Capture-Replay Tests in J2ME

Testy *capture-replay* w środowisku J2ME

Marcin Zduniak Bartosz Walter Dawid Weiss

Institute of Computing Science
Poznan University of Technology

2007

Who is who

- Marcin Zduniak (the graduate)
- Bartosz Walter (thesis supervisor)
- Dawid Weiss (original concept, mentoring)

What are “software tests”?

What are “software tests”?

(Wikipedia)

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

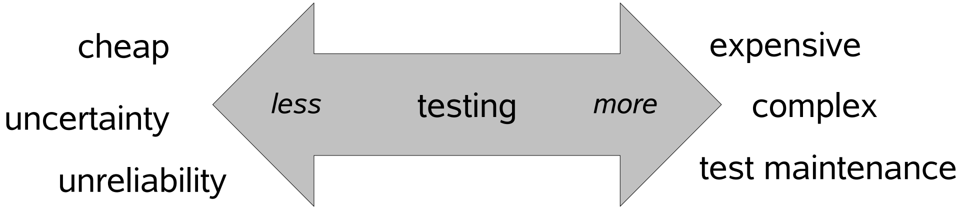
How can we “test software”?

Unit tests
Correctness of individual units of source code

Module/ integration tests
Chunks of functionality, sometimes the entire program. testing in various target environments (O/S's, processors etc.).

Acceptance tests
Compliance to customer's requirements; often manual work.

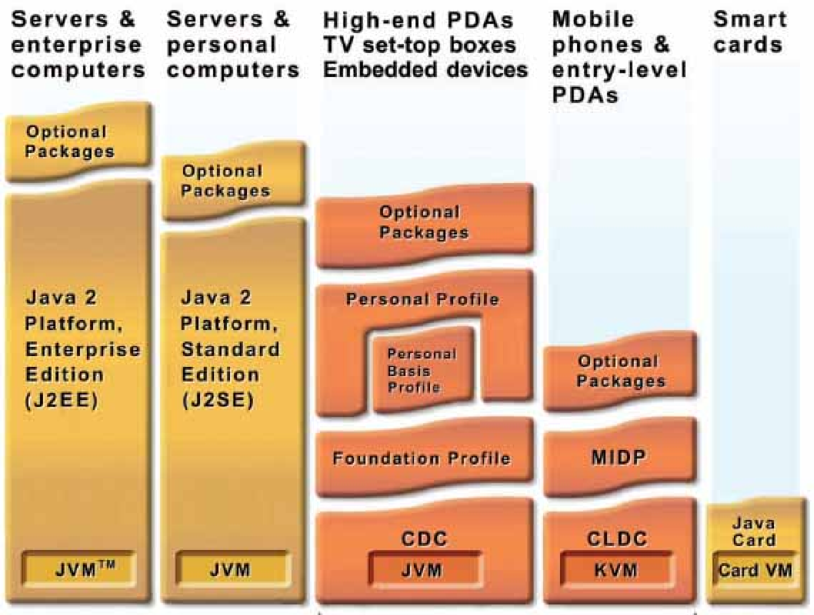
Regression tests
System stability/ correctness in response to ongoing changes.


cheap
uncertainty
unreliability
less
testing
more
expensive
complex
test maintenance

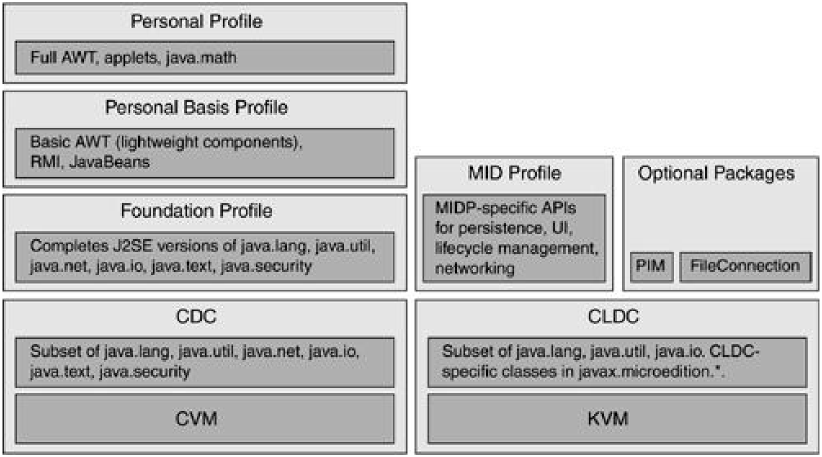
Java 2 Micro Edition

- A specification.
- A subset of Java Virtual Machine.
- A subset of standard Java library.
- **Many vendors**.

Java 2 Micro Edition


Servers & enterprise computers
Servers & personal computers
High-end PDAs TV set-top boxes Embedded devices
Mobile phones & entry-level PDAs
Smart cards
Optional Packages
Optional Packages
Optional Packages
Optional Packages
Java 2 Platform, Enterprise Edition (J2EE)
Java 2 Platform, Standard Edition (J2SE)
Personal Profile
Personal Basis Profile
Foundation Profile
MIDP
CDC
CLDC
Java Card
JVM™
JVM
JVM
KVM
Card VM

Java 2 Micro Edition


Personal Profile
Full AWT, applets, java.math
Personal Basis Profile
Basic AWT (lightweight components), RMI, JavaBeans
Foundation Profile
Completes J2SE versions of java.lang, java.util, java.net, java.io, java.text, java.security
CDC
Subset of java.lang, java.util, java.net, java.io, java.text, java.security
CVM
MID Profile
MIDP-specific APIs for persistence, UI, lifecycle management, networking
Optional Packages
PIM
FileConnection
CLDC
Subset of java.lang, java.util, java.io. CLDC-specific classes in javax.microedition.*.
KVM

Programming in J2ME

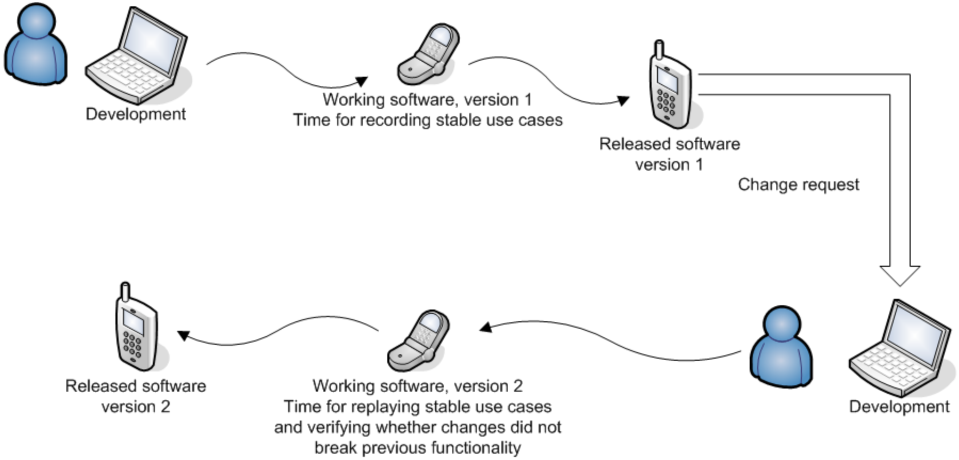
- Each mobile device has different hardware.
- Different KVM implementations (and bugs).
- “Standard” APIs implemented in different ways.
- A number of non-standard APIs and proprietary solutions.
- No system-level support for application testing.

Conclusion:
programming and testing is difficult in J2ME.

Project scope

1. Focus on capture-replay tests (GUI and other events).
2. Should facilitate integration and regression tests in J2ME.
3. Should work on emulators and actual devices.

Example of use


Development
Working software, version 1
Time for recording stable use cases
Released software version 1
Change request
Development
Working software, version 2
Time for replaying stable use cases and verifying whether changes did not break previous functionality
Released software version 2

Capture-replay and regression testing.

Related projects

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

And the ultimate answer is. . .

RobotME

(of course the ultimate question still being “what’s 6×9?”)

The core idea

- Follow the regular capture-replay pattern.
- Cater for missing “robot” API by modifying the software at the **bytecode** level.
- Verify replay-phase correctness by analysis of captured events.

Dynamic code injection

- Identify places which should generate an event (“injection points”).
- Intercept parameters at injection points, injecting custom proxies.

Injection points: subclassing

Custom inheritance from system classes (subclassing).
`Form`, `Canvas`, `MIDlet`...

```java
public class MyMidlet extends MIDlet {
   protected void startApp() throws MIDletStateChangeException {
      // application code here.
   }
}
```

We need to intercept the call to `startApp()` method.

- Make MyMidlet a subclass of RobotMIDlet?

```java
public class MyMidlet extends RobotMIDlet {
```

all methods virtual, multi-level inheritance

- Make `MyMidlet` a subclass of `RobotMIDlet`?

```java
public class MyMidlet extends RobotMIDlet {
```

all methods virtual, multi-level inheritance

- Make a custom subclass `RobotMIDlet$1` extend `MyMidlet`?

```java
public class RobotMIDlet$1 extends MyMidlet {
```

finalized classes, multi-level inheritance

- Make MyMidlet a subclass of RobotMIDlet?

```java
public class MyMidlet extends RobotMIDlet {
```

all methods virtual, multi-level inheritance

- Make a custom subclass RobotMIDlet$1 extend MyMidlet?

```java
public class RobotMIDlet$1 extends MyMidlet {
```

finalized classes, multi-level inheritance

- Use delegation pattern and alter the code of startApp()?

```java
public class MyMidlet extends MIDlet {
   protected void startApp() throws MIDletStateChangeException {
      RobotME.event(this, STARTAPP_BEFORE);
      try {
         // application code here.
      } finally {
         RobotME.event(this, STARTAPP_AFTER);
      }
   }
}
```

code bloat, goto, architectural flaws

Injection points: references

Direct use of an object (reference tracking).

Command, Item

```
public class MyApplication {
   private static final Command MY_COMMAND =
      new Command("Cmd label", Command.OK, 1);

   public MyApplication() {
      Form f = new Form("Title");
      f.addCommand(MY_COMMAND);
   }
}
```

We need to track the identity (and value) of commands.

- Substitute all constructors of `Form` with a custom proxy?

```
Form f = new RobotME$Form("Title");
```

clashes with custom inheritance

- Substitute all constructors of Form with a custom proxy?

```
Form f = new RobotME$Form("Title");
```

clashes with custom inheritance

- Intercept all calls to addCommand()?

```
f.addCommand(RobotME.addCommand(f, MY_COMMAND));
```

not too bad

Injection points: listeners

Listeners (system callbacks).
`ItemStateListener`, `CommandListener`, `ItemCommandListener`

```java
public class MyForm extends Form implements CommandListener {
   private final Command ADD_COMMAND;

   public MyClass {
      this.ADD_COMMAND = new Command("ADD", Command.OK, 1);
      this.setCommandListener(this);
   }

   public void commandAction(Command c, Displayable d) {
      // event handling code.
      if (c == ADD_COMMAND) {
         // ...do something.
      }
   }
...
```

We need to track (and stimulate) commands for the listener. Note there is only **one** listener on a `Form`.

- Intercept all calls to `setCommandListener()`?

```java
this.setCommandListener(RobotME.setCommandListener(this, command));
```

What if somebody uses “==” to compare listeners?

- Intercept all calls to `setCommandListener()`?

```java
this.setCommandListener(RobotME.setCommandListener(this, command));
```

  What if somebody uses “==” to compare listeners?

- How to remember the command received (if it’s a dynamic reference)?

  The reference changes between runs.

- Intercept all calls to `setCommandListener()`?

```java
this.setCommandListener(RobotME.setCommandListener(this, command));
```

  What if somebody uses “==” to compare listeners?

- How to remember the command received (if it’s a dynamic reference)?

  The reference changes between runs.

- How to generate an identical event dynamically?

  The event should match the original `Display`/ `Form` pair.

Examples of injected code

Original code (fragment):

```java
final Form form = new Form("Questionnaire");
...
form.addCommand(CMD1_EXIT);
form.addCommand(CMD2_OK);
form.setCommandListener(this);

Display.getDisplay(this).setCurrent(form);
```

Code modified for the recording phase:

```java
Form form = new Form("Questionnaire");
...
form.addCommand(b); // NOTE: variable names have been obfuscated.
RobotMERecorder.getRecorderInstance().commandAddedToDisplayable(b, form);
form.addCommand(c);
RobotMERecorder.getRecorderInstance().commandAddedToDisplayable(c, form);
form.setCommandListener(this);
Display.getDisplay(this).setCurrent(form);
RobotMERecorder.getRecorderInstance().setCurrentDisplayable(form);
```

Code modified for the replay phase:

```java
Form form = new Form("Questionnaire");
form.addCommand(b);
RobotMEReplaying.getReplayingInstance().commandAddedToDisplayable(b, form);
form.addCommand(c);
RobotMEReplaying.getReplayingInstance().commandAddedToDisplayable(c, form);
form.setCommandListener(this);
RobotMEReplaying.getReplayingInstance()
   .commandListenerSetOnDisplayable(this, form);
Display.getDisplay(this).setCurrent(form);
RobotMEReplaying.getReplayingInstance().setCurrentDisplayable(form);

RobotMEReplaying.getReplayingInstance().startReplaying();
```

A teraz przerwa dla miłośników granul.

A teraz przerwa dla miłośników granul.

At the bytecode level

Java can be quite pleasant (and surprising!)

Java in assembler mode ;)

- The stack.

Java in assembler mode ;)

- The stack.
- Local variables.

Java in assembler mode ;)

- The stack.
- Local variables.
- Opcodes and their mnemonics.

Java in assembler mode ;)

- The stack.
- Local variables.
- Opcodes and their mnemonics.
- Code verification.

Reverse-engineering Java code

```java
public static final void method(int i)
{
   System.out.println(i);
}
```

$\downarrow$

Reverse-engineering Java code

```java
public static final void method(int i)
{
   System.out.println(i);
}
```

$\downarrow$

```
public static final void method(int i)
{
   getstatic #16 <Field PrintStream System.out>
   iload_0
   invokevirtual #22 <Method void PrintStream.println(int)>
   return
}
```

Reverse-engineering Java code

```java
   private final long sum(int a, int b) {
      return a + b;
   }

   public final void method() {
      System.out.println(sum(2, 50));
   }
```

Reverse-engineering Java code

```
private final long sum(int a, int b) {
   return a + b;
}

public final void method() {
   System.out.println(sum(2, 50));
}
```

```
private final long sum(int a, int b) {
   iload_1
   iload_2
   iadd
   i2l
   lreturn
}
public final void method() {
   getstatic #20 <Field PrintStream System.out>
   aload_0
   iconst_2
   bipush 50
   invokespecial #26 <Method long sum(int, int)>
   invokevirtual #28 <Method void PrintStream.println(long)>
   return
}
```

Reverse-engineering Java code

```java
public final void method(int i) {
   switch (i) {
      case 1:
      case 25:
      case -5:
      case 1128:
         break;
      default:
         throw new RuntimeException();
   }
}
```

Reverse-engineering Java code

```java
public final void method(int i) {
   switch (i) {
      case 1:
      case 25:
      case -5:
      case 1128:
         break;
      default:
         throw new RuntimeException();
   }
}
```

```
public final void method(int i)
{
   0 0:iload_1
   1 1:lookupswitch default 47
        -5: 44
         1: 44
        25: 44
      1128: 44
   2 44:goto 55
   3 47:new #16 <Class RuntimeException>
   4 50:dup
   5 51:invokespecial #18 <Method void RuntimeException()>
   6 54:athrow
```

Reverse-engineering Java code

```java
public final int method(int i) {
   try {
      if (i == 0) {
         return 0;
      }
      if (i == 1) {
         return 1;
      }
      if (i == 2) {
         return 2;
      }
   } finally {
      System.out.println("aaa");
      System.out.println("bbb");
      System.out.println("ccc");
   }
   return -1;
}
```

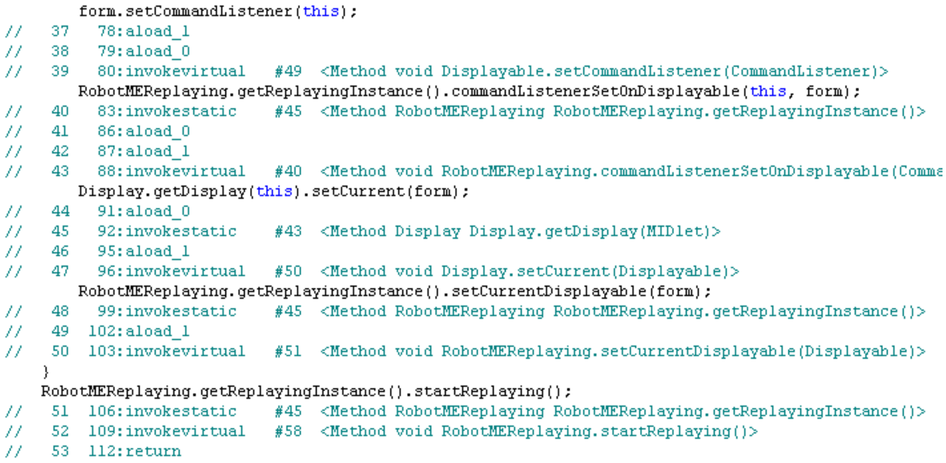
What will this compile into?

SUN's compiler

```
  public final int method(int i) {
     if (i == 0) {
        System.out.println("aaa"); System.out.println("bbb");
        System.out.println("ccc");
        return 0;
     }
     if (i == 1) {
        System.out.println("aaa"); System.out.println("bbb");
        System.out.println("ccc");
        return 1;
     }
     if (i == 2) {
        System.out.println("aaa"); System.out.println("bbb");
        System.out.println("ccc");
        return 2;
     } else {
        System.out.println("aaa"); System.out.println("bbb");
        System.out.println("ccc");
        return -1;
     }

exception_handler:
        System.out.println("aaa"); System.out.println("bbb");
        System.out.println("ccc");
     throw exception;
  }
```

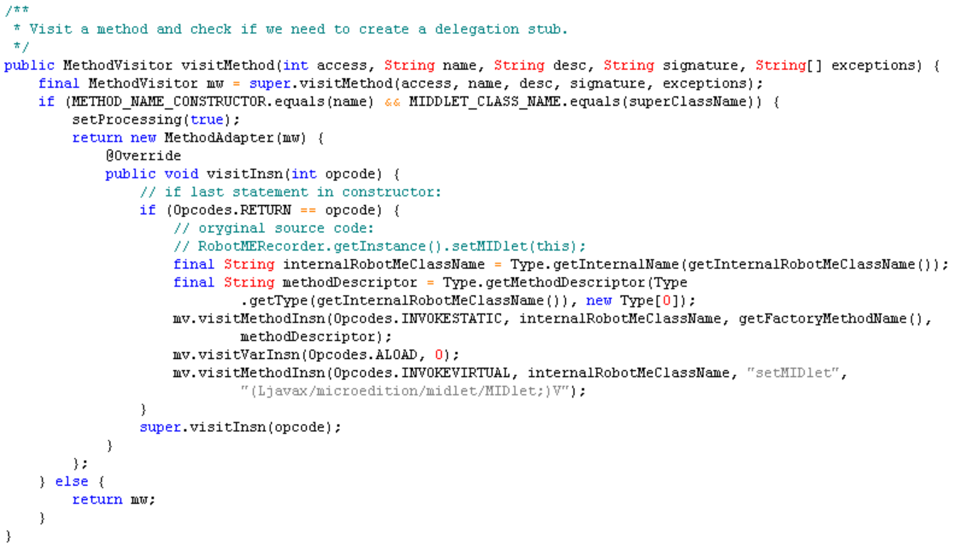
IBM's compiler

```java
   public final int method(int i) {
      int j;
      if (i != 0)
         goto _L13;
      jsr local;
      j = 0;
      return j;
_L13:
      if (i != 1) goto _L27;
      jsr local;
      j = 1;
      return j;
_L27
      if (i != 2) goto _L2;
      jsr local;
      j = 2;
      return j;
_L2:
      jsr local;
      return -1;
local:
      System.out.println("aaa");
      System.out.println("bbb");
      System.out.println("ccc");
      ret;
   }
```



```
        form.setCommandListener(this);
//   37   78:aload_1
//   38   79:aload_0
//   39   80:invokevirtual   #49  <Method void Displayable.setCommandListener(CommandListener)>
        RobotMEReplaying.getReplayingInstance().commandListenerSetOnDisplayable(this, form);
//   40   83:invokestatic    #45  <Method RobotMEReplaying RobotMEReplaying.getReplayingInstance()>
//   41   86:aload_0
//   42   87:aload_1
//   43   88:invokevirtual   #40  <Method void RobotMEReplaying.commandListenerSetOnDisplayable(Comm
        Display.getDisplay(this).setCurrent(form);
//   44   91:aload_0
//   45   92:invokestatic    #43  <Method Display Display.getDisplay(MIDlet)>
//   46   95:aload_1
//   47   96:invokevirtual   #50  <Method void Display.setCurrent(Displayable)>
        RobotMEReplaying.getReplayingInstance().setCurrentDisplayable(form);
//   48   99:invokestatic    #45  <Method RobotMEReplaying RobotMEReplaying.getReplayingInstance()>
//   49  102:aload_1
//   50  103:invokevirtual   #51  <Method void RobotMEReplaying.setCurrentDisplayable(Displayable)>
    }
    RobotMEReplaying.getReplayingInstance().startReplaying();
//   51  106:invokestatic    #45  <Method RobotMEReplaying RobotMEReplaying.getReplayingInstance()>
//   52  109:invokevirtual   #58  <Method void RobotMEReplaying.startReplaying()>
//   53  112:return
```

Bytecode-level changes.



```java
/**
 * Visit a method and check if we need to create a delegation stub.
 */
public MethodVisitor visitMethod(int access, String name, String desc, String signature, String[] exceptions) {
    final MethodVisitor mw = super.visitMethod(access, name, desc, signature, exceptions);
    if (METHOD_NAME_CONSTRUCTOR.equals(name) && MIDDLET_CLASS_NAME.equals(superClassName)) {
        setProcessing(true);
        return new MethodAdapter(mw) {
            @Override
            public void visitInsn(int opcode) {
                // if last statement in constructor:
                if (Opcodes.RETURN == opcode) {
                    // oryginal source code:
                    // RobotMERecorder.getInstance().setMIDlet(this);
                    final String internalRobotMeClassName = Type.getInternalName(getInternalRobotMeClassName());
                    final String methodDescriptor = Type.getMethodDescriptor(Type
                            .getType(getInternalRobotMeClassName()), new Type[0]);
                    mv.visitMethodInsn(Opcodes.INVOKESTATIC, internalRobotMeClassName, getFactoryMethodName(),
                            methodDescriptor);
                    mv.visitVarInsn(Opcodes.ALOAD, 0);
                    mv.visitMethodInsn(Opcodes.INVOKEVIRTUAL, internalRobotMeClassName, "setMIDlet",
                            "(Ljavax/microedition/midlet/MIDlet;)V");
                }
                super.visitInsn(opcode);
            }
        };
    } else {
        return mw;
    }
}
```

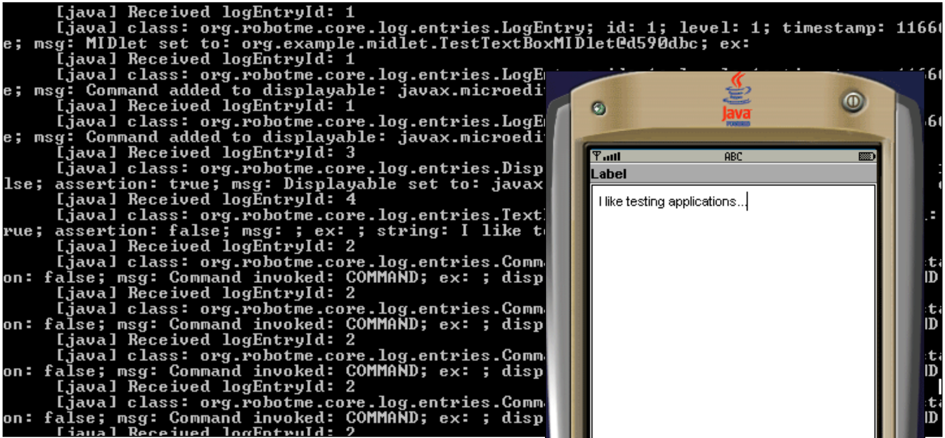
ASMLib is used for preprocessing bytecode (statically).

Test maintenance

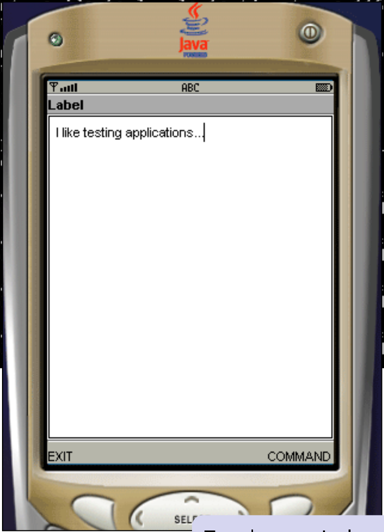
Maintenance through **human-comprehensible test scripts**.

```
<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>
```

Time for a live demo!



```
     [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.Text
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
```



I like testing applications...

Server console.

Emulator window.

Summary

- Testing is difficult in J2ME.
- Bytecode manipulation can provide a substitute for the required API functions.

- The prototype a bit gritty, but functional.

Little victories

- Springer LNCS publication (10th BIS conference).
- UAM Foundation — “Pomysł na biznes” competition.

Thank you for your attention.