Testing of Scout Application
Ludwigsburg, 27.10.2014
The Tools approach...
The Testing Theory approach...

- Unit testing
- White box testing
- Black box testing
- Integration testing
- Functional testing
- System testing
- End-to-end testing
«What is your goal?»
Application under test
The application under test
Requirements (1)

Modification of the input fields...

Specific format: <name> [<parts>] value: <value>
Requirements (2)

Only the available parts are listed in the field.

Is only a part available, the field is disabled.
Requirements (3)

Export button:
- validate the form
- register the minifig in the server
- reset the form
A Scout application

Client Application

Client

ui

shared

Scout Client

Java / Eclipse

Server Application

Server

shared

Scout Server

Java / Eclipse

Webserver

Service Tunnel
A Scout application

Client Application

- Scout Client
- Java / Eclipse

Server Application

- Scout Server
- Java / Eclipse
- Webserver

Service Tunnel

ui

client

shared

server

shared
A Scout application

Client Application

- Client (ui)
- Client (shared)

Scout Client
Java / Eclipse

Server Application

- Server (shared)

Scout Server
Java / Eclipse
Webserver

Service Tunnel
Unit testing
Test for logic in the shared plugin
Test with the Scout services
@RunWith annotation

Annotate the test class with the Annotation:

```java
@RunWith(ScoutClientTestRunner.class)
public class DesktopFormTest {

    // ...

}
```

It adds:

- Equinox OSGi Runtime
- Scout Context, Services, ...
Mock remote Services

Create the mock

```java
private IDesktopProcessService m_mockService = Mockito.mock(IDesktopProcessService.class);
```

Define the behavior for your tests:

```java
Mockito
    .when(m_mockService.load(
        Mockito.any(DesktopFormData.class)))
    .thenReturn(someFormData);
```
TestingUtility.registerServices(..)

» Dynamically register your mocked service:

@Before
public void setUp() {
    m_registeredServices = TestingUtility.registerServices(
        Activator.getDefault().getBundle(), 1000, m_mockService);
}

@After
public void tearDown() {
    TestingUtility.unregisterServices(m_registeredServices);
}
Demo
Integration tests
Integration tests
Integration tests: Example setup

➡ Deploy the server:
  - In a managed environment (database, external services...)
  - As near as possible from the productive environment

➡ Start an head-less client:
  - Browse through the data (outline, pages)
  - Open some forms

➡ Depending on how-much effort you want to put in the client, it is possible to write one generic test for all pages and forms
Automated user tests
Test procedures
Test procedures
Test with Jubula
Jubula

- As a user would work – passing through all layers
- Test creation, execution, analysis

- Drag and drop test creation:
  - No recording
  - No programming
  - Very similar to development code

- Constant feedback about quality
  - Acceptance testing
  - Regression testing
Workflow
Using the specification to automate tests
Testing an application with Jubula

1. Write test
2. Start AUT
3. Perform object mapping
4. Run test
5. Analyse
Assign ids to the scout fields

New with Luna

- DSKTFORM_NAME_TXTFLD
- DSKTFORM_LEGS_SMRFLD
- DSKTFORM_EXPORT_BUTFLD
Scout UI Tests
Unit tests with UI
public class DesktopFormUiTest extends AbstractTestWithGuiScript {

    @Override
    protected void runModel() throws Throwable {
    }

    @Override
    protected void runGui(IGuiMock gui) throws Throwable {
    }
}
I GuiMock

» Abstraction for the UI layer

» Definition of UI interaction:
  - `gui.pressKey(Key)`
  - `gui.typeText(FieldType, int)`
  - `gui.gotoField(type, index)`
  - ...

» Interface with implementations:
  - For Swing
  - For Swt
«How are you testing your Scout application?»
Summary

- As with any other application, writing automated tests for your eclipse scout application is possible.

- Everything is possible
  - Unit tests
  - UI tests
  - Integration tests
  - Performance tests

- There is a cost, so:
  test only what makes sense for your application.
Thank You