© 2015 by the authors; licensee RonPub, Lübeck, Germany. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/3.0/).

Open Access

Open Journal of Web Technologies (OJWT)
Volume 2, Issue 1, 2015
http://www.ronpub.com/ojwt
ISSN 2199-188X

Context-Dependent Testing of Applications for Mobile Devices

Tim A. Majchrzak [A], Matthias Schulte [B]

[A] University of Agder, Gimlemoen 25, 4630 Kristiansand, Norway, tima@ercis.de
[B] viadee Unternehmensberatung GmbH, Anton-Bruchausen-Straße 8, 48147 Münster, Germany, Matthias.Schulte@viadee.de

Abstract

Applications propel the versatility of mobile devices. Apps enable the realization of new ideas and greatly contribute to the proliferation of mobile computing. Unfortunately, software quality of apps often is low. This at least partly can be attributed to problems with testing them. However, it is not a lack of techniques or tools that make app testing cumbersome. Rather, frequent context changes have to be dealt with. Mobile devices most notably move: network parameters such as latency and usable bandwidth change, along with data read from sensors such as GPS coordinates. Additionally, usage patterns vary. To address context changes in testing, we propose a novel concept. It is based on identifying blocks of code between which context changes are possible. It helps to greatly reduce complexity. Besides introducing our concept, we present a use case, show its application and benefits, and discuss challenges.

Type of Paper and Keywords

Short Communications: *App, mobile, mobile app, mobile device, test, testing, context*

1 Introduction

It is hard to believe that the first iPhone was introduced only a few years ago (in 2007 [31], to be precise). Contrasting former approaches such as personal digital assistants (PDAs) and feature-phones, smartphones and tablet computers have become devices used by nearly everyone. Their popularity among consumers and enterprises is still rising [16, 17]. Unsurprisingly, companies embrace the new possibilities for a variety of activities [33].

Modern mobile devices challenge the performance offered by personal computers shipped few years ago. Moreover, they are equipped with special hardware such as cameras and a multitude of sensors. However, their versatility is propelled by the software they use – or rather that make use of *them*. Mobile applications – *apps* – nowadays form an ecosystem of their own. They draw greatly from the possibilities offered by the devices, for example by using an integrated camera or making use of localization via GPS.

Developing apps is a relatively new practice with little experience and even less *tradition*. While there are already many textbooks, scientific literature is limited to articles from conference proceedings and – to a smaller extent – journals. Work typically tackles specific issues rather than giving a bigger picture. While app development is not too different from developing rich-client applications or Web applications for PCs [23], there are particularities.

Among others, *software testing* is one of these. Testing software is a cumbersome task [32]. Many techniques require sophistication. Effective and efficient software testing remains a challenge despite decades of research. At the same time, proper testing greatly con-

tributes to an application's value. Insights from software engineering including testing strategies and techniques can be applied to apps. However, testing differs in a number of ways: Firstly, apps are not developed on the platform they run (mobile device) but on a PC. Testing on emulators will not yield the same results as testing natively. Moreover, it can be inefficient. Secondly, testing on mobile devices is laborious and very hard to automate. Additionally, it is tough to ensure uniform conditions during testing. Thirdly, tool support currently is limited. Not all PC tools have mobile device counterparts. Fourthly, many apps combine various technologies and even programming paradigms and languages. Most apps contain some mixture of native programming and Web technology.

The most profound difference, however, is *context*. Mobile devices are subject to many different contexts; the simplest one is location, since mobility typically means (very) frequent slight changes of position. There are many further context changes such as network condition, availability of data from sensors, and even social issues such as alternating users on one device. Therefore, devices need not only be tested *as they are* but taking context into considerations. Our experience is that testing results greatly differ in dependence of the context.

We present a novel approach for software testing of apps that takes into account the context changes inherent to mobile devices. Our article makes a number of contributions. Firstly, we explain the background of context as an influencing factor of apps – it has implications beyond the realm of testing. Secondly, we introduce our unique approach for handling context in testing. Thirdly, we facilitate employing our approach by showing a real-life scenario. And fourthly, we generalize our findings and discuss the next steps.

This paper is structured as follows. Section 2 highlights the relevance of context changes for mobile device usage. In Section 3 we distinguish our work from other mobile testing approaches. Section 4 explains our approach in detail, first with a focus on theory and then by presenting a scenario. Application, Limitations and Challenges are discussed in Section 5. Finally, we draw a conclusion in Section 6.

This article is an greatly extended and revised version of paper [34]. It extends the concepts proposed by us in [47].

2 Mobile Devices and Context

2.1 Context in General

The usage paradigm of apps contrasts that of applications on PCs and that of Web sites (i.e. Webapps with no optimization). The main difference is *mobility*. Even if an app not explicitly takes notice of mobility (say, a common game) it is influenced by it: mobility might lead to changing conditions such as varying connectivity. Moreover, app lifecycle models differ from that of applications for PCs in that they might be halted on external events such as incoming phone calls.[1] Apps are built to be used in different situations and in a changing environment. Various kinds of devices are employed. Besides smartphones and tablets, there are hybrids such as the so called *phablets* [48] and devices such as smart TVs [21].

The above considerations lead to an insight: Apps are heavily influenced by the *context* they reside in. At the same time, apps are capable of making use of contextual factors. To give an example: when you drive out of town your smartphone might need to switch to a cellular system providing less bandwidth (adapting to context) but help you find a nearby swimming lake by matching geolocation information with map data (utilization of context).

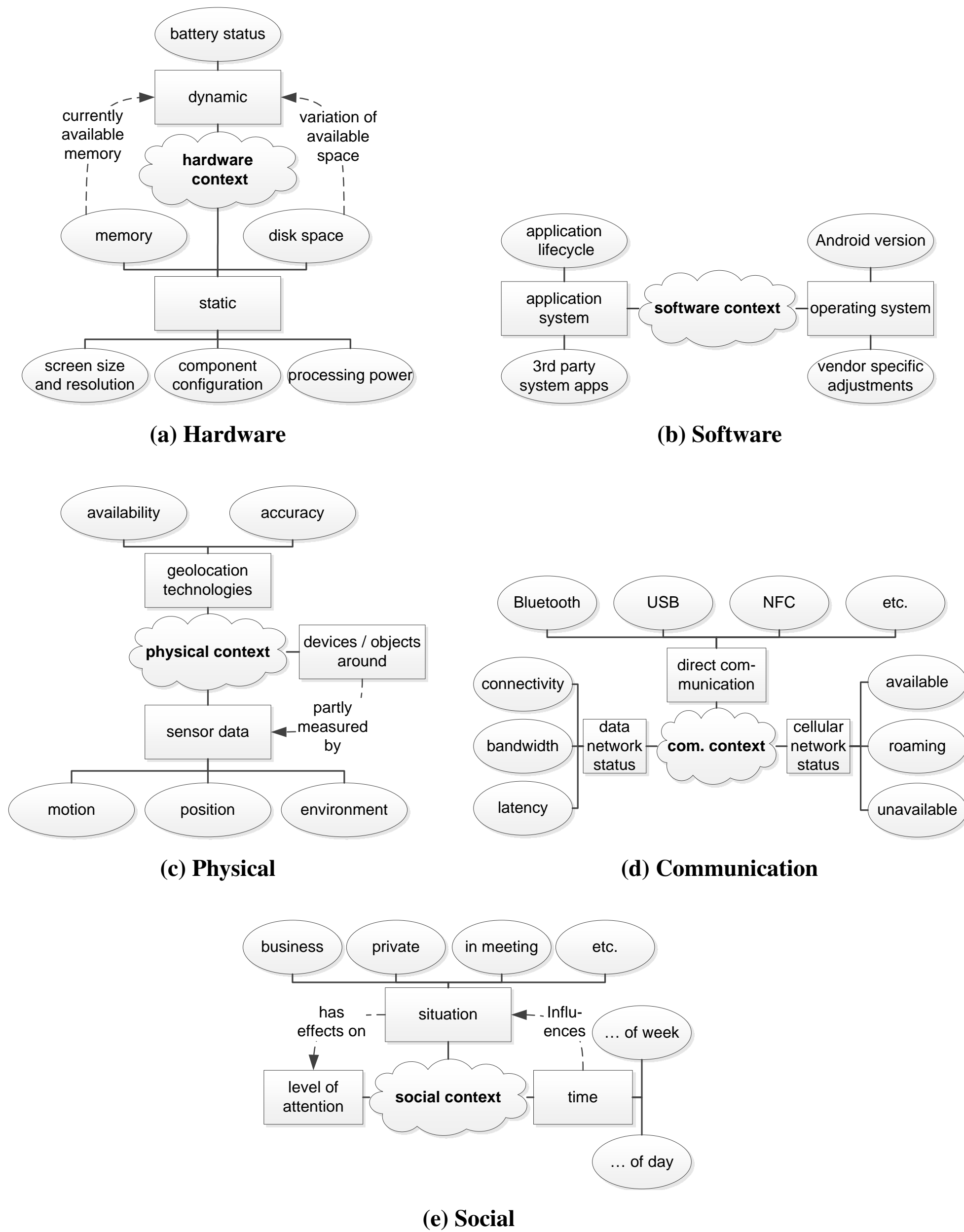
In the following, we propose a categorization scheme for app context that distinguishes five core contexts. It is also sketched in Figures 1a-1e (see p. 29).

Our proposal is not to be seen as (another) formal model for context. In fact, it is a framework of context on mobile devices. Moreover, as presented here it is instantiated for current Android platforms. Since some aspects are device-dependent, minor adaptions are needed for other platforms. Future changes to the ecosystem of mobile apps might lead to the need for additional adjustments. While our concept might be extended to a formal-concept at some point, this would be a topic for a article of its own.

2.2 Hardware Context

The market for mobile devices is greatly fragmented [17]. Apps nowadays run on a multitude of devices, ranging from watches (*smartwatches*) to TVs. While these devices typically are utilized via touch interfaces, they greatly differ in the components they are built from (Figure 1a). Screen size, resolution, screen brightness, contrast and refresh rate, device size, sensors, device extensions, and input means besides touch (e.g. a remote control) vary. Particularly the level of mobility, the screen size and the sensor equipment make some devices hardly comparable besides being based on silicon chips and enabled to run apps. An app might require a camera, thereby limiting the number of devices it can be run on by some degree. If, however, the app requires a specific minimum resolution, the number of excluded devices rises sharply.

[1] App lifecycle models are not further discussed here. More details are given by [43].



(a) Hardware

(b) Software

(c) Physical

(d) Communication

(e) Social

Figure 1: Contexts

Besides the particularities of mobile devices, also contextual factors already known from PCs and servers have to be considered. This includes available memory and disk space. Battery capacity is another factor.

The rate of technology advances in mobile device hardware is still very high. This dynamic is accompanied by the strive to make devices "app enabled" – smart TVs most likely are only a beginning. It is impossible for developers to forecast the upcoming changes, thereby becoming able to design apps perfectly responsively (cf. [40]). Thus, the hardware context can be problematic.

Different hardware can be handled by a strategy of seeking for a kind of "greatest common divisor" among devices. Think of an app that lets you scan barcodes on goods in order to compare prices with products listed in a Web database. There is no reason to consider hardware that is not mobile, such as TV sets. The app should notify users if no camera can be utilized (instead of malfunctioning). Context testing should include reaction to cameras below the quality deemed to be the minimum threshold, at the threshold, and above it – similarly to equivalence partitioning testing [32, p. 28].

2.3 Software Context

Fragmentation in terms of hardware goes along with software fragmentation. Thus, software forms a context of its own (Figure 1b is an example for the Android platform). With at least the four popular platforms (Android, Blackberry, iOS, and Windows Phone) [17], each existing in a variety of versions, app development is problematic. Some versions greatly differ and application programming interfaces (APIs) employed by older, still frequently used devices might be deprecated by now. Moreover, vendors tend to amend official releases with additions of their own. This worsens the situation particularly in the market of Android devices.

It is unlikely that soon a middleware layer [1] or a cross-platform approach [22] will alleviate the problem. In fact, apps existing both in a native version and as a mobile Webapp even complicate testing. Additionally, device-specific software components contribute to complexity.

Context-related problems might also arise from the combination of contextual factors of hardware and software. Apps could behave very similarly using two different versions of a mobile platform, e.g. having a similar performance. This may be different when running on slow devices (which would be a changing hardware context). Data formats might be suited in one place but not in the other.

2.4 Physical Context

Mobility means that the physical context (Figure 1c) is continuously changing [45]. Location determines other contextual factors such as connectivity [46]. Most devices have one or more means to determine their location (more or less accurate, depending on the method and, again, context).

The physical context also comprises other devices that can be contacted with technology such as Bluetooth, Near Field Communication (NFC) and wireless LANs. Moreover, devices are usually equipped with a number of sensors such as gyroscopes, accelerometers, fixtures for temperature measurement and similar units. Their feedback is depending on the physical context. It has to be particularly noted that not all physical contexts can be simulated with today's means – this specifically applies to sensors. Even when using actual devices, testing the physical context can become tremendously hard.

2.5 Communication Context

Depending on the location, connection parameters vary, which influences communication (Figure 1d). The main parameters are availability, bandwidth and latency, along with minor parameters such as jitter. These are determined by the service provider used, the cellular services supported by the device, the current location, and to some extent external circumstances (i.e. the physical context). For example, connection quality might be impaired by weather or by many people using the network cell (e.g. in a sports stadium).

Location typically determines whether only "slow" systems like GPRS or EDGE are available, or technologies such as UMTS, HSPA or LTE can be used. Moreover, devices might connect to wireless LANs, circumpassing the need for a mobile data connection via a service provider.

The communication context gradually changes continuously. The pace of change usually aligns with the rate of mobility. Apps ought to be robust. They should also maintain functionality in offline scenarios. Ideally, they should adjust to the context. For example, an app might load low resolution images instead of full pictures if the available bandwidth drops below a threshold.

The communication context is more complex than it appears. For example, a high-bandwidth cellular service might be readily available but not used by the phone because it is located abroad and thus in roaming mode with disabled data services. A test that checks whether bandwidth is available, attempts to perform an operation and then fails (being unable to send data) might be very hard to interpret and, thus, confusing.

Testing includes simulations of small bandwidths,

high latencies, constant changes in connection quality, and abrupt unavailability of service as well as resuming service. Unfortunately, which bandwidth, latencies etc. are acceptable is highly depending on the app being tested as well as on the user's expectations and usage patterns – and on context.

2.6 Social Context

The social context (Figure 1e) is harder to grasp than the other context categories. It comprises of user-specific ways using an app and of usage particularities.

Firstly, an increasing number of mobile devices are used both for work and for private purposes. *Bring your own device* (BYOD) [12] policies allow private smartphones within a companies' premises (and probably to use its network). However, some apps are used for work while others are used for personal reasons. Some might be used for both but not necessarily in the same way. Due to security reasons, apps might even need to rely on different data sources and make sure that company and private data is not intermingled.

Secondly, while most apps are tailored to single-person use, some mobile devices are used by more than one person. Imagine a child playing a game on a parent's smartphone or a customer viewing material on a (e.g. insurance) sales person's tablet. However, different people use apps uniquely and an app's capabilities might need to be adjusted. For example, customers should see *their* material but must not see materials prepared in the same app for other customers (e.g. an estimate of insurance premiums).

Thirdly, users' attention span will not be the same in all situations. Typical app usage is characterized by changing attention. This can be explained with the mobile nature: the simplest example is people that send short messages while walking. Naturally, they cannot keep staring on the smartphone screens, or they risk bumping into a street light. Moreover, it has been found in a study that even the time of usage influences which apps are used [50].

The social context is very challenging for testing. While most other context changes might be hard to simulate, it is rather straightforward to describe and to measure them. Social contextual factors, though, are fuzzy to some degree, hard to estimate and in many cases impossible to exhaustively simulate and measure.

2.7 Consequences

Apps have to cope with a variety of contexts that are hard to be assessed and to be kept track of. For several contextual factors, frequent changes have to be expected and patterns of change not necessarily are predictable. Making sure that apps behave as expected has to be added on top of the regular testing activities. Due to a multiplicative relationship, this makes testing much more complicated, requires considerable more time, and can be exhausting. The described challenges call for support in effective testing, and for automation to increase efficiency.

3 Related Work

There is not much scientific literature on app development that goes beyond case studies or applying existing methods. In combination with the novelty of our approach this explains why not much closely related work could be identified.

The literature on testing is vast (cf. [32, Chap. 2] for an overview). App development is too new to be given special attention in textbooks on testing. Nevertheless, standard literature is a valuable source since testing of Web-based applications is typically covered (e.g. [44, Chap. 22]). Moreover, techniques for testing of graphical user interfaces (GUI) can typically be applied to apps. Security testing of Web applications [4] is different in concept to our approach, despite some context changes also posing security risks.

Mobility is sometimes covered in textbooks, typically along with Web application [41, Chap. 6]. Some recommendations are already outdated; in general, there is little sophisticated advice. Nevertheless, without mentioning *context* directly, some discussions relate to our ideas. For example [41, pp. 166] highlights different connection speeds of mobile phone services – even though changes during connection time are not mentioned.

Most textbooks on app development do not address testing; exception handling and other app-specific error processing concepts are rather discussed (cf. e.g. [29, Chap. 4]). It has to be kept in mind that most titles are not academic and address beginners, though. At least an e-book on Android testing could be identified [36]. Nevertheless, the small attention given to quality aspects – let alone testing – in app development books is disappointing. At the same time, some authors try to disillusion hobbyist programmers about the ease of making money by implementing apps [51].

There is a high number of papers that present work on app testing, often in an research-in-progress state. Typical directions of research are test automation [14], user-centered testing [20], tools [18, 24], testing approaches [3, 37], and specifically user interface testing [53, 54, 9, 10]. All these papers – and quite a few more that deal with similar issues – only roughly relate for they tackle testing of apps but are conceptually different. In general, the research papers suggest that app testing will mature. Some authors even contrive next-generation

business models such as offering *Mobile Testing-as-a-Service* [15]. Testing (or rather the level of support for it) also is a criterion when assessing mobile device platforms [35].

Strictly speaking, even random testing of apps [30] is related to our work. Rather than comparing concepts it will, however, be a task of future work to compare the efficacy of sophisticated app testing methods (such as the one presented in this article) to simple ones (such as random testing).

The sole more closely related work we identified is presented by Amalfitano et al. [2]. They consider context as the result from events that can be triggered by the user, the phone, or external activities. To consider events in testing, they propose to identify patters. These are used for manual, mutation-based and exploration-based testing. Our work is not competing with theirs but complementary.

The relevance of context in mobile computing has been discussed as early as in 2001 [42]. In most cases context is used in connection with awareness, i.e. the ability of mobile devices to perceive context changes and react accordingly [7]. Context is used to recommend apps [39, 27], adapt automatically [19], and to predict usage patterns [52]. While these works contribute to our understanding of contexts, they only support testing by explaining *what* can cause context changes.

Finally, there is general work on testing that keeps context in mind. Papers deal with context awareness (e.g. in autonomous systems [49] or focusing sensors [8]), model checking [13], multithreading [38], and Web services [6]. The notion of *context* differs; the term is employed in various fields of computer science. Nevertheless, several of the cited articles have a similar understanding of context as we do and underline the importance of being sensitive to context changes.

Summing up, there is a plethora of roughly related work that elaborate the importance of context changes and describe different contexts. However, there is merely one other approach that combines context changes with testing so far.

4 Context-Sensitive Testing

4.1 General Considerations

To combine context changes with app testing, it has to be possible to automatically change context parameters whilst test cases are executed. With block-based context-sensitive testing we focus on context elements that are dynamic, i.e. possibly changing during apps usage. Doing so, the changing contexts that are induced by a user being mobile can be simulated.

The naive approach would be to specify the context for each test case a priori. While being beneficial for unit tests, asserting that a single code unit produces an expected output in a certain context would not be helpful for testing business processes with integration tests. As many influencing contextual factors are not static but change constantly during usage, a more dynamic approach is needed.

Providing the possibility (e.g. by an API) to specify context changes within test cases would allow to simulate changing contexts. However, this approach is still static. Context changes have to be explicitly stated *within* test cases. Thus, when testing scenarios including different variations of context, many test cases are required. Each test case would contain the same test code; only statements for changing contexts would vary. This multiplication is not desirable. Test cases would be costly to develop and execute, and hard to maintain let alone to reuse. Moreover, the right sequence of context changes has to be anticipated beforehand.

4.2 Blocks, Assertions and Context Changes

To provide a solution that fosters dynamic changes of context, we use modularization. Test cases are split into *blocks*; between each block a *change of context* is possible. Blocks are reused and combined with different context changes. Therefore, testing multiple scenarios is possible without duplication of test code. It is even possible to generate scenarios without user-interaction. As blocks are the foundation of our concept, we call it *block-based context-sensitive testing*.

A given test case may result in a number of blocks that form the structure of a context-sensitive test. Each block contains *operations* and *assertions*. Similar to unit testing frameworks, operations are needed to simulate user interaction and assertions are used to verify expected behavior. Our idea is to derive blocks from existing test cases, such as the so called *happy path* – a basic test case without extraordinary context changes. Doing so, a single test case may be transformed into a structure of blocks that can be used to generate context-dependent test cases. To preserve the intention of test cases, blocks have to be ordered and executed consecutively.

Listing 1 illustrates in a schematic way how a test case looks like. If the test code itself would be divided into blocks, this schematic test case contains two of them: one from lines 3 to 7 (*block A*) and one from lines 10 to 12 (*block B*). The scope of each block has to be atomic w.r.t. changing contexts. In other words, during execution of test operations belonging to one single block, the context remains stable. Only between blocks changes of context are possible (lines 2 and 9). To realize different scenarios, test operations do not need to be repeated but only the context changes in between have to be altered.

Listing 1: Schematic test case with context changes

```
1  public void testExample(){
2  contextChange(contextA);
3  clickSomewhere();
4  enterText();
5  clickButton();
6  ...
7  assertThat();
8
9  contextChange(contextB);
10 doSomeOtherThings();
11 ...
12 assertThat();
13 }
```

The solution is expected to be most beneficial utilizing a generator that automatically creates test cases from blocks. Manual effort for writing context-sensitive test cases is reduced and redundant test code minimized. Moreover, there is a separation of concerns: Operations within blocks are only used to address an app's functionality; together with context changes in between they are used to assess the behavior under changing contexts.

As many apps may behave differently in certain situations (e.g. display error messages when no connection is available), assertions have to be extracted from blocks due to their potential dependency on a context. For each block at least one *default assertion* has to be assigned. This assertion is used to verify the app's standard behavior. For each known context, a specific assertion may be defined that is used to assess context-dependent behavior. The blocks shown in the schematic test case in Listing 1 therefore have to be tailored smaller. Strictly spoken, assertions in lines 7 (Block A) and 12 (Block B) are not part of the blocks but have to be treated as another building block of our concept. Depending on the app's expected behavior, assertions may be *default* or *context-sensitive*.

The building blocks of the concept of block-based context-sensitive testing are shown in Figure 2. The structure of a test case is depicted by an ordered list of blocks. Each of them has at least one default assertion. Further assertions for different contexts may be added.

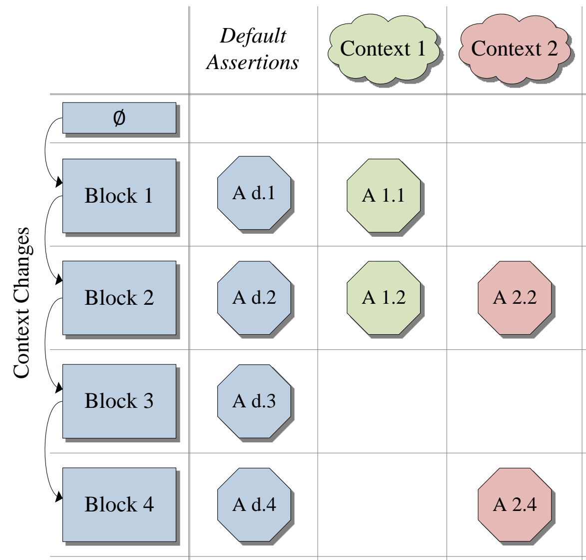


Figure 2: Concept of block-based context-sensitive testing

The second block is executed similarly to the first one; a context-specific assertion part is used as well. This changes when reaching `Block 3`. As can be seen in Figure 2, this block does not have any specific assertions. Consequently, the expected behavior is invariant between various contexts. Therefore, the default assertion that verifies this behavior is evaluated. Finally, `Block 4` is executed together with assertion `A2.4`, which is the assigned assertion for `Context 2`.

As the matrix in Figure 2 shows, there are numerous execution paths considering the fact that prior to each block the context is changeable and alternative assertions can be defined. The matrix grows with the number of contexts but will be sparse when not defining specific assertion parts for all blocks.

Using the concept of block-based context sensitive testing, the structure of the test remains static as the list of blocks in the sample shows. However, the context in between is changing and also the used assertions can vary.

4.3 Context-Dependent Test Case Execution

A possible test execution is illustrated in Figure 3. The beginning of the test case is denoted as the empty set $\emptyset$. Before the first block is executed, `Context 1` is established. Next, an assertion fitting to the current context is searched. Since this block has a specific assertion for `Context 1`, this one is used. As the assertion holds, execution continues. Between `Block 1` and `Block 2` the context is changed again. This time, `Context 2` is established and kept for the remaining execution.

4.4 Sample Scenario

To illustrate practical benefits, we implemented a proof-of-concept tool and evaluated the approach in cooperation with an industry partner. We use a simple app to explain how it works: a client for the micro blogging service *Twitter*. The app contains two screens, one for logging in to the service and another for posting messages. Error messages are shown when communication with the service is impossible due to missing connectivity.

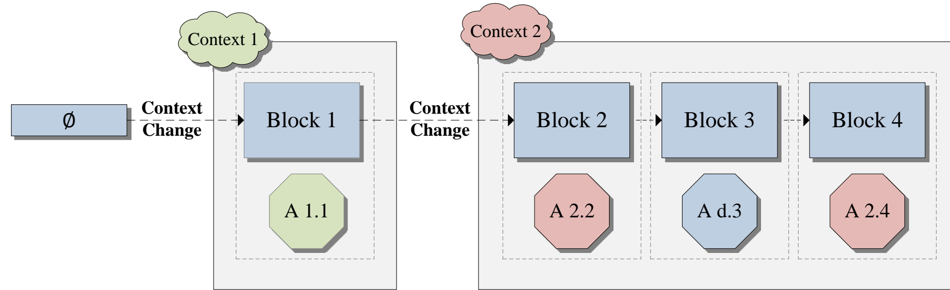


Figure 3: Example of test execution using block-based context-sensitive testing

To prepare the setting, a jar archive containing our proof-of-concept has to be added to the *classpath* of the app's Android testing project. Moreover, as the Internet connectivity has to be changed to test the app in various contexts, the permissions shown in Listing 2 have to be added to the `AndroidManifest.xml` of the app if not already contained.

Listing 2: Required Android permissions

```
1 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
2 <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
```

The following sample test is conducted in form of an integration test, testing the app in conjunction with the real system environment. The high level process steps are: logging in with invalid credentials, logging in with correct credentials, and posting a message. These three steps form the blocks that are used to test the Twitter app. Each step can later be conducted in a different context.

Listing 3 (see next page) shows the first block implemented with our solution. The test operations, namely filling in user name and password and clicking the login button afterwards, are implemented in the operation-part of the block (lines 4 to 9). The credentials used in that block are invalid; therefore, the app is expected to show a corresponding message. This is checked by the default assertion (lines 10 to 12). However, if the app is in *disconnected* context, it is expected to show an error message. In order to verify this context-dependent behavior, an alternative assertion is added for the disconnected context (lines 15 to 20).

For each of the above stated process steps a block with context-specific assertions is implemented and added to a list of blocks. Using a generator that creates different mutations in terms of context changes from that list, test cases are generated. Figure 4 shows one possible test case. The first two blocks are executed in connected context and their default assertions are used for verifying. Before executing the third block, the context is changed. The context dependent assertion for that block checks whether in the *disconnected* context the message "No network connection" is shown. It becomes clear that even in a small scenario as illustrated here a lot of different execution paths are possible, which underlines why testing without our approach would be burdensome.

Listing 3: Sample block implementation

```
1  Context discContext = new Context(ConnectionStatus.DISCONNECTED);
2
3  Block loginIncorrectBlock = new Block(){
4    public void operation() {
5      enterText(usrName, "dummy@user.de");
6      enterText(pwd, "1234");
7      clickOnButton(0);
8    }
9
10   public void defaultAssertion() {
11     assertTrue(waitForText("Authentication failed!"));
12   }
13 };
14
15 loginIncorrectBlock.addAlternativeAssertion(discContext, new Assertion() {
16   public void assertion() {
17     assertTrue(waitForText(
18       "Could not login: No network connection."
19     ));
20   }
21 });
```

We used two sample generators to create test cases from the blocks explained above. One alternates the context between each block. Another generates *n+1* test cases where *n* is the number of blocks contained. In each test case the generator changes the context at different steps in the process. For the first test case the context is changed before the first block is executed. In the second test case it is changed after the first block has been executed, and so on.

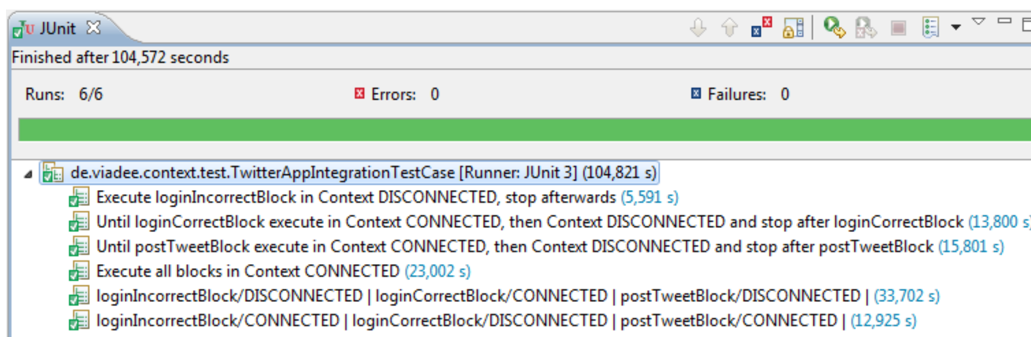
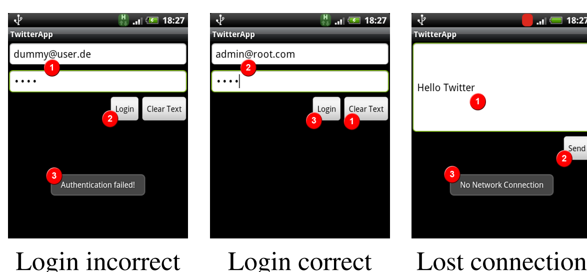


Figure 5: Example JUnit result



Login incorrect Login correct Lost connection

Figure 4: Example process steps for test case execution for various conditions

Finally, test cases are executed by means of JUnit. Like any other test for the Android platform, they are running on the device or emulator itself. Figure 5 shows test results for one execution of the above stated example.

5 Discussion

5.1 Exemplary Tool, Application and Merits

To demonstrate the producibility of our concept, we have implemented an exemplary Android tool. It is written in Java and can be checked out from GitHub [11]. Release under the Apache License allows free usage and modification. It can be inspected and used in other projects. In particular, studying the source code will allow rapid development of similar tools for platforms such as Apple iOS and Microsoft Windows Phone.

While the tool is not a core contribution of our work, it demonstrates the feasibility of our approach. In particular, it allows to retrace our concept and to check its practicability for own scenarios. Moreover, the implementation can be seen as a first step towards powerful context-based testing tools.

Together with an industry partner[2], we exemplarily tested a few apps and evaluated our approach of block-based context-sensitive testing. By conducting interviews and doing tests as shown in Section 4.4 we identified the core contributions.

We found that by using our proof-of-concept implementation it is possible to test apps in various contexts effectively. Blocks of test code are reusable and thus the effort in writing tests is reduced. This reduction also holds true for the amount of duplicated test code which, in practice, would be hard to maintain.

By using generators the concept even gets more effective in terms of finding context-related issues in apps. Testers only have to implement blocks and define context-specific assertions once, but many test cases are generated automatically. When writing manual tests, the context changes may only be inserted at parts where context-related issues are anticipated. However, using our approach helps to find errors in code units where they are not expected beforehand.

Manually conducting field tests, i.e. practical tests in various contexts, is often expensive in terms of time and budget. Thus, they are neglected frequently. Our concept offers an alternative to conduct at least some of these tests in laboratory conditions. Doing so, context-related issues may be identified before release of an app, which could reduce costs for fixing them and increases the app's quality.

However, our concept cannot (yet) be a panacea when dealing with context during development and testing of apps. We deem it a supportive mean for testing parts of apps which are heavily influenced by context.

5.2 Limitations

Due to the novelty of our approach, a number of limitations have to be mentioned.

Firstly, there are conceptual limitations. Having an ordered linear list of blocks allows for only one way of execution. However, changes of context may also induce different business processes to be executed. Thus, the

[2] The industry partner is the employer of one of us (Schulte) at the same time.

order of blocks and the decision if a block is executed may also depend on the context used. For instance, if in our above sample test (see Section 4.4) the login block would be executed in disconnected context, the test has to be stopped because the other process steps cannot be reached. We introduced a method for aborting test case execution in our proof-of-concept to cope with this. That method has to be included in assertion parts corresponding with contexts causing the business process to stop. Anyhow, our concept is not able to deal with process steps that only have to be executed in certain contexts. Closely related, assertion parts are only dependent on the current context but not on the prior execution path of the test. In practice, the expected behaviour of process step may change according to the context the prior steps are executed in. Overcoming this limitation requires a refinement.

Secondly, the case study is a first example of feasibility, yet not exhausting. Effectiveness has to be proven in industrial settings, both qualitatively and quantitatively. We got very positive feedback from an industry partner and approval from the practitioners' testing community (see Section 5.1). Feedback has to be assessed scientifically to become a valid evidence, though.

Thirdly, our tool is a prototype and, thus, poses limitations. In particular, it is scarcely commented and it is not yet very user friendly. Due to employing testing best practices such as relying on a common unit testing tool like JUnit [26] it is easy to use for programmers and technical testers. For domain experts, hobbyist programmers, and to a lesser extent for pure Web developers it will, however, be cumbersome due to a missing graphical user interface.

These limitations have to be kept in mind yet none of them question the general feasibility of our approach. They also lead to future work.

5.3 Challenges and Future Work

One major practical challenge in using our approach is to find the right size of blocks in order to balance the effort of slicing test cases into blocks with various assertions against the probability of finding additional faults by doing so. “Best practices”, or guidelines of how context dependent test cases may look like, need to be developed.

For future work, an extension of our tool could prove beneficial. One idea is to enable graphical modelling, which would ease test case creation. Ideally, models e.g. describing business processes could be reused. Moreover, better support for test case generation w.r.t. context selection is desirable. For example, the tool could keep track of assertions and contexts and provide testers with reasonable suggestions. Additionally, it could track coverage and advise on finding minimal sets of test cases that cover all possible paths.

To make the approach more versatile, we also intend to learn from other works that deal with scaling up computer science concepts. Certainly, for larger programs means will be needed to cope with an ever-increasing number of concepts and possibilities. An example of approaches that could be worth a look is the work on AHEAD by Batory, Sarvela and Rauschmayer [5]. Also, we will try to relate even stronger to existing concepts to overcome the remaining challenges. For example, insights from aspect-oriented programming (AOP) regarding assertion handling [25] and even-driven testing techniques [28] might be applicable.

Best practices for context-sensitive testing would complement the work with our approach. Compiling them requires a base of regular users and a lot of time. Nevertheless, increased awareness of context issues would facilitate progress.

A challenge is to become able to cope with ample contexts. While our approach theoretically is capable of dealing with arbitrary context changes, actually simulating these changes for testing is very hard. Problems particularly arise from parallel (or rather quasi-parallel) changes of contexts. As a consequence, future work needs to refine our concept and investigate into the consequences of context changes (some might e.g. be *harmless*). Research could lead to better tools for simulation, making our approach even more useful.

6 Conclusion

In this paper we presented a novel approach for testing apps. We coined it block-based context-sensitive testing since we address context changes while defining blocks between which they can take place. The approach enables much more versatile and effective testing, while the overhead is small due to the relatively low level of complexity. Our concept of context changes extends the literature of context-sensitivity in mobile computing. We have shown the viability of our approach in a case study and by presenting a prototype.

Testing of apps is far from being hassle-free. We expect new challenges to arise due to the extended capabilities of devices and platforms, e.g. w.r.t. sensors and parallel execution. The proliferation of the Internet of Things (IoT) will likely contribute to the complexity that needs to be coped with. Our work on context-sensitive testing will continue.

Acknowledgments

We would like to thank viadee Unternehmensberatung GmbH for their support.

References

[1] V. Agarwal, S. Goyal, S. Mittal, and S. Mukherjea, "Mobivine: A middleware layer to handle fragmentation of platform interfaces for mobile applications," in *Proc. of the 10th ACM/IFIP/USENIX Int. Conf. on Middleware*. New York, NY, USA: Springer, 2009, pp. 24:1–24:10.

[2] D. Amalfitano, A. R. Fasolino, P. Tramontana, and N. Amatucci, "Considering context events in event-based testing of mobile applications," in *Proc. 2013 IEEE Sixth Int. Conf. on Software Testing, Verification and Validation Workshops (ICSTW)*. Washington, DC, USA: IEEE Computer Society, 2013, pp. 126–133.

[3] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in *Proc. ACM SIGSOFT 20th Int. Symp. on the Foundations of Software Engineering (FSE)*. New York, NY, USA: ACM, 2012, pp. 59:1–59:11.

[4] M. Andrews and J. A. Whittaker, *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*. Addison-Wesley, 2006.

[5] D. Batory, J. N. Sarvela, and A. Rauschmayer, "Scaling step-wise refinement," *IEEE Transactions on Software Engineering*, vol. 30, no. 6, pp. 355–371, 2004.

[6] F. Belli and M. Linschulte, "Testing composite web services–an event-based approach," in *Proc. IEEE Int. Conf. on Software Testing, Verification, and Validation Workshops (ICSTW)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 307–310.

[7] M. Böhmer, C. Lander, and A. Krüger, "What's in the apps for context?" in *Proc. 2013 ACM Conf. on Pervasive and Ubiquitous Computing Adjunct Publication (UbiComp)*. New York, NY, USA: ACM, 2013, pp. 1423–1426.

[8] P. Campillo-Sanchez, E. Serrano, and J. A. Botía, "Testing context-aware services based on smartphones by agent based social simulation," *J. Ambient Intell. Smart Environ.*, vol. 5, no. 3, pp. 311–330, 2013.

[9] W. Choi, "Automated testing of graphical user interfaces: A new algorithm and challenges," in *Proc. 2013 ACM Workshop on Mobile Development Lifecycle (MobileDeLi)*. New York, NY, USA: ACM, 2013, pp. 27–28.

[10] W. Choi, G. Necula, and K. Sen, "Guided GUI testing of Android apps with minimal restart and approximate learning," *SIGPLAN Not.*, vol. 48, no. 10, pp. 623–640, Oct. 2013.

[11] "contextTesting @GitHub," 2015, https://github.com/viadee/contextTesting.

[12] G. Disterer and C. Kleiner, "Using mobile devices with BYOD," *Int. J. Web Portals*, vol. 5, no. 4, pp. 33–45, 2013.

[13] L. Duan and J. Chen, "An approach to testing with embedded context using model checker," in *Proc. 10th Int. Conf. on Formal Methods and Software Engineering (ICFEM)*. Berlin, Heidelberg: Springer, 2008, pp. 66–85.

[14] J. Gao, X. Bai, W.-T. Tsai, and T. Uehara, "Mobile application testing: A tutorial," *Computer*, vol. 47, no. 2, pp. 46–55, 2014.

[15] J. Gao, W.-T. Tsai, R. Paul, X. Bai, and T. Uehara, "Mobile testing-as-a-service (MTaaS)," in *Proc. 2014 IEEE 15th Int. Symp. on High-Assurance Systems Engineering (HASE)*. IEEE CS, 2014, pp. 158–167.

[16] "Gartner Press Release," 2012, http://www.gartner.com/it/page.jsp?id=1924314.

[17] "Gartner Press Release," 2014, http://www.gartner.com/newsroom/id/2944819.

[18] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "Reran: Timing- and touch-sensitive record and replay for android," in *Proc. 2013 Int. Conf. on Software Engineering (ICSE)*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 72–81.

[19] T.-M. Gronli, J. Hansen, G. Ghinea, and M. Younas, "Context-aware and cloud based adaptation of the user experience," in *Proc. 2013 IEEE 27th Int. Conf. on Advanced Information Networking and Applications (AINA)*. Washington, DC, USA: IEEE Computer Society, 2013, pp. 885–891.

[20] K. Haller, "Mobile testing," *SIGSOFT Softw. Eng. Notes*, vol. 38, no. 6, pp. 1–8, Nov. 2013.

[21] T. Hanna, *Apps on TV*. Berkely: Apress, 2012.

[22] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Evaluating cross-platform development approaches for mobile applications," in *Proc. 8th WEBIST, Revised Selected Papers*, ser. Lecture Notes in Business Information Processing (LNBIP). Springer, 2013, vol. 140, pp. 120–138.

[23] H. Heitkötter, T. A. Majchrzak, U. Wolffgang, and H. Kuchen, *Business Apps: Grundlagen und Status quo*, ser. Working Papers. Förderkreis der Angewandten Informatik an der Westfälischen Wilhelms-Universität Münster e.V., 2012, no. 4.

[24] G. Hu, X. Yuan, Y. Tang, and J. Yang, "Efficiently, effectively detecting mobile app bugs with appdoctor," in *Proc. Ninth European Conf. on Computer Systems (EuroSys)*. New York, NY, USA: ACM, 2014, pp. 18:1–18:15.

[25] U. Juárez-Martínez, G. Alor-Hernández, R. Posada-Gomez, J. Santos-Luna, J. Gomez, and A. Gonzalez, "An aspect-oriented approach for assertion verification," in *First International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, 2009, pp. 74–79.

[26] "JUnit," 2015, http://www.junit.org/.

[27] A. Karatzoglou, L. Baltrunas, K. Church, and M. Böhmer, "Climbing the app wall," in *Proc. 21st ACM Int. Conf. on Information and Knowledge Management (CIKM)*. New York, NY, USA: ACM, 2012, pp. 2527–2530.

[28] A. Kumar and R. Goel, "Event driven test case selection for regression testing web applications," in *International Conference on Advances in Engineering, Science and Management (ICAESM)*, 2012, pp. 121–127.

[29] H. Lee and E. Chuvyrov, *Beginning Windows Phone App Development*. Apress, 2012.

[30] Z. Liu, X. Gao, and X. Long, "Adaptive random testing of mobile application," in *2nd International Conference on Computer Engineering and Technology (ICCET)*, vol. 2, 2010, pp. V2–297–V2–301.

[31] M. Macedonia, "iPhones Target the Tech Elite," *Computer*, vol. 40, pp. 94–95, 2007.

[32] T. A. Majchrzak, *Improving Software Testing: Technical and Organizational Developments*. Heidelberg: Springer, 2012.

[33] T. A. Majchrzak and H. Heitkötter, "Development of mobile applications in regional companies: Status quo and best practices," in *Proc. 9th Int. Conf. on Web Information Systems and Technologies (WEBIST)*, 2013.

[34] T. A. Majchrzak and M. Schulte, "Context-Dependent App Testing," in *Proc. of the 27th Conference on Advanced Information Systems Engineering (CAiSE) Forum*. CEUR, 2015, pp. 73–80.

[35] T. A. Majchrzak, S. Wolf, and P. Abbassi, "Comparing the capabilities of mobile platforms for business app development," in *Proc. of the 8th SIGSAND/PLAIS EuroSymposium on Systems Analysis and Design Information Systems: Development, Applications, Education*, ser. Lecture Notes in Business Information Processing (LNBIP), S. Wrycza, Ed. Springer, 2015.

[36] D. T. Milano, *Android application testing guide*. Packt Publishing, 2011.

[37] N. Mirzaei, S. Malek, C. S. Păsăreanu, N. Esfahani, and R. Mahmood, "Testing android apps through symbolic execution," *SIGSOFT Softw. Eng. Notes*, vol. 37, no. 6, pp. 1–5, 2012.

[38] M. Musuvathi and S. Qadeer, "Iterative context bounding for systematic testing of multithreaded programs," *SIGPLAN Not.*, vol. 42, no. 6, pp. 446–455, Jun. 2007.

[39] N. Natarajan, D. Shin, and I. S. Dhillon, "Which app will you use next?: Collaborative filtering with interactional context," in *Proc. 7th ACM Conf. on Recommender Systems (RecSys)*. New York, NY, USA: ACM, 2013, pp. 201–208.

[40] M. Nebeling and M. C. Norrie, "Responsive design and development: Methods, technologies and current issues," in *Proc. of the 13th International Conference on Web Engineering (ICWE)*. Berlin, Heidelberg: Springer, 2013, pp. 510–513.

[41] H. Q. Nguyen, *Testing Applications on the Web*. Wiley, 2003.

[42] L. E. Nugroho, "A context-based approach for mobile application development," Ph.D. dissertation, Monash University, 2001.

[43] E. Payet and F. Spoto, "An operational semantics for android activities," in *Proc.of the ACM SIGPLAN 2014 Workshop on Partial Evaluation and Program Manipulation (PEPM)*. New York, NY, USA: ACM, 2014, pp. 121–132.

[44] W. Perry, *Effective methods for software testing*, 3rd ed. New York, NY, USA: Wiley, 2006.

[45] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Proc. of the 1994 First Workshop on Mobile Computing Systems and Applications (WMCSA)*. Washington, DC, USA: IEEE Computer Society, 1994, pp. 85–90.

[46] A. Schmidt, M. Beigl, and H.-W. Gellersen, "There is more to context than location," *Computers & Graphics*, vol. 23, no. 6, pp. 893–901, 1999.

[47] M. Schulte and T. A. Majchrzak, "Kontextabhängiges Testen von Apps," *Testing Experience de*, pp. 8–13, Oktober 2013.

[48] S. Segan, "Phablet History," 2012, http://www.pcmag.com/slideshow/story/294004/enter-the-phablet-a-history-of-phone-tablet-hybrids.

[49] S. Taranu and J. Tiemann, "General method for testing context aware applications," in *Proc. 6th Int.*

Workshop on Managing Ubiquitous Communications and Services (MUCS). New York, NY, USA: ACM, 2009, pp. 3–8.

[50] H. Verkasalo, "Propensity to use smartphone applications," in *Proc. 5th UBICOMM*, 2012.

[51] J. Williamson, *App Idiots*. self-published (e-book), 2014.

[52] Y. Xu, M. Lin, H. Lu, G. Cardone, N. Lane, Z. Chen, A. Campbell, and T. Choudhury, "Preference, context and communities: A multi-faceted approach to predicting smartphone app usage patterns," in *Proc. 2013 Int. Symp. on Wearable Computers (ISWC)*. New York, NY, USA: ACM, 2013, pp. 69–76.

[53] W. Yang, M. R. Prasad, and T. Xie, "A grey-box approach for automated GUI-model generation of mobile applications," in *Proc. 16th Int. Conf. on Fundamental Approaches to Software Engineering (FASE)*. Berlin, Heidelberg: Springer, 2013, pp. 250–265.

[54] C.-C. Yeh, S.-K. Huang, and S.-Y. Chang, "A black-box based android GUI testing system," in *Proc. 11th Annual Int. Conf. on Mobile Systems, Applications, and Services (MobiSys)*. New York, NY, USA: ACM, 2013, pp. 529–530.

AUTHOR BIOGRAPHIES

Tim A. Majchrzak is an associate professor at the Department of Information Systems at the University of Agder in Kristiansand, Norway. He received BSc and MSc degrees in Information Systems and a PhD in economics (Dr. rer. pol.) from the University of Münster. His research comprises both technical and organizational aspects of software engineering, often in the context of Web technologies and Mobile Computing. He has also published work on several interdisciplinary Information Systems topics. Tim's research projects typically have an interface to industry.

Matthias Schulte is an IT Consultant at viadee Unternehmensberatung GmbH. He was awarded BSc and MSc degrees in Information Systems by the University of Münster. During his studies, he focused on software engineering and, in particular, on concepts for software testing as well as business process management. His professional work also focuses on the area of quality management and software engineering. Mobile application development recently attracted Matthias' attention.