Automated Integration Tests for Mobile Applications in Java 2 Micro Edition

Dawid Weiss, **Marcin Zduniak**

Institute of Computing Science
Poznan University of Technology

2007

Automated Integration Tests for
Mobile Applications in Java 2 Micro Edition

Dawid Weiss, **Marcin Zduniak**

Institute of Computing Science
Poznan University of Technology

2007

Hello everyone, my name is Marcin Zduniak, I am a grad student at Poznan University of Technology and I'm here to present the initial results of my Master's thesis.

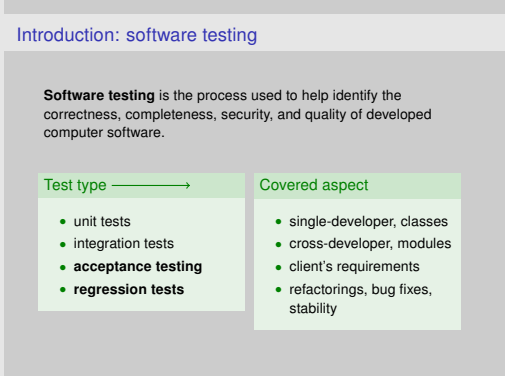
The subject of my work was a proof-of-concept – to design and implement a semi-automatic capture-replay testing environment for mobile applications written for the Java 2 Microedition environment.

I would like to share with you the difficulties of this task and the approaches in which we solved them. First of all, let me give you an outline of the motivation and scope of our work.

Introduction: software testing

Software testing is the process used to help identify the correctness, completeness, security, and quality of developed computer software.

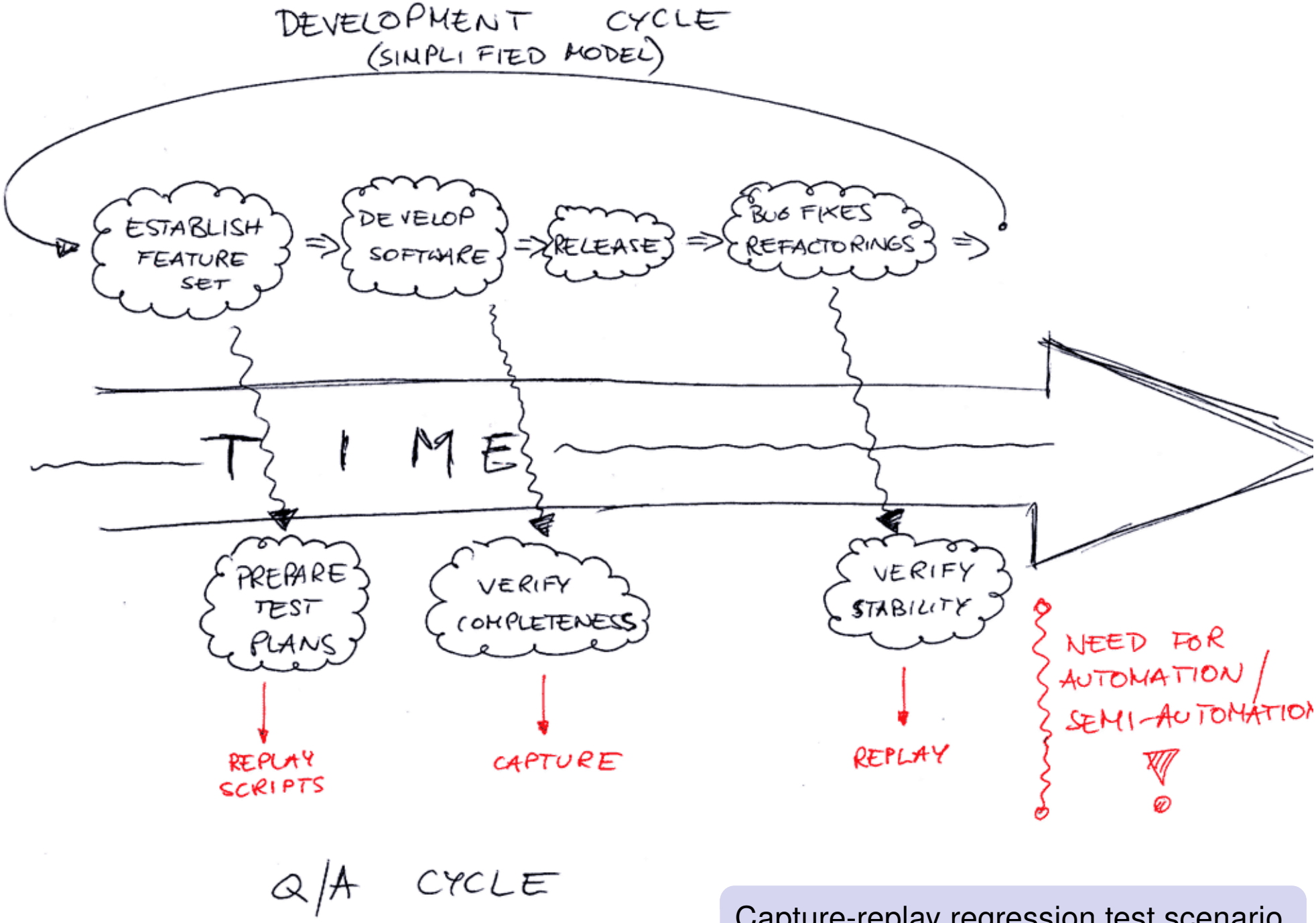
Test type ⟶	Covered aspect
• unit tests	• single-developer, classes
• integration tests	• cross-developer, modules
• **acceptance testing**	• client's requirements
• **regression tests**	• refactorings, bug fixes, stability



There are many categories of software tests, as shown on the slide. Some concentrate on verifying correctness of atomic units of the software (such as unit tests), some on assuring proper high-level functionality (acceptance tests) and yet another class on providing quality assurance to the already written software – these are regression tests.

In case of business applications (we generally consider this to be any form-based application, with many screens and transitions between them), a convenient way to implement regressions tests is through **capturing** the interaction between the program and the user and **replaying** this interaction during testing, checking if the output is identical as before.

Let me quickly demonstrate what this procedure is about.



Capture-replay regression test scenario.

Note that at the beginning the set of features planned for the release is a base for preparing test scripts. Very often these scripts are informal and are a result of the user's expectations.

After (or better: during) the software is developed, the informal scenarios are checked against a candidate release. If the scenarios are fulfilled we release the software. Note that during this checking the interaction with the program can be recorded and later reused to verify if the software behaves exactly as it used to during the initial release. This process can be automated fairly easily for desktop applications, but turns out to be quite difficult for mobile applications.

Marcin, zmień sobie ten rysunek, jeśli masz czas/ ochotę :)

Motivation

Programming and testing in J2ME is **more difficult** compared to desktop programs:

- each device has slightly different hardware configuration,
- “standard” APIs from various vendors differ,
- a number of non-standard APIs and proprietary solutions exists.

Java 2 Mobile Edition **has no support** for fundamental facilities required to implement a capture-replay testing environment:

- lack of system-level support for handling events (GUI, sounds, SMS, . . .),
- lack of system-level support for simulating events.

Motivation

Programming and testing in J2ME is **more difficult** compared to desktop programs:

- each device has slightly different hardware configuration,
- "standard" APIs from various vendors differ,
- a number of non-standard APIs and proprietary solutions exists.

Java 2 Mobile Edition **has no support** for fundamental facilities required to implement a capture-replay testing environment:

- lack of system-level support for handling events (GUI, sounds, SMS, ...),
- lack of system-level support for simulating events.

In reality, mobile application development stops at the level of unit tests because of lack of tools and techniques that would allow proper acceptance and regression tests. This slide presents a list of reasons why this is the case.

First, in spite of the API specification, there is a wide range of hardware and software vendors. This leads to various incompatibilities and quirks – most notably that a piece of software must be **repeatedly tested over and over on all available device/ software configurations**.

Additionally, the standard mobile environment does not provide any support for intercepting system events (the standard Java edition has a `java.awt.Robot` class for this).

These issues made us think if it is at all possible to automate capture-replay tests in the J2ME environment.

Related work

- J2ME Unit
- Sony-Ericsson Mobile JUnit
- Motorola Gatling
- CLDC Unit
- IBM Rational Test RT
- Research In Motion – BlackBerry Fledge emulator

- J2ME Unit
- Sony-Ericsson Mobile JUnit
- Motorola Gatling
- CLDC Unit
- IBM Rational Test RT
- Research In Motion – BlackBerry Fledge emulator

We did a literature search as well as search for commercial products that would offer the functionality we required. Because of limited time I am not able to provide you with a full list of features of these products, but it should suffice to say that none of them fully implemented what we required. The most “advanced” solutions – such as IBM’s or BlackBerry’s – are still quite limited. For example, they run on software emulators (not on real devices) or are bound to a specific environment (as it is the case with BlackBerry).

Scope of work

- Design an environment for implementing acceptance, integration and regression tests for business applications in Java ME.
- Possibly automatic capture-replay.
- Should work on both actual devices and emulators (!).
- Test scripts should be easily interpreted and modified/ maintained by human operators.

Scope of work

- Design an environment for implementing acceptance, integration and regression tests for business applications in Java ME.
- Possibly automatic capture-replay.
- Should work on both actual devices and emulators (!).
- Test scripts should be easily interpreted and modified/ maintained by human operators.

We set up several goals which we wanted to achieve, outlined on the slide.

Note that while this project is very close to a technical/ engineering one, it is absolutely not trivial to solve. Our major goal was to determine if it is at all possible to design and implement an elegant capture-replay solution.

(revise from this point on)

Dynamic code injection

- One possible solution for dealing with environment's constraints.
- Intercepting **events** and injecting our custom **proxies** at several **injection points**.
- User interaction simulator.

```
1  public final class MyMidlet extends MIDlet {
2      protected void startApp()
3          throws MIDletStateChangeException {
4          // original code
5      }
6  ...
```

```
1  public final class MyMidlet extends MIDlet {
2      protected void startApp()
3          throws MIDletStateChangeException {
4          // record: before-start-event
5          try {
6              this.orig$startApp();
7              // record: after-start-event
8          } catch (Throwable t) {
9              // record: start-exception-event
10         }
11     }
12
13     private void orig$startApp()
14         throws MIDletStateChangeException {
15         // original code
16     }
17 ...
```

Code injection example – capturing `startApp` event.

```
1 public class Class1 implements CommandListener {
2 ...
3   public void commandAction(Command c, Displayable d) {
4     // Handling command event
5   }
6 ...
7 }
```

```
1 public class Class1 implements CommandListener {
2 ...
3   public void commandAction(Command c, Displayable d) {
4     RobotMERecorder.getRecorderInstance().commandIvoked(c, d);
5     // Handling command event
6   }
7 ...
8 }
```

Code injection example – capturing key events.

```
1 TextBox textBox = new TextBox("Title", "Hello world", 100, TextField.ANY);
2 textBox.addCommand(CMD_EXIT);
3 textBox.addCommand(CMD_OK);
4
5 Display.getDisplay(this).setCurrent(textBox);
```

```
1 TextBox textBox = new TextBox("Title", "Hello world", 100, TextField.ANY);
2 textBox.addCommand(CMD_EXIT);
3 RobotMERecorder.getRecorderInstance().cmdAddedToDisplayable(CMD_EXIT, textBox);
4 textBox.addCommand(CMD_OK);
5 RobotMERecorder.getRecorderInstance().cmdAddedToDisplayable(CMD_OK, textBox);
6
7 Display.getDisplay(this).setCurrent(textBox);
8 RobotMERecorder.getRecorderInstance().setCurrentDisplayable(textBox);
```

Code injection example – capturing widgets creation.

Phase 1: Recording

- Intercepting user events (key press, pointer events, text edit).
- Intercepting environment events (sounds, screen changes, Internet access, SMS).
- Transferring to remote console (through GPRS, bluetooth, IRDA or serial port).

Phase 2: Replaying

- Simulating user events (key press, pointer events, text edit).
- Assertions, injection points: Intercepting environment events (sounds, screen changes, Internet access, SMS).
- Transferring to remote console (through GPRS, bluetooth, IRDA or serial port).

Phase 3: Test maintenance

Maintenance through **human-comprehensible test scripts**.

```
<scenario>
  <event timestamp="1000">
      <displayable-changed title="Hello screen" type="TEXTBOX" />
  </event>

  <event timestamp="2000">
      <command cmdLabel="Start app" displayableTitle="Hello screen" />
  </event>

  <event timestamp="3000">
      <textbox-modification assertion="true" strongAssertion="true"
                            string="I like testing" />
  </event>
</scenario>
```

Assertions example

- TODO

Screenshots from test recording session.

```
     [java] Received logEntryId: 1
     [java] class: org.robotme.core.log.entries.LogEntry; id: 1; level: 1; timestamp: 1166
e; msg: MIDlet set to: org.example.midlet.TestTextBoxMIDlet@d590dbc; ex:
     [java] Received logEntryId: 1
     [java] class: org.robotme.core.log.entries.LogE
e; msg: Command added to displayable: javax.microedi
     [java] Received logEntryId: 1
     [java] class: org.robotme.core.log.entries.LogE
e; msg: Command added to displayable: javax.microedi
     [java] Received logEntryId: 3
     [java] class: org.robotme.core.log.entries.Disp
lse; assertion: true; msg: Displayable set to: javax
     [java] Received logEntryId: 4
     [java] class: org.robotme.core.log.entries.Textl
rue; assertion: false; msg: ; ex: ; string: I like t
     [java] Received logEntryId: 2
     [java] class: org.robotme.core.log.entries.Comm
on: false; msg: Command invoked: COMMAND; ex: ; disp
     [java] Received logEntryId: 2
     [java] class: org.robotme.core.log.entries.Comm
on: false; msg: Command invoked: COMMAND; ex: ; disp
     [java] Received logEntryId: 2
     [java] class: org.robotme.core.log.entries.Comm
on: false; msg: Command invoked: COMMAND; ex: ; disp
     [java] Received logEntryId: 2
     [java] class: org.robotme.core.log.entries.Comm
on: false; msg: Command invoked: COMMAND; ex: ; disp
     [java] Received logEntryId: 2
```

Server console.

Emulator window.

Summary

- Useful testing framework for J2ME environment **does not exist**.
- Solution: **dynamic code injection**.
- For **developers** and **clients**.

Thank you for your attention.