

UI Testing Patterns and Best Practices

EclipseCon 2009

Phil Quitslund and Dan Rubel
Instantiations, Inc.

UI Test Patterns and Rules of Thumb

Conditions When timing is uncertain, prefer waiting for conditions over simple pauses in execution.

Condition Monitors Consider implementing a condition monitor if the application under test generates unpredictable interactive events.

Conditions as Asserts Use conditions as a means to assert facts about the application under test.

Conditions as Invariants Use conditions as a means to assert invariants.

Stand-Alone Tests Minimize interaction between tests by insisting tests build up (and tear down) as much application context as possible.

Helpers Push common test functionality into reusable test helpers.

Self-Verifying Helpers Simplify test code by pushing pre- and post-condition validation into helper methods.

Independent Helpers Make helpers nimble by ensuring that they do their own context setup and teardown.

Well-Behaved Helpers Helpers should have as few collateral side-effects as possible (and these side effects should be well-documented).

Strict Layering Build a test helper façade for the application under test and insist that tests only access the application through the façade.

Test Language Treat test helpers as primitives in a domain-specific test language.

Helper as deliverable Ensure helpers are well-factored by treating them as first-class deliverables.

Lock-step delivery Deliver helpers at the same time as associated application functionality.

Code Snippets

Listing 1: Project Creation First Cut

```
@Test
public void projectCreation() {
    click(menu("File/New/Other..."));
    //wait for wizard to show
    pause(3000);
    click(tree("General/Project"));
    click(button("Next"));
    enter("MyProject");
    click(button("Finish"));
    //wait for wizard to be disposed
    pause(3000);
    //TODO: verify project exists
}
```

Listing 2: Basic pause implementation

```
public static void pause(int ms) {
    try {
        Thread.sleep(ms);
    } catch (InterruptedException e) {
        //ignore
    }
}
```

Listing 3: Project Creation with a wait

```
@Test
public void projectCreation() {
    click(menu("File/New/Other..."));
    waitFor(DialogShowingCondition.titled("New"));
    //...
}
```

Listing 4: Basic conditional wait implementation

```
void waitFor(ICondition condition) {
    long start = currentTime();
    do {
        if (condition.test())
            return;
        pause(interval());
    } while (!timeout(start));
    throw new TimeoutException();
}
```

Listing 5: A Condition interface

```
public interface ICondition {
    boolean test();
}
```

Listing 6: DialogShowingCondition implementation sketch

```
final String title;

public void test() {
    Shell shell = getActiveShell();
    return matches(title, shell.getText());
}
```

Listing 7: A (bad) example of in-lined conditional tests

```
@Test
public void dontDoThis() {
    testForUnexpectedError();
    doSomething();
    testForUnexpectedError();
    doSomethingElse();
    ...
}

void testForUnexpectedError() {
    if (errorDialogIsOpen())
        dismissErrorDialog();
}
```

Listing 8: Eclipse JFace baked-in handling of the unexpected

```
class ErrorDialog {
    public static boolean AUTOMATED_MODE = false;
    ...
}

class WorkbenchTestable {
    public void testingStarting() {
        ...
        ErrorDialog.AUTOMATED_MODE = true;
        ...
    }
}
```

Listing 9: The ConditionMonitor interface

```
interface IConditionMonitor {
    void add(ICondition condition, IHandler handler);
    void process();
}
```

Listing 10: The IHandler interface

```
interface IHandler {
    handle();
}
```

Listing 11: ICondition context variations

```
interface ICondition {
    boolean test(Display context);
}

interface IHandler {
    void handle(Display context);
}

interface ICondition {
    //evaluated on the display thread
    boolean uiTest(Display context);
}
```

Listing 12: Project creation verification using a condition

```
@Test
public void projectCreation() {
    ...
    assertThat(ProjectExists.named("MyProject"));
}
```

Listing 13: Variations of assertThat

```
static void assertThat(ICondition c) {
    waitFor(c);
}

static void assertThat(ICondition c) {
    waitFor(c, ASSERT_TIME_OUT);
}
```

Listing 14: The createProject helper

```
@Helper
public void createProject(String name) {
    click(menu("File/New/Other..."));
    waitFor(DialogShowing.titled("New"));
    click(tree("General/Project"));
    click(button("Next"));
    enter(name);
    click(button("Finish"));
    waitFor(DialogDisposed.titled("Project"));
    assertThat(ProjectExists.named("MyProject"));
}
```

Listing 15: Tests with temporal coupling

```
@Test
public void projectCreation() {
    dismissWelcomePage();
    createProject(TEST_PROJECT);
}

@Test
public void projectExport() {
    exportProject(TEST_PROJECT);
}
```

Listing 16: A suite with temporal coupling

```
@RunWith(UITestSuite.class)
@Suite.SuiteClasses({
    ProjectCreate.class,
    ProjectExport.class
})
class ProjectTests { }
```

Listing 17: Uncoupled project tests

```
@Before
public void setup() {
    dismissWelcomePage();
    createProject(TEST_PROJECT);
}

void dismissWelcomePage() {
    if (welcomeIsShowing())
        dismissWelcome();
}

@After
public void projectCreation() {
    deleteProject(TEST_PROJECT);
}

@Test
public void projectCreation() {
    /* verified in setup */
}

@Test
public void projectExport() {
    exportProject(TEST_PROJECT);
}
```

Listing 18: A (mostly) independent helper

```
@Helper
void updateProject(String name) {
    ensurePackageExplorerIsVisible();
    contextClick(packageExplorerItem(name), "Update");
    waitFor(UptoDate.project(name));
}
```

Listing 19: Eclipse WizardNewProjectCreationPage

```
private final
void createProjectNameGroup(Composite parent) {
    ...
    projectNameField. /* package-private field*/
        addListener(SWT.Modify, nameModifyListener);
}

private Listener nameModifyListener = new Listener() {
    public void handleEvent(Event e) {
        setLocationForSelection();
        boolean valid = validatePage();
        setPageComplete(valid);
    }
};
```

Listing 20: WizardNewProjectCreationPage refactored to use a presenter

```
private final
void createProjectNameGroup(Composite parent) {
    ...
    projectNameField.
        addListener(SWT.Modify, new Listener() {
            public void handleEvent(Event e) {
                presenter.nameChanged(e);
            }
        });
}

private final ProjectPagePresenter;
WizardNewPageCreationPage(...) {
    ...
    presenter = new ProjectPagePresenter(this);
}
```

Listing 21: ProjectPagePresenter

```
private final IProjectPageView view;

ProjectPagePresenter(IProjectPageView view) {
    this.view = view;
}

void nameChanged(Event e){
    boolean valid = validatePage();
    view.setPageComplete(valid);
}
...
```

Listing 22: ProjectValidationTest

```
private final IProjectPageView viewStub =
    new ProjectPageViewStub();
private final ProjectPagePresenter presenter =
    new ProjectPagePresenter(viewStub);

@Test
public void finishState() {
    presenter.nameChanged(VALID_NAME);
    assertTrue(viewStub.pageComplete());
}

@Test
public void projectNameValidation() { ... }
```