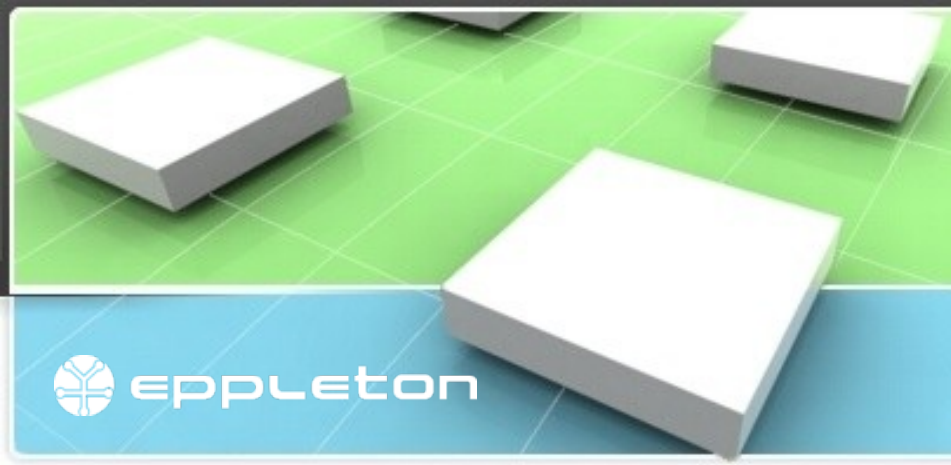
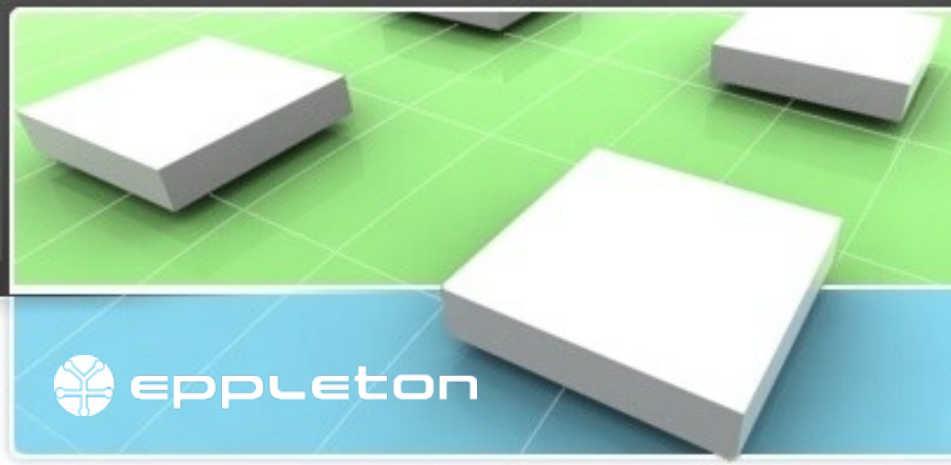
NetBeans Rich Client Platform Simpletests

Anton Epple
Eppleton IT Consulting



Unit tests for RCP Applications

- Introduced in 6.5
- Before: Xtest infrastructure
  - Unit & Functional Tests
  - hard to set up for custom RCP projects
- Modifications:
  - SimpleTest is based on Modules itself
  - => Easy setup



Unit tests for RCP Applications

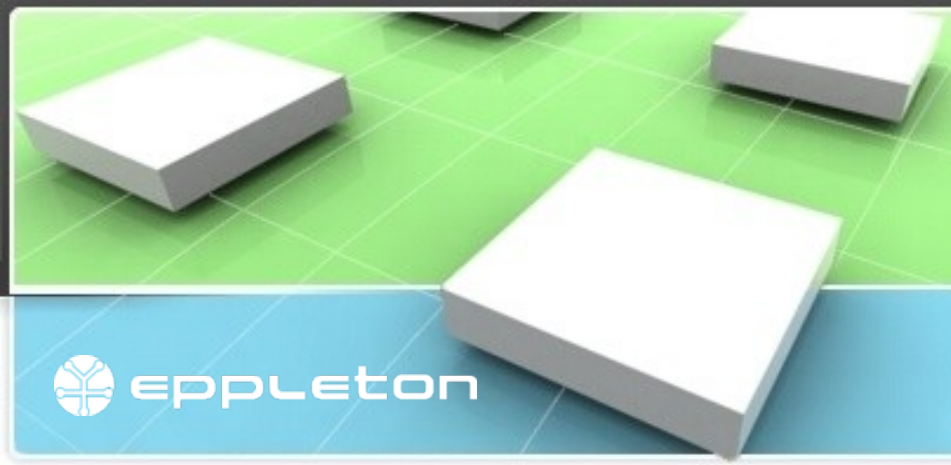
- JUnit 4
- NetBeans Extensions to Junit (NBJUnit):
  - compare files via assertFile
  - create working directories for testcases
  - write to log files
  - compare log files against reference (golden) files, etc.
- Jemmy for functional Tests
- NetBeans Extensions for Jemmy: Jelly

- Pattern for Dependency Injection with lookups:

```
abstract class DialogDisplayer {
    public abstract void notify(String msg);

    public static DialogDisplayer getDefault() {
      return Lookup.getDefault().lookup(DialogDisplayer.class);
    }
  }
```

- In Test:

```
public class MyTest extends NbTestCase {
    public MyTest(String name) {
        super(name);
    }
    protected void setUp() throws Exception {
        super.setUp();
        org.netbeans.junit.MockServices.setServices(MockDD.class);
    }
}
```



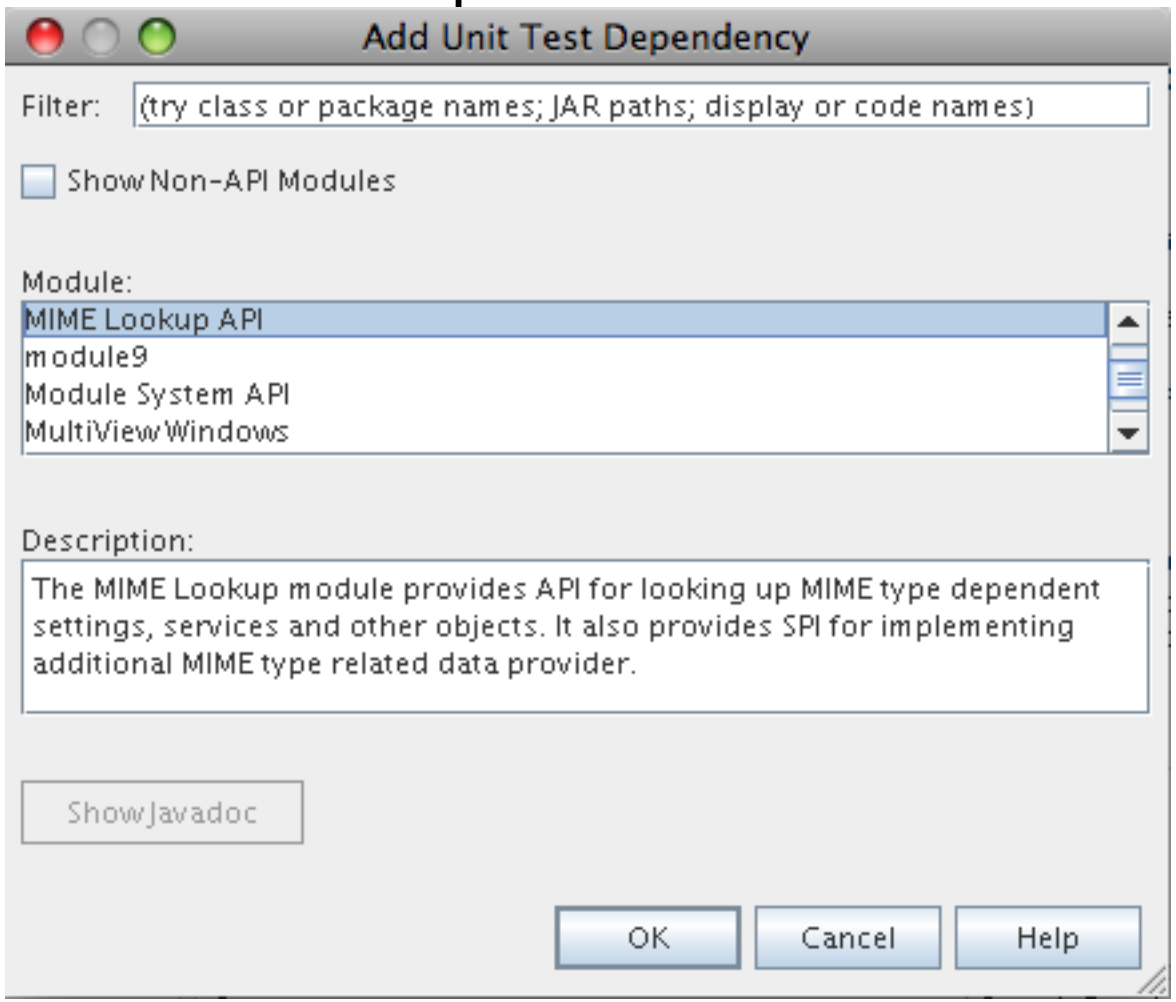
Unit tests for RCP Applications

- Setting up more than one service:

```
public class MyTest extends NbTestCase {
 static {
   System.setProperty("org.openide.util.Lookup", Lkp.class.getName());
 }

 public MyTest(String name) {
   super(name);
 }

 public static final class Lkp extends
org.openide.util.lookup.AbstractLookup {
    public Lkp() {
        this(new org.openide.util.lookup.InstanceContent());
    }

    private Lkp(org.openide.util.lookup.InstanceContent ic) {
        super(ic);
        ic.add(new DD());
```

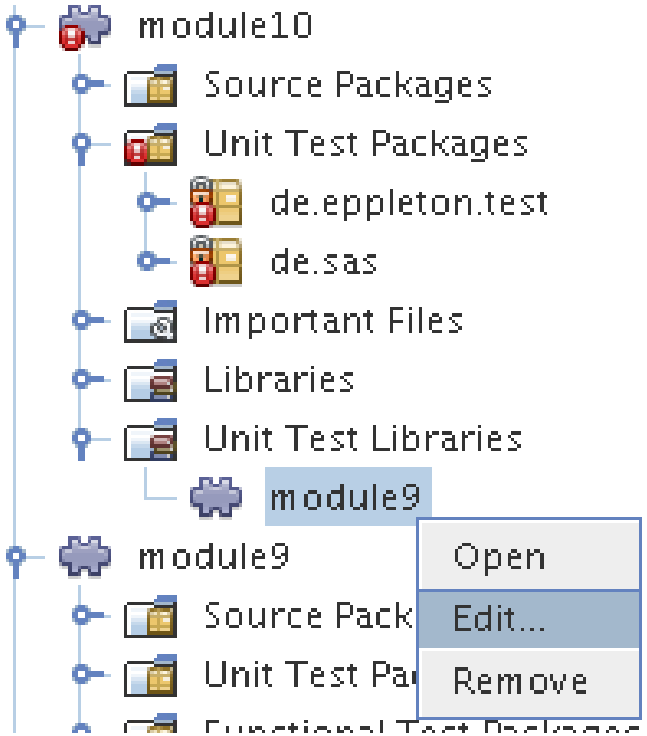
Unit Test Dependencies

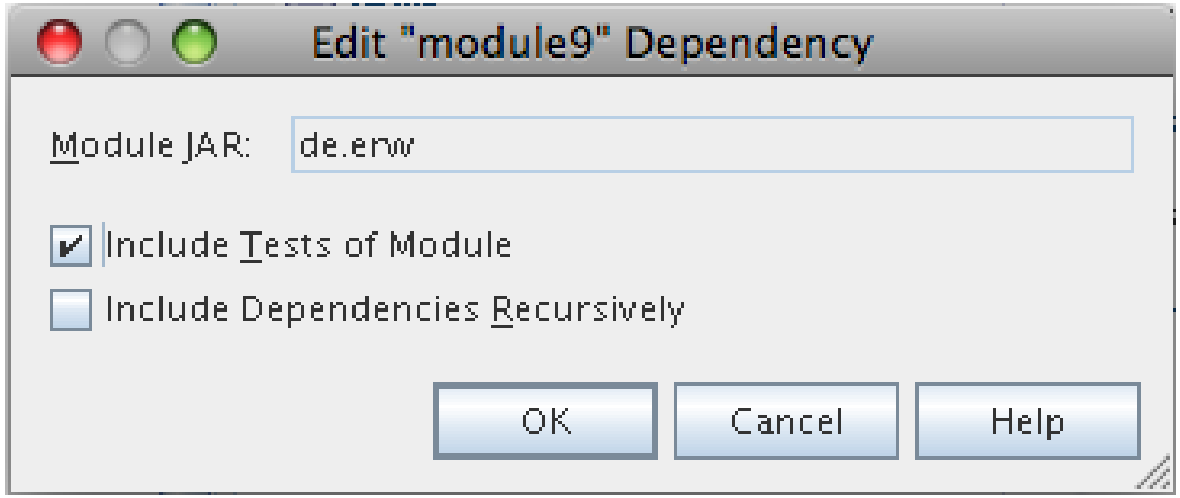
- Unit Tests can have dependencies on other Modules:

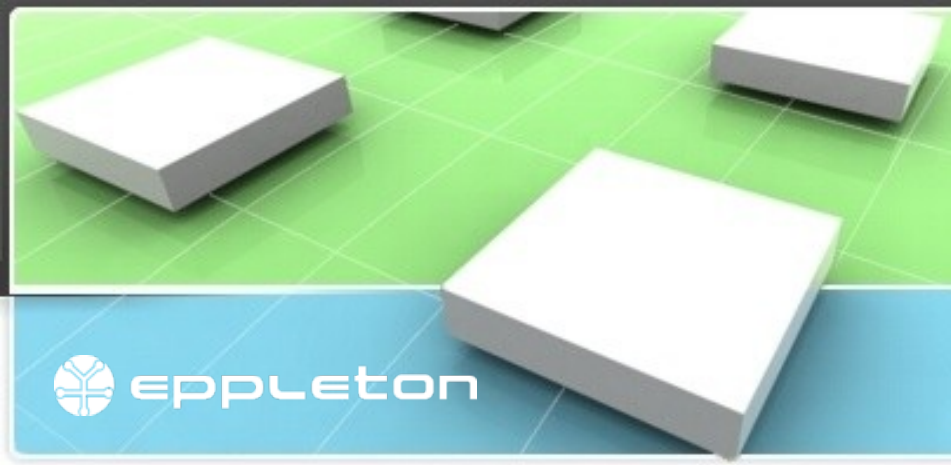


Unit Test Dependencies

- ...and their Test Classes (Demo NurEinTest):

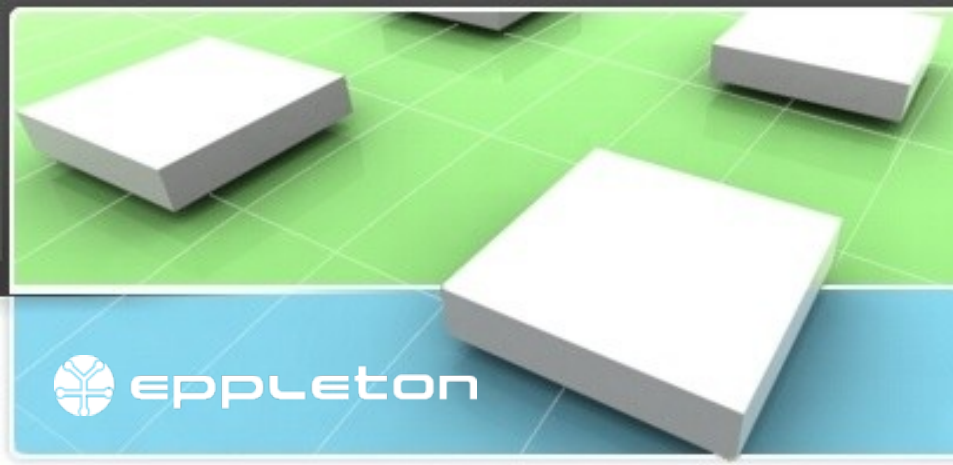






Unit Test Dependencies

- Considerations for adding Test Mock & Helper classes to modules:
- API Design: Programming against Interfaces just for testing problematic in terms of API evolution
    - Can't add new methods in Interface without breaking compatibility
- Alternatively: Testability part of the API
    - → Testable binary modules (Platform Chaining)



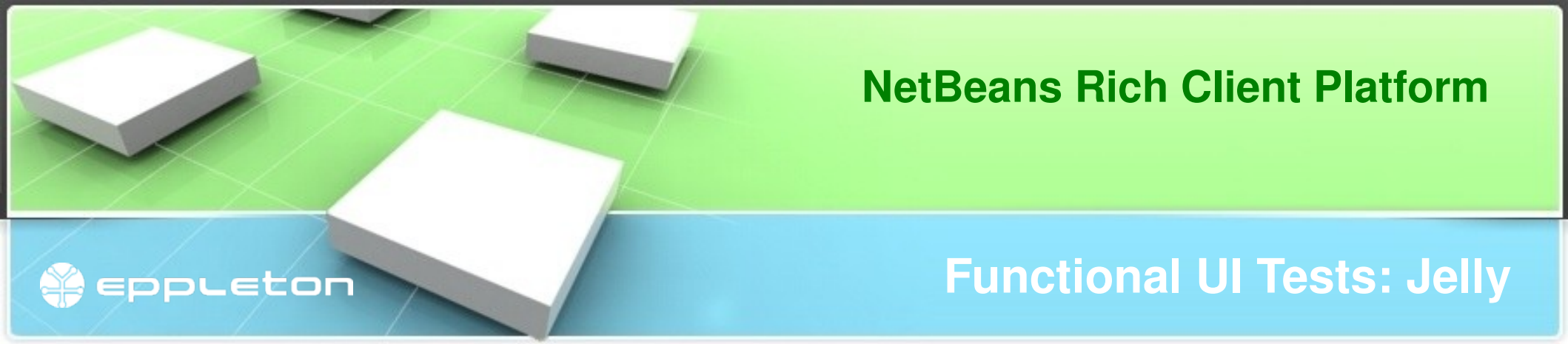
Unit Test in AWT Thread

- NBTestCase supports running in AWT Dispatch Thread:

```
public class MyTest extends NbTestCase {
      @override
      protected boolean runInEQ () {
        return true;
      }
   }
```

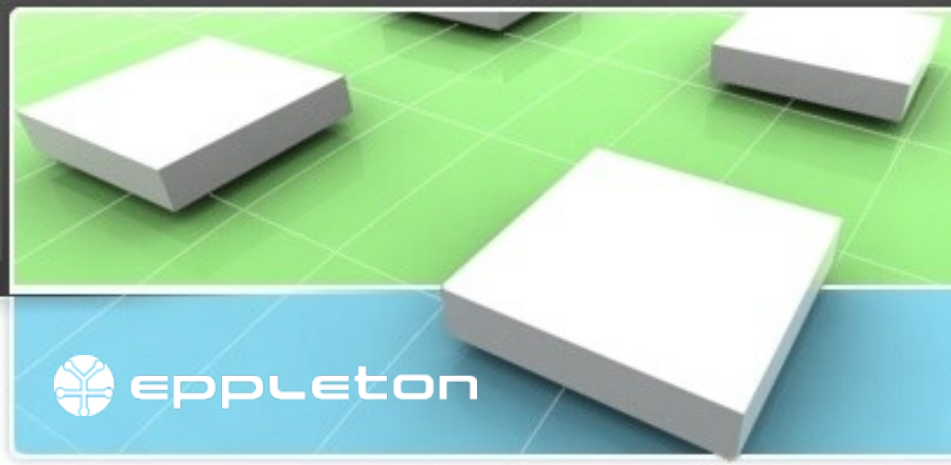
More Tips:

- http://openide.netbeans.org/tutorial/test-patterns.html



Functional UI Tests: Jelly

- NetBeans supports Jemmy for automated ui testing:
  - https://jemmy.dev.java.net/
  - Independent of test harness
- Jelly is a set of Extensions to Jemmy specifically to test NB Platform applications
- Jelly Tests are based on JUnit
- JellyTestCase extends NBTestCase
- Setup in project is simple but not straightforward...

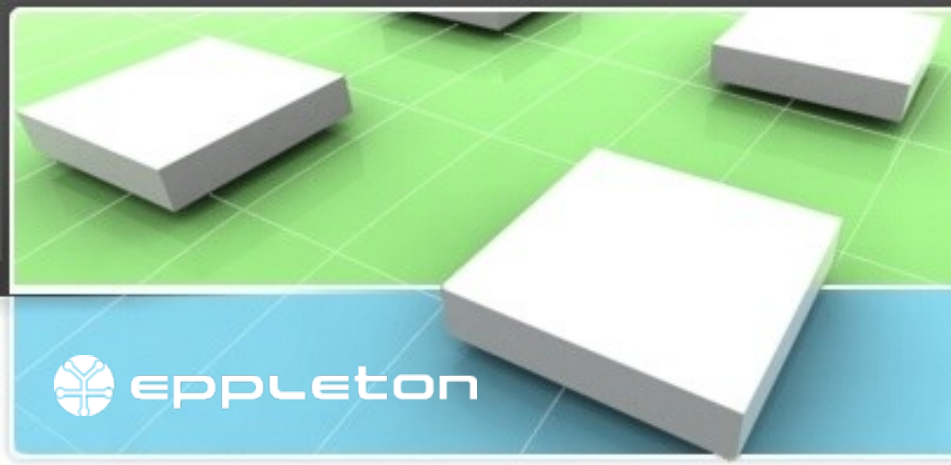


Functional UI Tests: Jelly

- Required Modules: Cluster harness, MultiView Windows + Visual Library from platform9
- In Modules test dir: folder qa-functional/src
- IDE restart adds 2 Nodes to Module:
  - Functional Test Packages
  - Functional Test Libraries
- Required Functional Test Libraries: jemmy, nbjunit, jellytools platform and junit4

- In project.xml (Project Metadata) add the recursive tag to the nbjunit dependency:

```
<test-dependency>
    <code-name-base>org.netbeans.modules.nbjunit</code-name-base>
   <compile-dependency/>
   <recursive/>
</test-dependency>
```



Functional Tests: JellyTestCase

```
public class SampleTest extends JellyTestCase {

    public SampleTest(String name) {
        super(name);
    }

    public static Test suite() {
        Configuration testConfig =
NbModuleSuite.createConfiguration(SampleTest.class);
        testConfig.addTest("testSomething");
        testConfig.clusters(".*").enableModules(".*");
        testConfig.gui(true);
        return NbModuleSuite.create(testConfig);
    }

    public void setUp() {
        System.out.println("########  "+getName()+"  #######");
    }

    public void testSomething() {
        new Action("File|Some Action", null).perform();
    }
}
```

Demo

Considerations:

- Possible to run automated UI test with Jemmy
- Downside:
  - Not easy to write & maintain
  - May cause false alarms (e.g. renamed Menu Items aren't necessarily Bugs)
- Manual Testing with test scripts

commit-validation

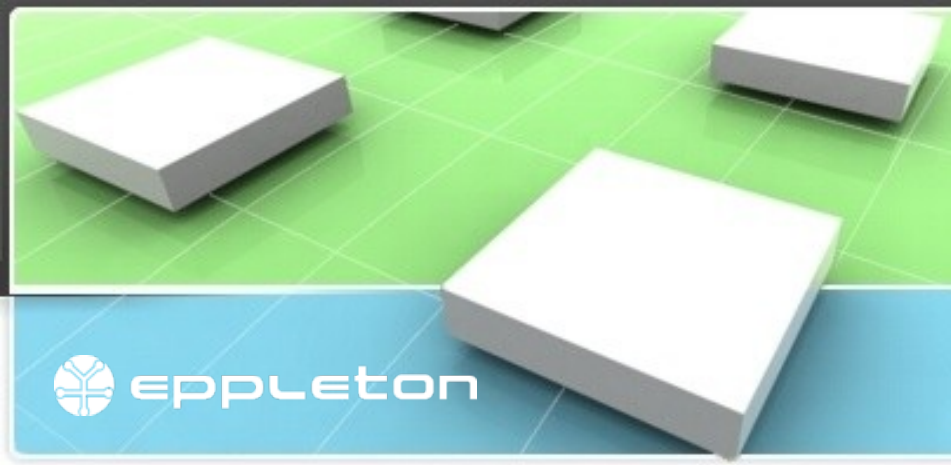
- Small Test suite runs before every commit (5 minutes)
- Large test suite runs on CI server

Example:

http://www.netbeans.org/community/guidelines/commit.html

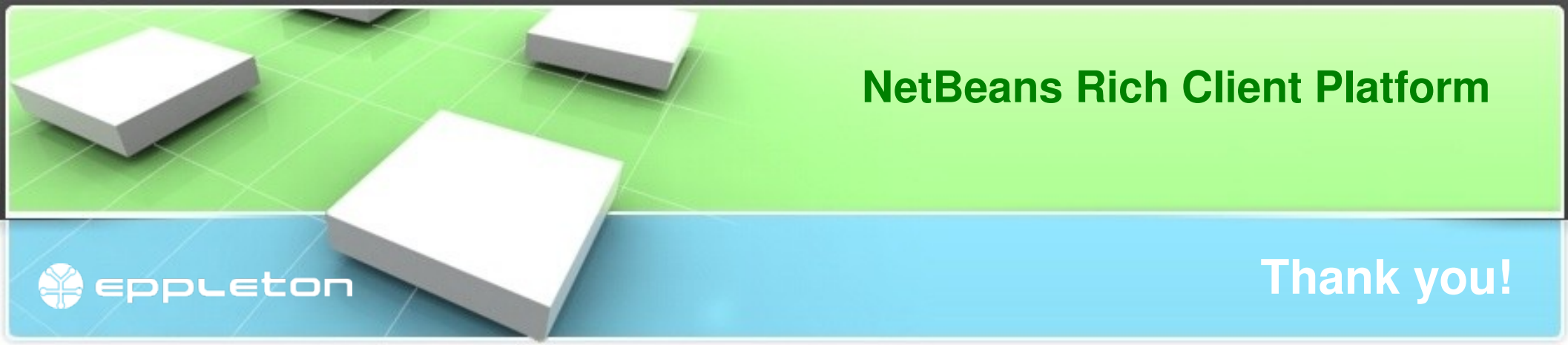
Besides Testing:

- PMD (PMD scans Java source code for Bugs, duplications, ...)
- FindBugs (static code analysis)
- CheckStyle (Enforce Coding Standards)
- Lint4j (find Locking & Threading Issues)
- SQE (contains all 4) in a NB Plugin (sqe.kenai.org)



Links & Resources

- http://wiki.netbeans.org/DevFaqUsingSimpletests
- http://wiki.netbeans.org/FitnessTestsWithoutX
- http://wiki.netbeans.org/XTestReplacementCookBook



Thank you!

Q & A