability/availability	Single client	Single client	Large scale	Ad hoc scale
	Scalability	No	No	Yes	Yes
Interoperability	Crosses devices	No	No	Yes	No
	Crosses platforms	No	Yes	Yes	No
	Crosses browsers	Yes	Yes	Yes	Yes
	Crosses networks	Limited	Limited	Yes	Limited
Usability and internationalization	Internationalization of mobile user operation	High cost and manual	High cost and manual	Automation enabled	Low cost and manual
Security and privacy	User security and privacy	Limited	Limited	Large scale	Ad hoc scale
	Communication security	No	Yes	Yes	Yes
	Transaction security	Yes	Yes	Yes	Yes
	Session security	Yes	Yes	Yes	Yes
	Server security	Limited	Yes	Yes	Yes
Mobility	Location-based function and behaviors	Based on simulated location	Preconfigured location	Based on configured mobile locations	Based on user locations
	Location-based user data and profile	Simulated user profile and data	Single user profile and data	Large-scale user profile and data	Crowd user profile and data
Compatibility and connectivity	Browser compatability	Single mobile browser	Single mobile browser	Configurable browsers on different devices	Browsers on various user devices
	Network connectivity	No	Singled network connectivity	Diverse connectivity	Preconfigured connectivity
	Platform compatability	Single platform	Any	Any	Any
Multitenancy	Tenant-based functions and behaviors	Yes	Yes	Yes	Yes
	Tenant-based QoS	Limited scale	Limited scale	Large scale	Ad hoc scale
	Tenant-based interfaces	Yes	Yes	Yes	Yes

Emulation-based testing. The emulation-based testing approach involves using a mobile device emulator (also known as a device simulator), which creates a virtual machine version of a mobile device for study on a personal computer. It is often included with a mobile platform's software development kit (such as Android SDK). It is relatively inexpensive because no testing laboratory is needed and no physical devices have to be purchased or rented, but it can only be used to assess

Web app testing aims to validate the quality of mobile Web apps using different Web browsers on diverse mobile devices.

system functionality in very limited contexts. Although this approach is low-cost, it has several limitations—for example, it has difficulty validating a full set of gestures because most emulators support very limited gestures and device-specific functions. Another challenge is its limited scale for testing QoS.

To overcome these problems, a simulation-based approach can create a mobile test simulator to mimic various mobile client operations (such as diverse gestures) and support more than one mobile client. However, even this workaround has limitations in validating device-specific mobile service functions. In addition, it is impossible to deal with diverse devices and mobile platforms/browsers because emulators are usually based on a specific device or platform.

Device-based testing. The device-based testing approach requires setting up a testing laboratory and purchasing real mobile devices, which is more costly than emulation-based approaches but can verify device-based functions, behaviors, and QoS parameters that other approaches cannot. In addition, it also has the advantage of being able to validate its underlying mobile networks via reconfigurations and selections in a testing environment. One of the major challenges with this approach is the problem it has in coping with rapid changes in mobile devices and platforms. Another challenge is its limitations related to system QoS because large-scale tests require many mobile devices, which is usually impossible for enterprises.

Cloud testing. The approach based on testing through the cloud is typically supported by testing vendors (such as www.nttdata.com/global/en). In a panel presentation at the Fifth International Workshop on Software Testing in the Cloud, Raj Rao, VP of software quality at NTT DATA, reported on his company's device-based cloud testing approach. The basic idea is to build a mobile device cloud that can support testing services on a large scale. This approach addresses the significant increase in demand for

mobile testing services by using a pay-as-you-go business model. It also allows different mobile users to provision their required testing environments via a rental service model. Compared with other approaches, this can be more cost-effective than device-based testing for large-scale applications, and it is much more effective for supporting diverse testing activities on mobile devices.

Crowd-based testing. The crowd-based testing approach involves using freelance or contracted testing engineers or a community of end users such as uTest (www.utest.com), along with a crowd-based testing infrastructure and a service management server to support diverse users. Currently, a service vendor supports primitive test management, a testing service, and bug reporting. Most mobile test operations are managed in an ad hoc way with very limited mobile test automation tools. This approach offers the benefits of in-the-wild testing without the need to invest in a lab or purchase or rent devices, but at the risk of low testing quality and an uncertain validation schedule.

NATIVE VERSUS WEB APPLICATION TESTING

Two types of mobile application software can undergo testing. *Native* apps are deployed and executed on mobile devices and usually depend on native APIs and dongles, such as camera and GPS APIs. *Web* apps consist of an app server and client software executed over Web browsers, through which users can access application services. Testing of the two types must take their differences into account.

Primary testing objectives

Native app testing aims to validate the quality of mobile apps downloaded and executed on select mobile platforms on different mobile devices. Testing focuses on functionality and behavior (including device-specific functions such as gesture interaction), QoS requirements, usability, security, and privacy.

Web app testing aims to validate the quality of mobile Web apps using different Web browsers on diverse mobile devices. Unlike native apps, Web apps usually provide users with a thin mobile client to access online functions from the back-end server. Thus, in addition to functionality and behavior, QoS requirements, usability, security, and privacy, mobile Web app testing focuses on connectivity and interoperability.

Testing environment

For apps native to mobile devices, the underlying mobile platform or operating system constitutes the testing environment. To achieve effective automation, testing solutions should be compatible, deployable, and executable on multiple platforms.

For mobile Web apps, the underlying Web browser is the testing environment.

User interfaces

UI testing for mobile native apps considers fat or smart mobile clients, rich media content and graphics, and gesture features.

For mobile Web apps, UI testing considers Web-based thin mobile clients, downloadable mobile clients, and browser-based rich media and graphics support.

Technology for mobile test automation

Technologies that enable automated testing of mobile native apps include programming languages such as Java (Android), Objective-C (iOS), and Visual C++ (Windows Mobile); standardized SDKs and development tools from device vendors; and application development platforms.

For mobile Web apps, test automation technologies are generally found online in the form of HTML5, CSS3, and JavaScript. The app developer might also use server-side languages or frameworks such as PHP, Rails, and Python.

Mobile connectivity testing

Diverse mobile wireless networks support the connectivity needs of both native and Web apps. Because connectivity affects application performance and interoperability, engineers must pay attention to it during testing.

For mobile native apps, this means considering online content synchronization during testing as well as download, deployment, and customization issues related to app stores and marketplaces.

For mobile Web apps, it is important to test with Internet connectivity and diverse wireless air protocols in mind, as well as the mobile connections between clients and servers.

Usability testing

Validating a mobile native app generally involves testing mobile device-based gestures, content, interfaces, and the general user experience—for example, how the user interacts with the camera, GPS, or a fingerprint sensor.

In contrast, usability testing for mobile Web apps typically focuses on Web-based GUI content, interfaces, and user operation flows. For example, a mobile travel app such as Dwellable (www.dwellable.com) supports travel information and related content on mobile browsers in different languages based on user location. Validating such mobile application needs mobile usability testing to assure the quality of mobile Web content as well as its presentation formats, styles, and languages.

Mobility testing

Mobility testing on a native device usually involves testing the device's location-based functions, features, data, profiles, and API. For example, a mobile travel app's content should be delivered and presented to users based on their current location; this would include airport

information, rental service offices, maps, and attraction points and related data. If the device cannot accept that data, the testing engineer needs to know that.

In contrast, testing mobility for mobile Web apps focuses on testing the quality of location-based system functions, data, and behaviors.

Mobile native features

Because apps depend on native mobile platforms and devices, they can be developed to support certain functions based on native APIs, such as credit card transactions and fingerprint validation. These are specific focuses in native mobile app testing; they generally are not tested for in mobile Web apps because they are so device specific. For example, testing digital game software on a tablet involves validating motion-based operations and interactions. The additional hardware components and APIs equipped to support the game software—such as motion-based operations based on a particular device—are an example of mobile native features.

For mobile Web apps, it is important to test with Internet connectivity and diverse wireless air protocols in mind.

STATE-OF-THE-ART TOOLS AND PROCESSES

Figure 2a shows a test process for mobile native apps based on our engineering experience and observations. The key is to assure the quality of functions and behaviors as well as QoS parameters on a mobile device:

- Step 1, component testing, includes both black- and white-box testing as well as native API interactions.
- Step 2, function testing, focuses on validating functions, GUI-based scenarios, and native behaviors, such as gesture testing on mobile clients.
- Step 3, QoS testing, checks QoS attributes, including performance, reliability, availability, and security.
- Step 4, feature testing, validates network connectivity, compatibility, interoperability, mobility, and usability
- Step 5, service testing, looks at mobile app services, including download, installation, deployment, service security, and synchronization.

Similarly, Figure 2b shows a test process for mobile Web apps. The key objective here is to assure the quality of the Web system under test on networks via mobile browsers:

- Step 1, component testing, validates the quality of software components in mobile Web clients and the related server components using black- and white-box testing methods.

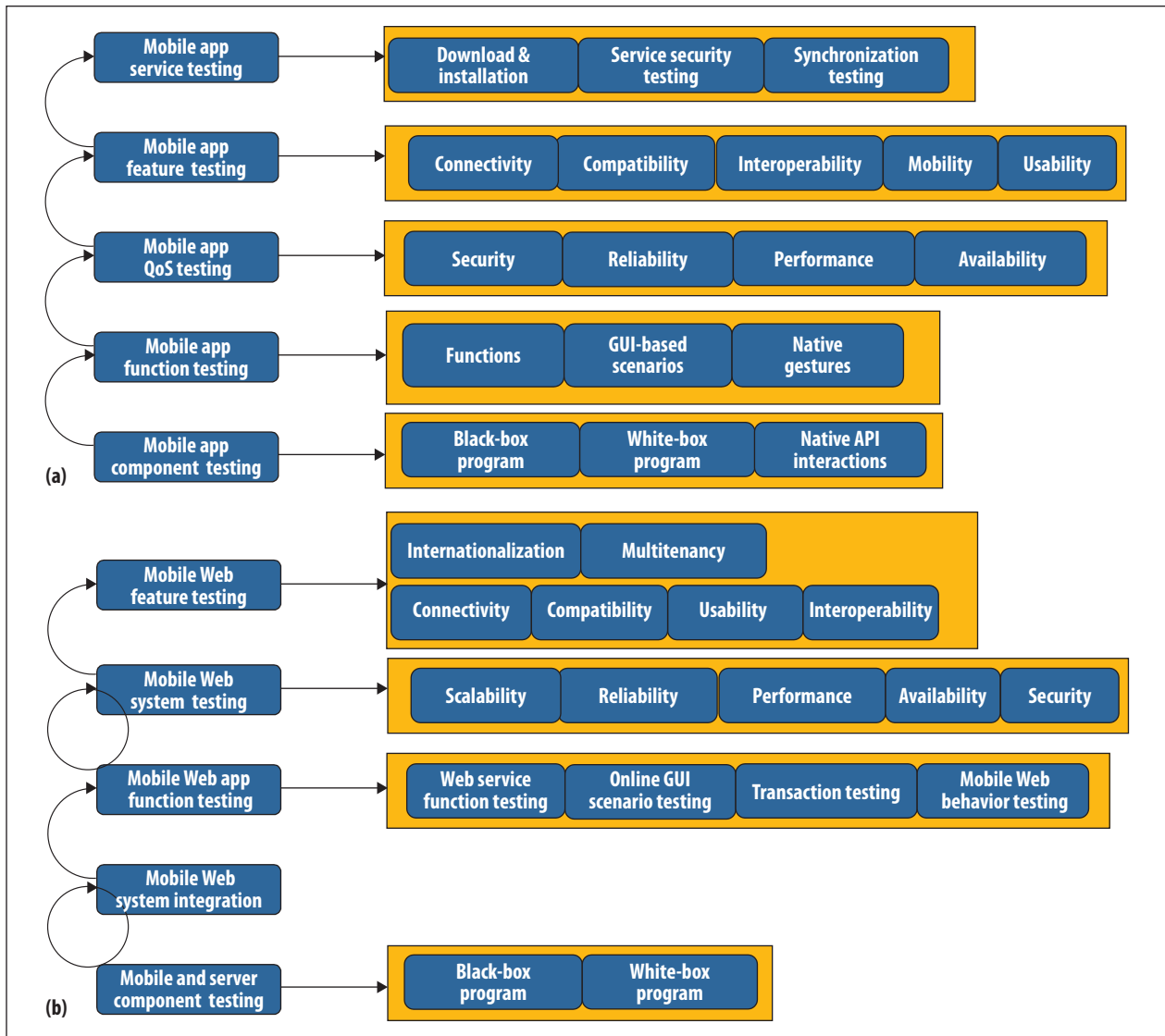


Figure 2. Test processes for mobile application testing. The test processes for (a) mobile native applications and (b) mobile Web applications differ slightly, but attempt to reach the same goals.

- Step 2, system integration, focuses on component integration with the system (mobile client and server).
- Step 3, function testing, targets mobile Web service function quality, end-to-end business transactions, Web GUI scenarios, and Web-based mobile behaviors and gestures.
- Step 4, system testing, considers QoS requirements, including end-to-end system performance, load, reliability, availability, security, and scalability.
- Step 5, feature testing, targets quality, such as mobile connectivity, compatibility, usability, interoperability, security, and internationalization.

Table 2 lists several popular commercial and open source mobile testing tools.

Most tools in the table were developed to support GUI-based functional testing, except NeoLoad and SandStorm, which were designed for load and performance testing. Keynote's MITE can also be used for performance testing on mobile devices.

Windows, Linux, and Mac are the most popular platforms for mobile testing tools, and most tools support multiple platforms, with very few exceptions—for example, Keynote's MITE only supports the Windows platform.

Most of the listed tools support both native and mobile Web app testing as well, again with a few exceptions. Keynote's MITE and Selenium's Android webDriver, for example, only support testing for mobile Web apps, while Selendroid, Appium, and Calabash are designed to support just native app testing.

Table 2. A comparison of mobile testing tools.

Attributes	MITE	MonkeyTalk	seeTest Mobile	NeoLoad	SandStorm	MobileCloud	Sikuli	Calabash	eggPlant	MonkeyRunner	Robotium	Dollop	Selenium Android WebDriver	Appium	Selendroid
GUI-based function testing	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Performance testing	Yes	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No
Load testing	No	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No
Linux	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Windows	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Mac	No	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Android OS	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
iOS	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
Windows OS	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No	No	Yes	Yes
Testing for mobile Web apps	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	No	Yes
Testing for mobile native apps	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Emulation-based testing	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Device-based testing	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Supported script language	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Record and replay	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	No	Yes	Yes	No	No
Open source	No	Yes	No	No	No	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
License/subscription	Yes	No	Yes	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes	No	No

Most of the listed tools support both emulation- and device-based mobile testing, once again with some exceptions, such as eggPlant and MonkeyRunner, which are designed for emulation-based mobile testing, and MobileCloud from QTP, which only supports device-based testing for native mobile apps.

All of the listed tools provide certain scripting features, and many of them support different languages

and technologies, such as Java, Python, and Jython. Some tools, such as MITE and MonkeyTalk, only support JavaScript.

Many of tools require subscription charges or license contracts, but several of the open source options are useful for building in-house mobile testing services. Appium, Selendroid, and Calabash exemplify this. In addition, numerous tools are designed for validating mobile

website content and performance—a good example is Gomez (www.gomez.com/website-performance-test).

However, today's tools also have several major limitations. First, they lack a unified automation infrastructure to support mobile testing on different platforms and browsers. Second, no tools support large-scale concurrent mobile test operations for system scalability testing. Third, there is a lack of standardized scripting solutions to support integration and interoperation among different tools.

To cope with the frequent upgrades of mobile devices and technologies, engineers need a reusable and cost-effective environment for testing on mobile devices.

ISSUES, CHALLENGES, AND NEEDS

Despite its growing importance, the field of mobile app testing continues to face numerous issues, challenges, and needs.

Test environments

According to feedback from test engineers, construction of mobile test environments still involves high costs and levels of complexity. Setting up a mobile test environment for multiple apps on each mobile platform for a range of devices is tedious, time-consuming, and expensive, and frequent upgrades in both device and platform spaces only exacerbate this challenge.

This brings up a key need in the mobile app testing field—a reusable and systematic approach to test environment setup and configuration that covers a diverse range of mobile devices and platforms. This solution could eliminate the tedious manual operations in place today and reduce costs as well. The ideal reusable environment would have the following characteristics:

- unified connectivity to different mobile platforms on mobile devices;
- a configuration capability that supports the systematic deployment, installation, and execution of a given app on different platforms for specific mobile devices; and
- diverse mobile network configuration options.

Mobile Web app testing requires validation with different browsers and Web technologies, which means test engineers need to evaluate different QoS parameters as well. This type of testing requires a way to generate and simulate large-scale mobile application requests and interactions with real or virtual test infrastructures and

solutions. Accordingly, engineers need to be able to select and configure diverse wireless network connectivity options as seamlessly as possible.

Standards, modeling, and coverage criteria

Today's software testing standards, such as ISO/IEC/IEEE 29119 (www.softwaretestingstandard.org) and ISO12207 (www.12207.com/test1.htm), provide well-defined guidelines for software testers regarding test organizations, processes, and documentation.

Existing test models and coverage criteria can help address mobile program structures, dynamic behaviors, and GUI operation flows, but we still need standards, test models, and coverage criteria to address the distinct requirements of mobile app testing.


In addition, the diverse mobile environments require a special model to address test coverage, taking into account underlying operational contexts that include a variety of mobile devices, platforms, browsers, and native APIs.

Test automation

Automated mobile app testing raises two further issues: the lack of standardization in mobile test infrastructure, scripting languages, and connectivity protocols between mobile test tools and platforms; and the lack of a unified test automation infrastructure and solutions that cross platforms and browsers on most mobile devices.

This goes back to the test environment challenge as well—to cope with the frequent upgrades of mobile devices and technologies, engineers need a reusable and cost-effective environment for testing on mobile devices, as well as an elastic infrastructure to support large-scale test automation. Ultimately, this will require device-based mobile test clouds equipped with connected, diverse, and replaceable mobile devices; scalable emulation clouds that can enable the creation, deployment, and control of large-scale mobile emulation testing; and a unified test control and execution solution that supports concurrent and large-scale test automation.

According to Juniper Research's latest study (www.marketingcharts.com/direct/cloud-based-mobile-market-to-grow-88-12043), the market for cloud-based mobile apps will grow 88 percent from 2009 to 2014, bringing an even stronger demand for mobile test automation solutions that can cope with the issues and challenges we described earlier. Cloud-based mobile testing offers a promising and cost-effective way to meet diverse mobile testing needs. We predict two different trends in mobile testing services: shared mobile device clouds set up by testing service vendors to support various testing needs, and crowd-based testing services that focus on usability testing and internalization. This field will only

continue to grow, and it is encouraging to see that the cloud might offer additional ways to meet its challenges. 

Reference

1. J. Harty, *A Practical Guide to Testing Wireless Smartphone Applications*, Morgan & Claypool, 2009.

Jerry Gao is a professor in the Department of Computer Engineering at San Jose State University. His research areas include cloud computing, testing-as-a-service (TaaS), mobile computing, and software testing and automation. Gao received a PhD in computer science and engineering from the University of Texas at Arlington. Contact him at jerry.gao@sjsu.edu.

Xiaoying Bai is an associate professor in the Department of Computer Science and Technology at Tsinghua University, China. Her research interests include model-driven testing and test automation techniques in various software paradigms such as distributed computing, service-oriented architecture, cloud computing, and embedded systems. Bai received a PhD in computer science from Arizona State University. Contact her at baixy@tsinghua.edu.cn.

Wei-Tek Tsai is a professor of computer science and engineering in the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University. His recent research work is related to service and cloud computing. Tsai received a PhD in computer science from the University of California, Berkeley. He was also a visiting professor at Polytechnic University of Bucharest, Romania, as well as a guest professor at Tsinghua University, Wuhan University, Northwest University, and Harbin Institute of Technology in China. Contact him at wtsai7@gmail.com.

Tadahiro Uehara works at Fujitsu Laboratories, Japan. His research interests include software engineering, especially object-oriented technologies and testing technologies for business applications. Uehara received an ME in intelligent science from the Tokyo Institute of Technology. Contact him at uehara.tadahiro@jp.fujitsu.com.



Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.

IEEE computer society

PURPOSE: The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field.

MEMBERSHIP: Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEBSITE: www.computer.org

Next Board Meeting: 4–7 February 2014, Long Beach, Calif., USA

EXECUTIVE COMMITTEE

President: Dejan S. Milošević

President-Elect: Thomas M. Conte; **Past President:** David Alan Grier;

Secretary: David S. Ebert; **Treasurer:** Charlene ("Chuck") J. Walrad; **VP,**

Educational Activities: Phillip Laplante; **VP, Member & Geographic**

Activities: Elizabeth L. Burd; **VP, Publications:** Jean-Luc Gaudiot; **VP,**

Professional Activities: Donald F. Shafer; **VP, Standards Activities:** James

W. Moore; **VP, Technical & Conference Activities:** Cecilia Metra; **2014**

IEEE Director & Delegate Division VIII: Roger U. Fujii; **2014 IEEE Director**

& Delegate Division V: Susan K. (Kathy) Land; **2014 IEEE Director-Elect &**

Delegate Division VIII: John W. Walz

BOARD OF GOVERNORS

Term Expiring 2014: Jose Ignacio Castillo Velazquez, David. S. Ebert, Hakan Erdogmus, Gargi Keeni, Fabrizio Lombardi, Hironori Kasahara, Arnold N. Pears

Term Expiring 2015: Ann DeMarle, Cecilia Metra, Nita Patel, Diomidis Spinellis, Phillip Laplante, Jean-Luc Gaudiot, Stefano Zanero

Term Expiring 2016: David A. Bader, Pierre Bourque, Dennis Frailey, Jill I. Gostin, Atsuhiko Goto, Rob Reilly, Christina M. Schober

EXECUTIVE STAFF

Executive Director: Angela R. Burgess; **Associate Executive Director & Director, Governance:** Anne Marie Kelly; **Director, Finance & Accounting:** John Miller; **Director, Information Technology & Services:** Ray Kahn; **Director, Membership Development:** Eric Berkowitz; **Director, Products & Services:** Evan Butterfield; **Director, Sales & Marketing:** Chris Jensen

COMPUTER SOCIETY OFFICES

Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036-4928

Phone: +1 202 371 0101 • **Fax:** +1 202 728 9614

Email: hq.ofc@computer.org

Los Alamitos: 10662 Los Vaqueros Circle, Los Alamitos, CA 90720

Phone: +1 714 821 8380 • **Email:** help@computer.org

MEMBERSHIP & PUBLICATION ORDERS

Phone: +1 800 272 6657 • **Fax:** +1 714 821 4641 • **Email:** help@computer.org

Asia/Pacific: Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan • **Phone:** +81 3 3408 3118 • **Fax:** +81 3 3408 3553 •

Email: tokyo.ofc@computer.org

IEEE BOARD OF DIRECTORS

President: J. Roberto de Marca; **President-Elect:** Howard E. Michel; **Past**

President: Peter W. Staecker; **Secretary:** Marko Delimar; **Treasurer:**

John T. Barr; **Director & President, IEEE-USA:** Gary L. Blank; **Director**

& President, Standards Association: Karen Bartleson; **Director & VP,**

Educational Activities: Saurabh Sinha; **Director & VP, Membership and**

Geographic Activities: Ralph M. Ford; **Director & VP, Publication Services**

and Products: Gianluca Setti; **Director & VP, Technical Activities:** Jacek

M. Zurada; **Director & Delegate Division V:** Susan K. (Kathy) Land;

Director & Delegate Division VIII: Roger U. Fujii

revised 17 Dec. 2013

