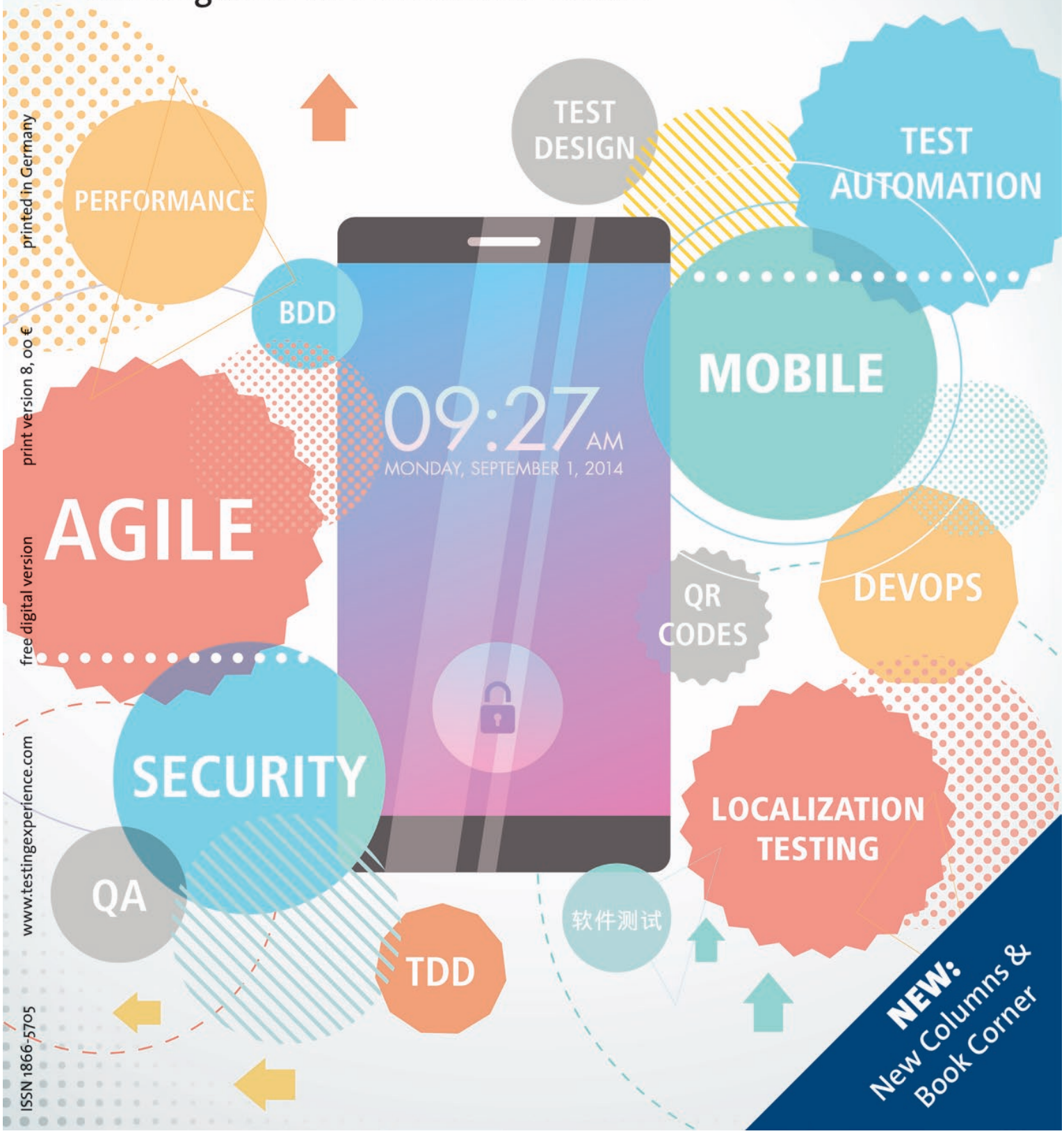
27

September 2014

testing experience

The Magazine for Professional Testers



printed in Germany
print version 8,00 €
free digital version
www.testingexperience.com
ISSN 1866-5705

NEW: New Columns & Book Corner

By Daniel Knott

How to Stress Test Your Mobile App

Stress and interrupt testing is an important part of the mobile testing process. With the help of tools, mobile testers are able to determine possible performance or stability issues of the app. To test your app against interrupts, you can manually trigger lots of notifications to the device while using the app. Notifications can be incoming messages, calls, app updates, or push notifications (software interrupts). Pressing the volume up and down button, or any other kind of hardware button, is also an interrupt (hardware interrupt) that can have an impact on your app.

Doing all those tasks manually means lots of work and is time consuming. In most cases, those test scenarios cannot be done manually, because it is very hard to simulate fast and multiple user inputs with one or two hands. But it can be done with the help of tools and it is really easy to integrate into the development and testing process.

Android Monkey

For Android apps, the Monkey tool [MON01] can be used, which is part of the Android SDK. Monkey is a tool that is able to run either on the physical device or the emulator. While running, it generates pseudo-random user events such as touch, click, rotate, swipe, mute the phone, shutdown the internet connection, and many more to stress test the app and to see how the app handles all those inputs and interrupts.

You need the package name of the Android apk file to execute the Monkey tool, otherwise the tool will execute its random commands to the entire phone instead of to the app under test.

With access to the app code, the package name can be found in the AndroidManifest.xml. If only the compiled apk file is available, mobile testers can use the Android Asset Packaging Tool [AAP02] (aapt), to get the package name from the app. aapt is located in the build-tools folder of the installed Android SDK version.

The path to aapt can look like this:

```
/../daniel/android/sdk/build-tools/android-4.4/
```

With the following command, the package name can be read out from the apk file:

```
./aapt d badging /daniel/myApp/myApp.apk | grep 'pack'
```

The output will look like this:

```
...
package: name='com.myApp' versionCode='' versionName=''
...
```

When the package name (in this case **com.myApp**) is available, execute Monkey with the help of adb (Android Debug Bridge) [ADB03].

The following command will start Monkey:

```
./adb shell monkey -p com.myApp -v 2000
```

The 2000 indicates the number of random commands that Monkey will perform on the app. With an additional parameter `-s` for *seed*, monkey will generate the same sequence of events again. This is really important when reproducing a bug, which happens during the Monkey execution time.

UI AutoMonkey

For iOS apps there is a similar tool available, called UI AutoMonkey [UIA04]. UI AutoMonkey is also able to generate multiple commands in order to stress test an iOS app. To use UI AutoMonkey, a UIAutomation Instruments template must be configured within Xcode. After the template has been configured, a JavaScript file needs to be written to tell the tool how many and which commands should be executed during the stress-testing session.

Sample UI AutoMonkey Script

```
1.  ...
2.  config: {
3.      numberOfEvents: 2000,
4.      delayBetweenEvents: 0.05,    // In seconds
5.          // Events that will be triggered on the phone
6.          eventWeights: {
7.              tap: 300,
8.              drag: 12,
9.              flick: 15,
10.             orientation: 20,
11.             clickVolumeUp: 10,
12.             clickVolumeDown: 10,
13.             lock: 1,
14.             pinchClose: 10,
```

```
            pinchOpen: 10,
            shake: 10
        },
        touchProbability: {
            multipleTaps: 0.05,
            multipleTouches: 0.05,
            longPress: 0.05
        }
}
...
```

If the script is written, it can be executed within Xcode to stress test the iOS app.

At the end of the test run, both tools will be generating an overview of possible errors or problems that occur within the app.

Note: The detailed installation instructions and the complete sample script can be found on the tool manufacturer's website.

Both tools can be integrated into a continuous integration system to run automatically after every commit. Stress and interrupt testing a mobile application is pretty simple and should be one part of the mobile testing strategy. Besides that, it generates a huge benefit for mobile testers, helping the team to build a reliable and robust mobile app. ■

References

[MON01]: Android Monkey
developer.android.com/tools/help/monkey.html

[AAP02]: Android Tool aapt
developer.android.com/tools/building/index.html

[ADB03]: Android Debug Bridge
developer.android.com/tools/help/adb.html

[UIA04]: UI Auto Monkey
github.com/jonathanpenn/ui-auto-monkey

> about the author

Daniel Knott has been working in the field of software testing since 2008. In his career he has worked for companies such as IBM, Accenture, XING, and AOE. In different agile projects Daniel gained in-depth knowledge of software testing, e.g., in mobile, search, recommendation, and web technologies. During his time at XING, Daniel worked as a Team Lead QA in the mobile team and developed a fully automated testing framework for Android and iOS.
Currently, he is working as Software Test Manager at AOE GmbH where he is responsible for test management and test automation in mobile and web projects. Daniel is also a frequent speaker at agile conferences, author of his blog and of articles in testing magazines.
Twitter: *@dnlknt*
LinkedIn: *www.linkedin.com/pub/daniel-knott/1a/925/993*
Blog: *www.adventuresinqa.com*