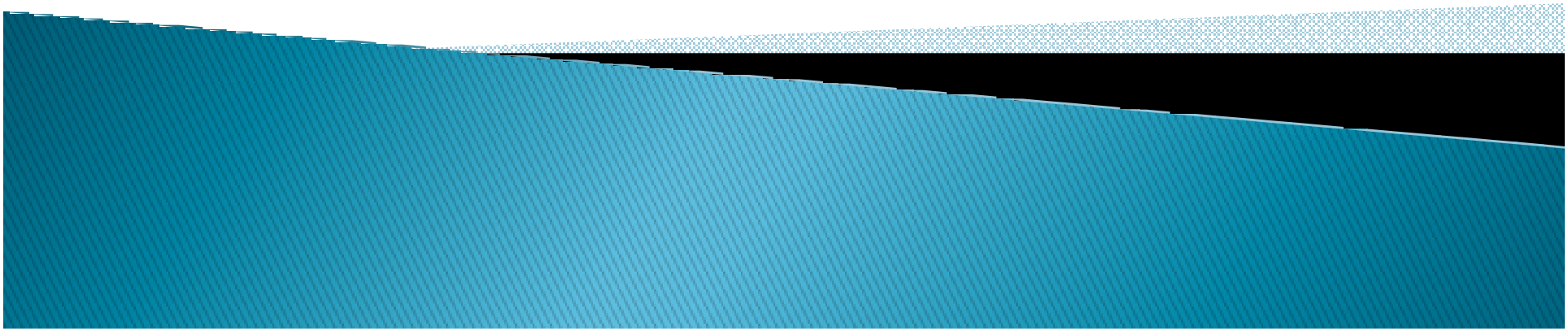
La place des tests

Cours LPSIL

2013



Agenda

- Test Types
- Tooling and Strategy

Questions?

- This slide is not at the end
- Ask questions when they come up. If it's out of place, we'll list it on a dedicated whiteboard page

Quality

- What is it?

Quality

- What is it?
- No defect ?

Quality

- What is it?
- No defect?
- or….
- Known defects?

Quality

- In theory, we'd like software with zero bug
- However, bug-free software is hardly achievable, given:
  - Time and financial constraints
  - Human limit vs. System size
  - Pressure of the competition

Quality

- Quality's purposes are to:
  - Know and document bugs
  - Verify them for regression
  - Find workarounds
  - Feed more requirements (bugs show product usage)

Quality Process

- Measurements, indicators, monitoring
  - -> Nov. 15th
- Defect management
  - -> Oct. 4th
- Testing
  - -> Today

Development lifecycles

- Several methodologies widely used:
  - Waterfall
  - Iterative
  - a combination of both (short iterative V-cycles)
  - eXtreme Programming, etc.
- Each allocates a large amount of time to testing phases
- Pareto law: 80% of the code written to handle error cases.

Types of tests

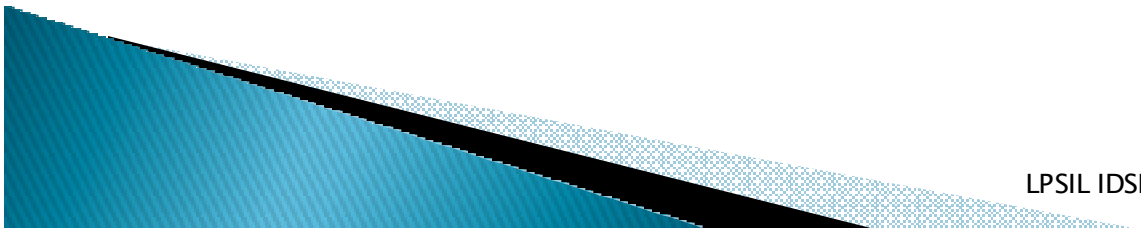
- Exercice

Types of tests

- Unit Tests
- Integration Tests
- GUI Tests
- Non-regression Tests
- Coverage Tests
- Load Tests
- Stress Tests
- Performance Tests
- Scalability Tests
- Reliability Tests
- Volume Tests
- Volume Tests
- Usability Tests
- Security Tests
- Recovery Tests
- L10N/I18N Tests
- Accessibility Tests
- Installation/Configuration Tests
- Documentation Tests
- Platform testing
- Samples/Tutorials Testing
- Code inspections
- …

Unit testing

- Purpose: test a single class, or even a single method
- Why?
  - Contract compliance
  - Regression
  - Bug isolation
  - Documentation (test code is a usage sample)



Unit testing

- How to perform this type test?
  - Invoke each method of the class
  - With various, representative sets of data
  - Capture the returned values
  - Check against expected results
  - Record success / failure

Unit testing

- How to automate?
  - Write a java method for each tested method
  - Have it perform with various data sets
  - Dump results in some file (E.g.: xml)
  - Report from result file

- the most interesting part is the body of the test method. The rest would be best provided by a framework

Unit testing

- Environment Example: jUnit
- Provides:
  - Test base class, with assertion utilities
    - assertTrue, assertNotNull, assertEquals, etc.
  - Mechanism for setting up each test, and cleaning after it => tests executes in the same, known context
  - Test suite assembling
  - Reporting, with xml and html report generation
  - ant integration
  - GUI
  - Integration in most IDEs (E.g.: Eclipse, IntelliJ, …)

Unit testing – jUnit sample

```
public class IlrCVSTestBase extends TestCase {
    public IlrCVSTestBase(String testName) {
    super(testName);
    . . .
    }
…}
public class IlrRepositoryRelationTestBase extends IlrCVSTestBase {
    private File moduleDirectoryUser1;
    private File moduleDirectoryUser2;
    private IlrRepository repository1 = new IlrBrmRepository();
    private IlrRepository repository2 = new IlrBrmRepository();

    public IlrRepositoryRelationTestBase(String testName, String aPropertyFileName) {
        super(testName, aPropertyFileName);
        assertTrue(getCVSClient().isConnectionPossible(getCVSRoot(0), getPassword(0)));
        assertTrue(getCVSClient().isConnectionPossible(getCVSRoot(1), getPassword(1)));
        getMediator(0).setCVSPassword(getPassword(0));
        getMediator(1).setCVSPassword(getPassword(1));
    }
```

Unit testing – jUnit sample

```
protected void setUp() throws Exception {
    super.setUp();
    moduleDirectoryUser1 = IlrCVSUtil.addFolder(null, getLocalDestinationPath(0));
    assertNotNull("moduleDirectory for user 1 is null", getModuleDirectoryUser1());
    moduleDirectoryUser2 = IlrCVSUtil.addFolder(null, getLocalDestinationPath(1));
    assertNotNull("moduleDirectory for user 2 is null", getModuleDirectoryUser2());
    moduleDirectoryUser1 = checkoutRepository(repository1, getMediator(0));
    assertNotNull("Couldn't check-out repository for user 1", getModuleDirectoryUser1());
    moduleDirectoryUser2 = checkoutRepository(repository2, getMediator(1));
    assertNotNull("Couldn't check-out repository for user 2", getModuleDirectoryUser2());
}
protected void tearDown() throws Exception {
    getRepository1().getPersistenceManager().close();
    getRepository2().getPersistenceManager().close();
    assertTrue(IlrCVSUtil.deleteFile(getModuleDirectoryUser1()));
    assertTrue(IlrCVSUtil.deleteFile(getModuleDirectoryUser2()));
    super.tearDown();
}
```

Unit testing – jUnit sample

```
protected IlrDynamicObjectModel findBom(IlrRepository aRepository) {
    IlrRefPackage refPack = aRepository.getExtent("Application");
    assertNotNull(refPack);
    IlrLibrary lib = (IlrLibrary) refPack.findModelElement("Template Library");
    assertNotNull(lib);
    IlrDynamicObjectModel bom = (IlrDynamicObjectModel)lib.getBOM();
    assertNotNull(bom);
    return bom;
}

protected void changeRelationsABIntoACAndForScenario1(IlrElement elem1A,
    IlrElement elem1B, IlrElement elem1C) throws IlrRepException {
    IlrStructuralFeature typeSF = elem1A.getStructuralFeature(IlrConstants.TYPE_REFERENCE);
    assertNotNull("Can't find type Structural Feature", typeSF);
    Object oldValue = elem1A.getValue(typeSF);
    assertTrue("old value should be elem B", oldValue == elem1B);
    elem1A.setValue(typeSF,elem1C);
    Object newValue = elem1A.getValue(typeSF);
    assertTrue("new value should be elem C", newValue == elem1C);
}
```

Unit testing – jUnit sample

```
public static TestSuite suite(){
    TestSuite suite = new TestSuite("IlrUpdateTestCase");
    suite.addTest(new IlrUpdateTestCase("testUpdateOnModifiedFile"));
    suite.addTest(new
      IlrUpdateTestCase("testUpdateOnUnmodifiedFolder"));
    suite.addTest(new IlrUpdateTestCase("testUpdateOnDeletedFolder"));
    suite.addTest(new
      IlrUpdateTestCase("testUpdateCleanOnDeletedFolder"));
    suite.addTest(new
      IlrUpdateTestCase("testUpdateFileWithMissingRevision"));
    suite.addTest(new IlrUpdateTestCase("testUpdateFileWithRevision"));
    suite.addTest(new
      IlrUpdateTestCase("testUpdateCleanFolderOnModifiedFolder"));
    suite.addTest(new IlrUpdateTestCase("testUpdateOnConflictFile"));
    suite.addTest(new IlrUpdateTestCase("testUpdateReadOnlyFile"));
    return suite;
}
```

Unit testing – jUnit sample

- Example:

```
public class FooTest
   void setUp();
   void tearDown();
   void testFunctionA();
   void testFunctionB();
```

- Lifecycle: what the test runner does:

```
FooTest f = new FooTest();
f.setUp();
f.testFunctionA();
f.tearDown();
f.setUp();
f.testFunctionB();
f.tearDown();
```

Unit testing – jUnit sample

ant integration:

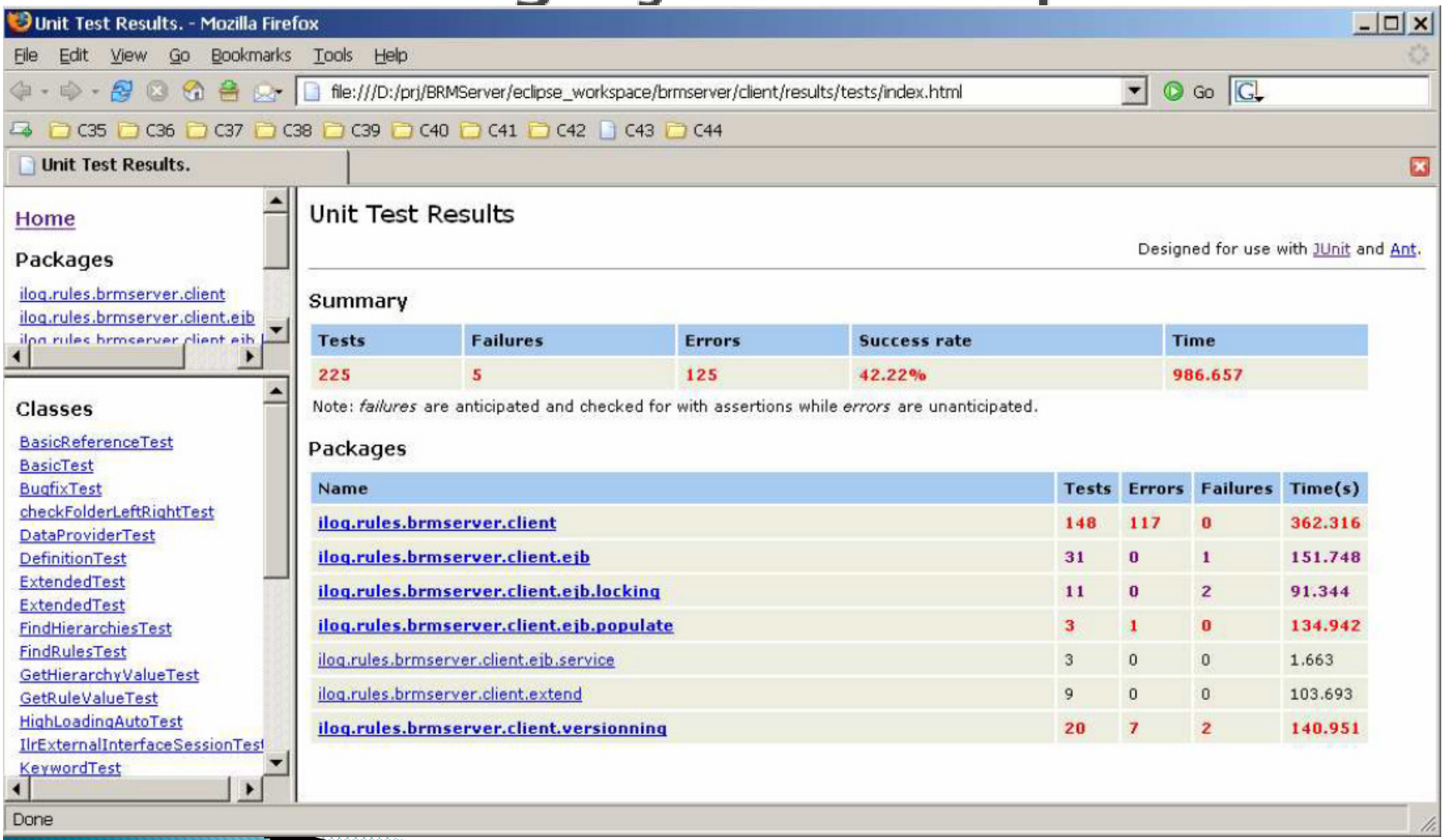
```
<target name="run.junit">
  <property name="junit.includes" value="**/*Tests.class" />
  <junit printsummary="yes" fork="yes" maxmemory="512m"
    haltonfailure="no">
    <classpath>
      <pathelement location="${classes}"/>
      <pathelement location="${scripts.dir}/lib/junit.jar"/>
      <pathelement location="${scripts.dir}/lib/dom4j-1.4-dev-8.jar"/>
      <pathelement location="${scripts.dir}/lib/ant-testutil.jar"/>
      <pathelement location="${integration.dir}/lib/dom.jar"/>
      <pathelement location="${integration.dir}/lib/j2ee-1.3.1.jar"/>
    </classpath>
    <jvmarg value="-Dproperties.file=${basedir}/properties.file"/>
    <batchtest todir="${tests.reports.dir}">
      <fileset dir="${classes}" includes="${junit.includes}“ excludes="${junit.excludes}" />
    </batchtest>
    <formatter type="xml"/>
  </junit>
</target>
```

Unit testing – jUnit sample

Reporting:

```
<target name="report" >
  <junitreport todir="${tests.reports.dir}">
  <fileset dir="${tests.reports.dir}“
  includes="TEST-*.xml" />
      <report todir="${tests.reports.dir}" />
  </junitreport>
</target>
```

Unit testing – jUnit sample



Integration testing

- Purpose: test the system (or part thereof) after integration of several components
- Why?
  - Although each component may work well separately, they may not operate correctly together, due to (among other reasons):
    - Communication issues
    - Synchronization issues
    - Different data ranges / data types
    - Misunderstanding of contracts
    - Bugs introduced during integration

Integration testing

- How to perform this type test?
  - Same as unit-tests, but:
    - on (a subset of) the whole system (that is, theresult of a (partial) integration)
    - Perform scenarios closer to real-life situation
  - Issue is often the GUI, so to work around this:
    - Several people stuck in a room typing all day long following written scenarios
    - Bypass the GUI by plugging the test tool at the layer just below it. GUI will then have to be tested separately

Integration testing

- White / black box ?
  - Black box:
    - Define input and expected output.
    - Input data into system
    - Compare actual output with expected result
    - This can be done without actual knowledge of how the system is built => easy to outsource or delegate to others

Integration testing

- White / black box ?
  - White box:
    - Same, but also look at the internal state of the system along the data path
    - Usually, can only be performed by the writers of the system:
      - Biased tests (they know the happy path)
      - Utilize resources that may be needed elsewhere => tests not done thoroughly
      - Often needed to understand complex scenario (E.g.:debugging)

Integration testing

- Tools:
  - Tests performed by tester teams:
    - Full duplicate of production environment: same database, app servers, etc.
    - Tools to quickly restore system in a “clean” state, E.g.: DB scripts, image drive, etc.
    - Internal Bug Tracking: BugZilla, ad hoc database
    - Reporting: spreadsheet, reporting component of dedicated bug tracking tool.

Integration testing

- Tools:
  - Tests performed by dev teams:
    - Ideally, in test environment as close as possible to production environment. Often, performed in dev environment, especially when testing partial integration.
    - Same type of tools as for unit-testing. Often beefed-up with scenarios.
    - For example, with jUnit, one can build scenarios with test suites, each step being a unit-test.
    - Similar tracking and reporting needs and tools

Integration testing

- Tools:
  - When GUI is involved
    - "Learning robots": record UI interaction in a (proprietary) scripting language, then replay and compare results with expected, at UI level
  - Often, ability to write directly in the dedicated scripting language.
  - Not very robust to change, often require manual intervention

Regression testing

- Purpose: detect regressions introduced between two releases of the system

- Why?
  - Regression DO happen
  - Side-effects
  - Specification changes
  - Bug correction leads to introduction of other bugs

Regression testing

- What tests can be used for regression testing:
  - Unit-tests, integration tests, pretty much anything that can easily be automated
  - The more the better
- How to perform this type test?
  - Run suites of tests against two releases of the software, with the same data set
  - Compare tests results
  - Log regression in bug tracking system
- Shows how important CM is

Usability testing

- Purpose: Find out is the system is really usable by its intended audience

- Why?
  - System is built by developers … but used by Business Users
  - Even minimal UI changes can confuse business users with years of experience of “doing it this way”
  - System has to face real-life usage

Usability testing

- How: Almost impossible to automate

- Tips:
  - Involve ergonomic specialists early in the project
  - Use reusable, standardized UI components
  - Take performance into account: a slow responding system won't be accepted easily
  - Have Business Users test early on UI mockups

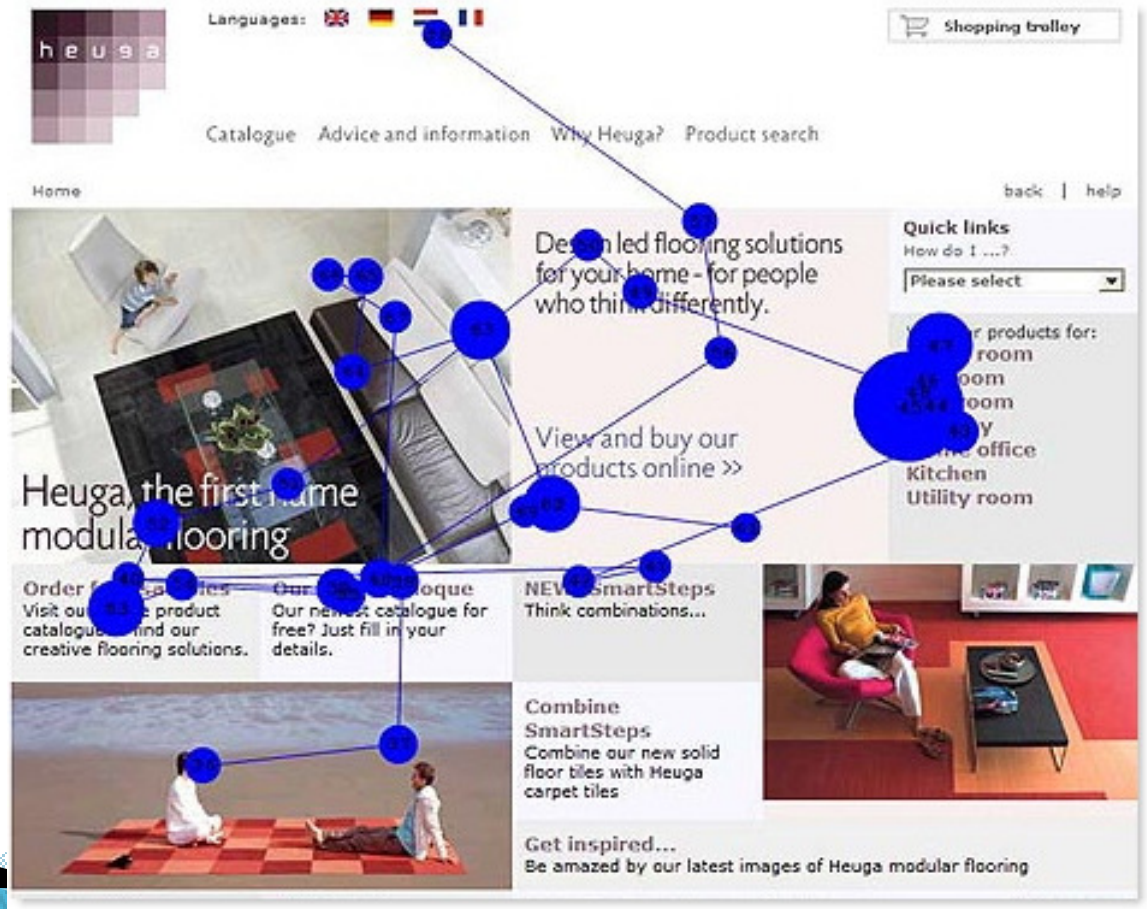
Usability testing

- How: Almost impossible to automate

- Tips:
  - Involve ergonomic specialists early in the project
  - Use reusable, standardized UI components
  - Take performance into account: a slow responding system won't be accepted easily
  - Have Business Users test early on UI mockups

Usability testing

- Tools: Eye tracking



Performance testing

- Purpose: test system performance, both globally (from a user transaction prospective) and locally (each function, each resource)
- Why?
  - User responsiveness (hence, acceptance)
  - Hardware costs
  - Detect resource contention issue that may only reveal in production

Performance testing

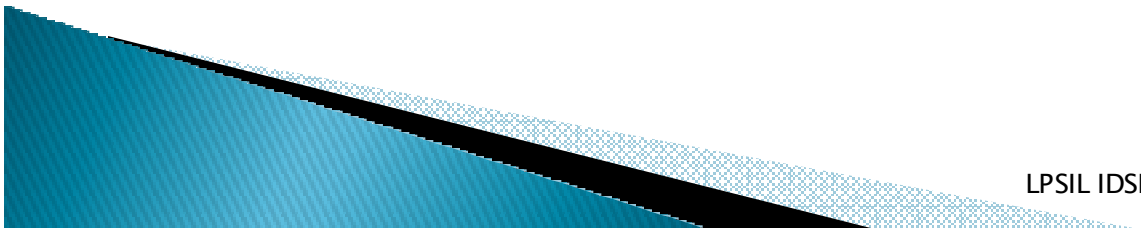
How to perform this type test?

- Globally
  - Perform test scenarios and stopwatch them
    - Manually (user testing and reporting times + subjective feedback)
    - Automated: frameworks such as HttpUnit, WebStressTool, etc.
  - Take into account system operative mode, E.g.: transactional, nightly batches
  - Measure against hardware dimensions and expected / worst case load

Performance testing

How to perform this type test?

- Locally
  - Instrument code at method level, using profilers (YourKit, OptimizeIt, JProbe, Jfluid, etc.)
  - Log: traces should be time stamped.
  - At resource level, E.g.: filter queries to DB, measure throughput against cpu usage, etc.



Scalability vs Load Testing

- Exercice: what is the difference?

Scalability testing

- Purpose: test system performance degradation under load increase

- Why?
  - Ideally, the system performance should be linear with load
  - Hardware costs forecast: if usage double, will hardware costs double as well, or more ?
  - Detect algorithmic issues, poorly coded functions

Scalability testing

How to perform this type test?

- Stress-load the system
  - Test scenarios with simulated heavy loads
  - Make sure the test clients are not the bottlenecks themselves: sufficient hardware, dimension stress test environment
- Plot performance vs. load and establish trend: linear, exponential ?
- Identify resource contention. For example, an app-server cluster with a single, slow database

Reliability vs Recovery

- Exercice: what is the difference?

Coverage testing

- What type of coverage ?
  - Lines of code
  - Platforms
  - Features

Translatability testing

- Purpose: ensure the system can be translated to other languages

- Why?
  - To detect hard-coded pieces of text
  - To check for icons/images with local meaning

Translatability testing

- How:
  - Using a pseudo-locale
  - Mostly manual process

Globalization testing

- Purpose: ensure the system can be operated once translated to another language

- Why?
  - To detect if translations mean something usable
  - To verify that translation didn't cause any regression

Globalization testing

- How:
  - Functional scenarios, manual
  - Requires native speakers

Accessibility testing

- Purpose: ensure the system can be operated by people with disabilities

- What?
  - Color-blind -> high contrast display
  - No-mouse operation
  - Zoomable fonts

Documentation testing

- Purpose: ensure the system is documented, in all supported languages
- What?
  - Documentation can be displayed
  - Doc is complete, in the right languages
  - Doc snapshots match real software
  - …

Code Inspections

- Recurrent Peer reviews

- Look at other developer code and spot:
  - awkward code
  - unnecessarily complex code
  - potential errors
  - sub-optimal algorithms

Security testing

- Purpose: test system security, as well as the (in)ability of the system to give access to other systems
- Why?
  - Weakest link of the chain
  - Hardware costs
  - Detect resource contention issue that may only reveal in production

Security testing

- Why ? video

Agenda

- Test Types
- Tooling and Strategy

Testing: When ?

- After development is done?
- During the development?
- Or even before?

- And/or

- Once product is released
  - Beta version
  - Regression testing on fixpacks
  - Customer scenarios

- Cost: 1x, 10x, 100x, 1000x, 10000x

Testing: When – During dev

- Unit-tests
  - At module level
- Integration tests
  - Works best with continuous integration
- Regression tests
  - All along
- System tests
  - Dedicated phase
- Acceptance tests
  - Before delivery

Testing: When – Once released

- Beta program needs be managed
- Customer cases
  - Show product usage
  - Exhibit scenarios we may not have used for testing

Testing: What ?

- New features
- Things customer are going to see first
  - Installers
  - Tutorials
  - documentation
- Things that have high impact if they break
- Code commits impact analysis

Testing: Who ?

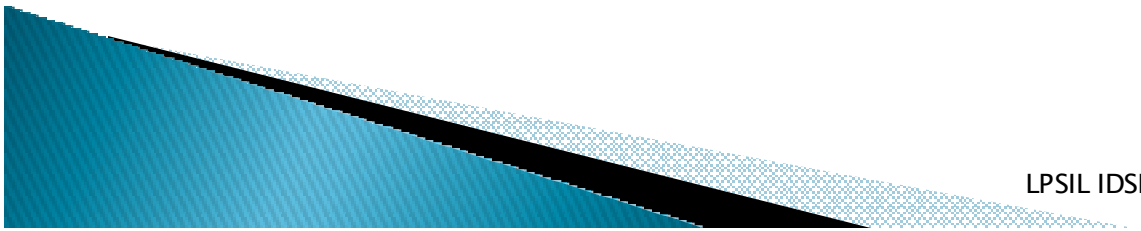
- Testers
- But also
- Developers
- Doc writers
- Product managers
- Customers…

Testing: How?

- Tooling
  - Unit-test: jUnit, Nunit, HttpUnit, Mock Objects, …
  - Integration: jUnit report
  - UI Robots: QFTest, Selenium
  - Coverage: Clover
  - Test plan manamgent: RQM, Mercury
- Frequency
  - Unit-tests: daily if not hourly
  - Integration tests: ideally daily
  - System, usability, etc: at least once per iteration
  - The more the merrier

Testing: How?

- Coverage
  - Line coverage
  - Platforms
    - OS, DB, browser, JVM version, etc.
  - Data ranges
    - Test case generation
- Combinatorial madness
  - Need smart choices
  - Need to document what was tested



Next session

- Defect Management (Oct 4th)