Executable Specifications for Xtext

Moritz Eysholdt

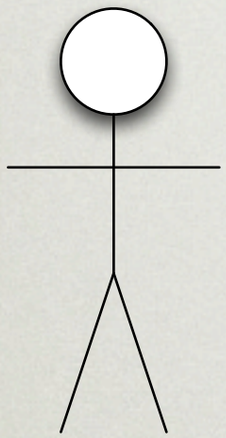
You're Developing a Language

Three Roles

- Language Designer (You)
- The Domain Expert
- Language Users

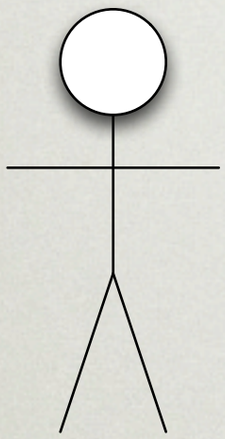
- yes, I simplify

Three Roles (1/3)



- You know something about Xtext
- You have some idea about Language Design

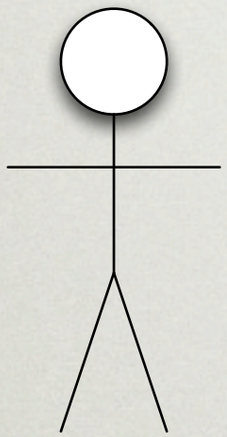
Three Roles (2/3)



Domain Expert

- Deep conceptual knowledge about the domain
- Defines requirements for language
- Has design ideas/expectations?
- Needs to be satisfied with the language

Three Roles (3/3)



Language Users

- Need to understand language
- Will use language regularly

Challenge

The single most important task during the project:

communicate how to use the language

Examples to the Rescue

- create example documents
- and talk about them

Three Important Use Cases

- Specify the Language
- Develop IDE (smart editor, code gen, ...)
- Pass Acceptance Test
- Teach Language

Use Case #1

- Specify the language
  - Discuss examples with customer
    - You guide and prototype
    - Customer knows use cases
    - You or both design
- Result: **Specification by Examples**

Use Case #2

- Develop IDE: Needs automated tests
  - **Use Examples as Test Data**
    - Extend examples with corner cases
    - Enhance examples with test expectations
- not having automated tests...
  - ...is like not using a seat belt in a roller coaster
  - ...will degenerate your code, ‘cause you fear change

Use Case #3

- Acceptance Testing: Customer wants to know...
- ...if and how a uses case is implemented
  - **Prove by Example**
- ...if the IDE (editor, code generator) works
  - **Prove by Passed Test Case**

Use Case #4

- Teach The Language
  - **Explain language by Examples**

- Learning a Language is a Challenge. Help the Users.
- Users won’t accept a language if they don’t understand it
- More Users -> better Return Of Investment

Demo

Xpect

How does this work?

- Tests are JUnit tests (with custom runner)
- Tests can be configured with setups
- Tests can have parameters

JUnit

```
// XPECT myXpectTest --> expected
```

```
import org.junit.runner.RunWith;
import org.xpect.runner.XpectRunner;
import org.xpect.runner.Xpect;

@RunWith(XpectRunner.class)
public class MyTest {

    @Xpect
    public void myXpectTest(IStringExpectation expectation) {
        expectation.assertEquals("actual");
    }

    @Test
    public void myJUnitTest() {
        Assert.assertEquals("expected", "actual");
    }
}
```

Test Setup & Config

```
/* XPECT_SETUP MyTest
    ResourceSet {
        ThisFile {}
        File "test2.dmodel" {}
    }
END_SETUP */
```

```
import org.xpect.xtext.lib.setup.XtextStandaloneSetup;
import org.xpect.xtext.lib.setup.XtextWorkspaceSetup;

@RunWith(XpectRunner.class)
@XpectSetup(XtextStandaloneSetup.class)
public class MyTest {

}
```

Test Parameters

```
// XPECT evaludated --> 31
3 + 4 * 7;
```

```
@RunWith(XpectRunner.class)
@XpectSetup(XtextStandaloneSetup.class)
public class InterpreterTest {

    @Xpect
    public void evaludated(
            @StringExpectation IStringExpectation expectation,
            @ThisOffset Expression expression)
    {
        BigDecimal actual = new Calculator().evaluate(expression);
        expectation.assertEquals(actual);
    }
}
```

Annotations define Parameter Source

- From Setup

```
@ThisModel // access to the root EObject
@ThisResource // access to the EMF Resource
@ThisOffset // access to the offset or current EObject
```

- From XPECT myTest <parsedParameter> --> ...

```
@ParameterParser(syntax="arg0=INT ‘x’ arg1=STRING+ arg2=OFFSET")
```

- Expectations

```
@StringExpectation // string
@LinesExpectation // list of strings, one item per line
@CommaSeparatedValuesExpectation // list, comma separated items
```

Comparison

	Xpect	Plain Junit
Specify Language	yes	no
Integration Test	yes	yes
Acceptance Test	yes	no
Teach Language	yes	no
IDE support for Test Data	yes	no
Simple Evolution of Test Data	yes	no
Unit Test (Without Parsing Test Data)	no	yes

Thank You

http://www.xpect-tests.org

fork me at github :)