

Capture Replay Testing

endtest

Web



and many others...



UI TESTING FOR ANDROID
espresso

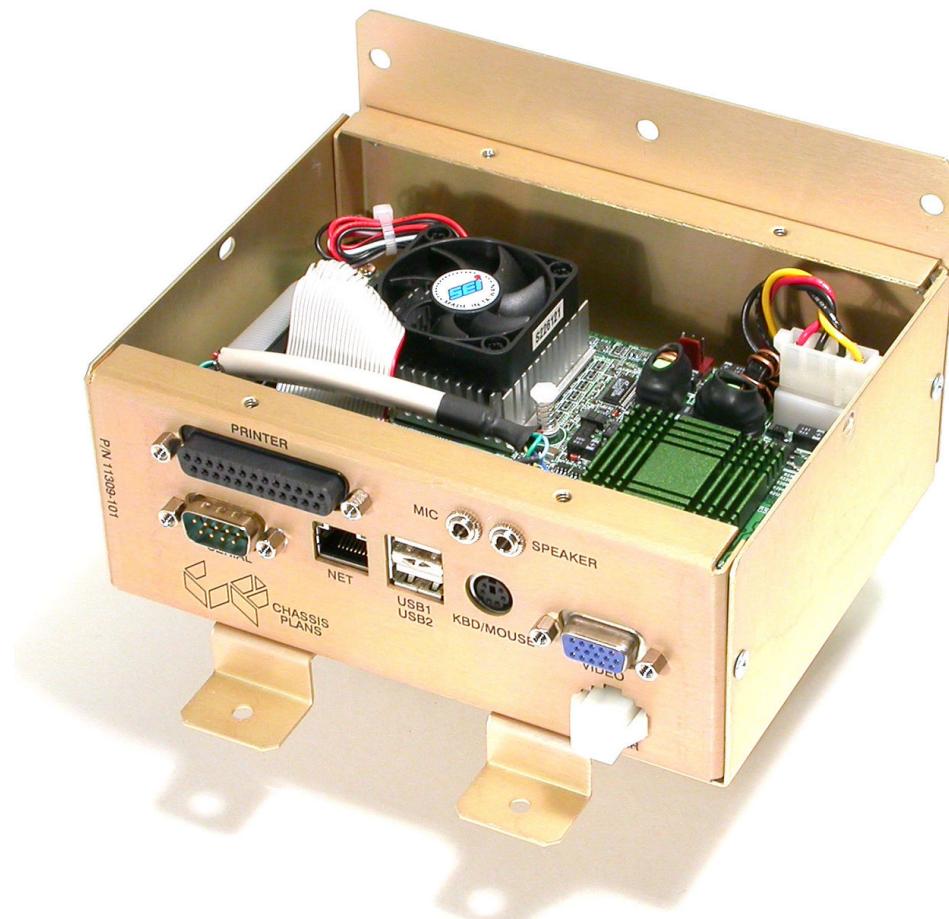
appium
mobile automation made awesome



Mobile

Robotium

Origins



Nowadays

- GUI testing
- Mobile device testing
- Web application testing
- ...

Basic idea (1)

- Record interactions with the system
 - User perspective interactions mostly



Basic idea (2)

- Play the previous recording



Easy and great?

- Easy: absolutely!
 - Popular for web and mobile development
 - Easy for non-experts as well!
- When I look around: not much other types of testing going on
 - But these capture/replay tests are extremely brittle!
 - Tiny changes in the UI or in the functionality break them
 - They are typically slow...
 - How much feedback do they really give? (what if you were to throw mutation testing against these tests?)



2nd testing assignment (1)

- Use a capture/replay tool on at least one software system (Selenium, Espresso, Appium, Robotium, ... all are fine!)
 - Capture your own scenario *or*
 - Use an existing scenario
- Describe at least 1 instance of brittleness of the tests
 - Look at the history of the system and what the test does

2nd testing assignment (2)

- Read the paper(s) mentioned in the assignment
- Optional: read blog post(s) on pros/cons of capture replay
- Write a 4-page report explaining to your manager why your fictive company should (not) use capture/replay tools. Argument your decision as well as possible.
 - Think: software engineering experience, bugs, brittleness, type of application, etc.

Not only “capturing” works...

- Linear scripting
 - You “script” tests at a very high level
 - And then replay them

```
import java.util.regex.Pattern;
import java.util.concurrent.TimeUnit;
import org.junit.*;
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;
```

At the top of your Selenium firefox Driver class add the following namespaces

```
public class testClass {
    private WebDriver driver;
    @Before
    public void setUp() throws Exception {
        driver = new FirefoxDriver();
    }
}
```

Next under the main section of your SeleniumDriver class create an instance of the FirefoxDriver

```
@Test
public void testtestclass() throws Exception {
    driver.get("http://www.wikipedia.org/");
    Assert.assertEquals("Wikipedia", driver.getTitle());
    Assert.assertEquals("English", driver.findElement(By.cssSelector("strong")).getText());
    driver.findElement(By.cssSelector("strong")).click();
    Assert.assertEquals("Wikipedia, the free encyclopedia", driver.getTitle());
}
```

Verify and check the correct page is loaded using assertion with the driver's getTitle , getText etc methods.

```
@After
public void tearDown() throws Exception {
    driver.quit();
}
```

Help to kill all the WebDriver instances

Linear scripting pros/cons

- Fast to create
- Flexible
- Requires programming skills
- Fragile
- Hard to maintain (not really modular, number of scripts, ...)



Improved scripting approaches

- Modular scripting → build a library with most common scripting tasks that can be reused
 - E.g., Selenium
- Data driven testing

	A	B	C	D	E
1	Test Case	Number 1	Operator	Number 2	Expected
2	Add 01	1	+	2	3
3	Add 02	1	+	-2	-1
4	Sub 01	1	-	2	-1
5	Sub 02	1	-	-2	3
6	Mul 01	1	*	2	2
7	Mul 02	1	*	-2	-2
8	Div 01	2	/	1	2
9	Div 02	2	/	-2	-1
10					