EUROPEAN SMALLTALK USER GROUP

IWST 2024

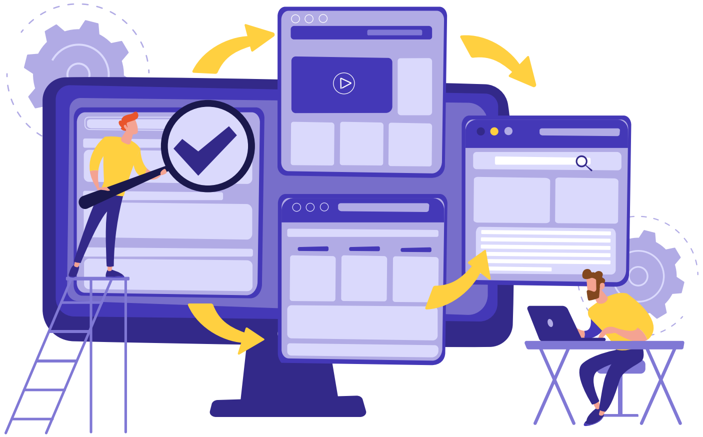
Modest-Pharo: Unit Test Generation Based on Traces and Metamodels

Gabriel Darbord[1]
Fabio Vandewaeter[1]
Anne Etien[1]
Nicolas Anquetil[1]
Benoit Verhaeghe[2]

[1]Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
[2]Berger-Levrault, France

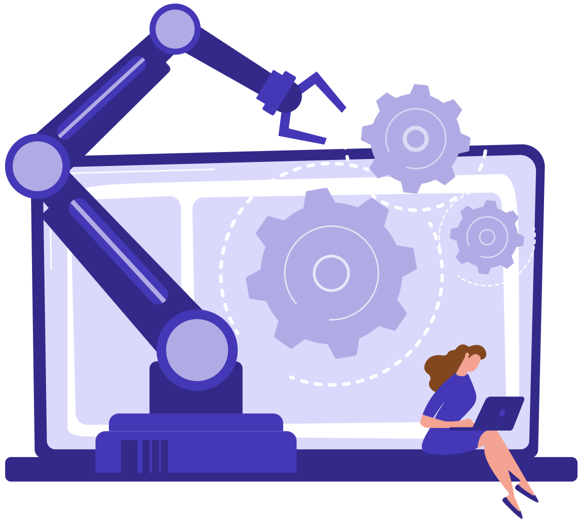
The Importance of Testing

- Nowadays, when developping new software systems:
  - 20-50% time spent on testing
- We test because we want:
  - Bug detection and prevention
  - Quality assurance
  - User satisfaction
  - Non-regression
  - Confidence
  - Etc.



Our Test Generation Approach

- Using software models and execution traces
  - static and dynamic analysis
- Our objective is to generate tests that are:
  - Relevant
  - Readable
  - Maintainable
  - Not requiring existing tests
  - Not contaminating



Test Oracles

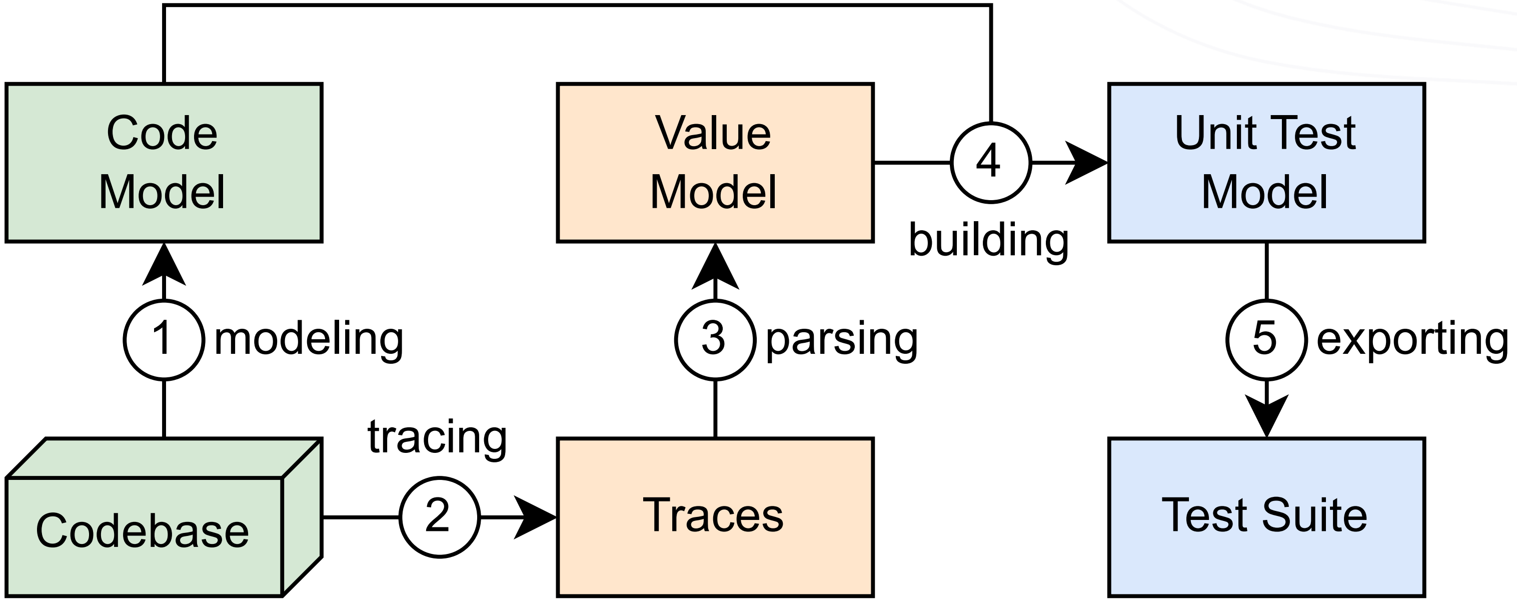
- How can we verify that a program returns the correct answer?
- Mechanism that determines whether a test has passed or failed
- Oracles hold the “truth”
- In our case:
  - Consider legacy to be correct
  - Capture behavior using traces
  - Verify updated behavior matches traces

  →Non-Regression Testing

Example of a Generated Test

```
"Arrange"
aString := 'CK123J'.
lbCTokenizer := LbCTokenizer new.
expected := OrderedCollection withAll: { 'C'. 'K123'. 'J' }.
"Act"
actual := lbCTokenizer tokenize: aString.
"Assert"
self assert: actual deepEquals: expected
```

Approach Steps



Modeling with the Moose Platform

- Moose is a platform for software analysis
- It allows to:
  - Represent a software system in a model
  - Query, manipulate, transform, and visualize models

Tracing by Instrumentation

- MetaLink, MethodProxies…
  - Before and after method
- Requisite payload:
  - Identity of target method
  - Serialized arguments
  - Serialized return value
  - Serialized receiver

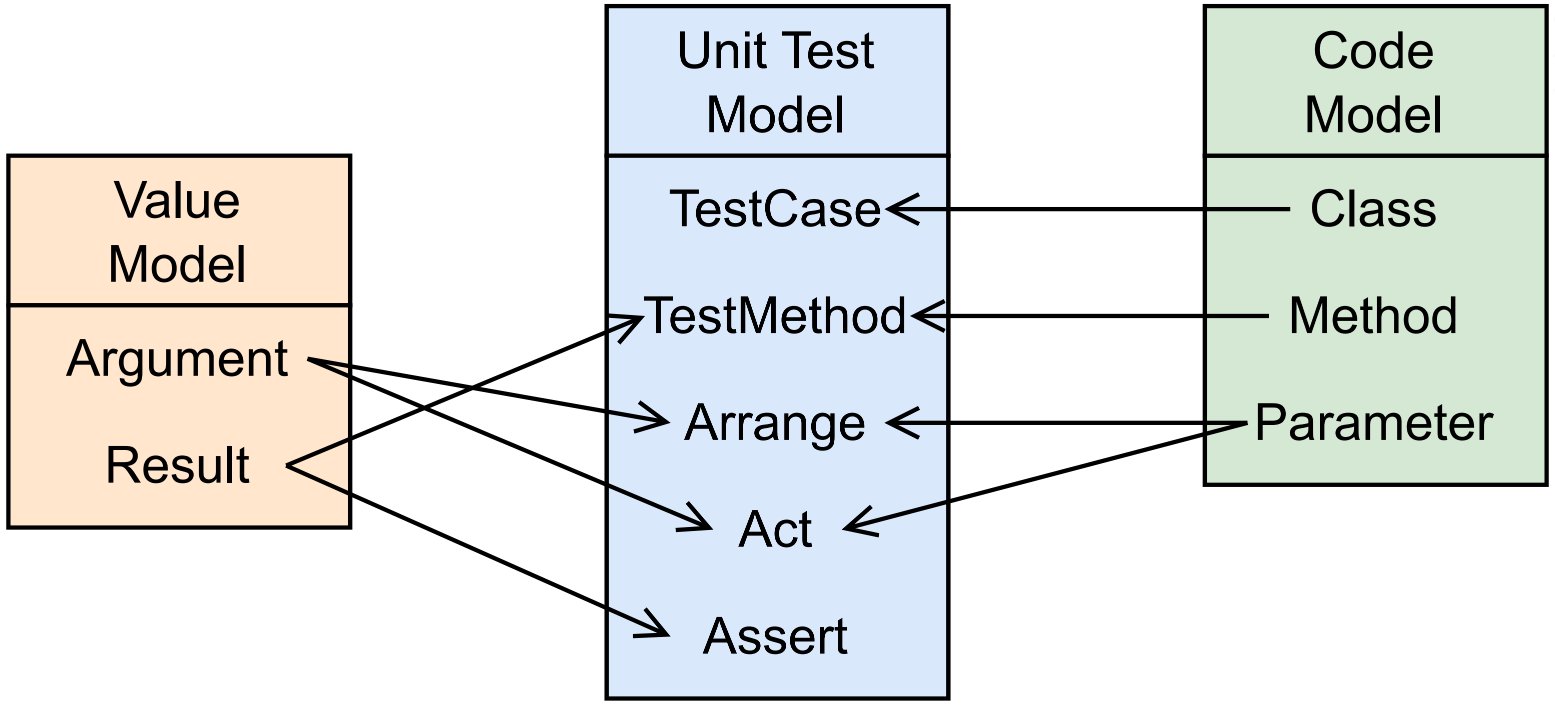
Parsing Trace Data

- JSON based on **Jackson**
  - Type metadata
    →Dynamic typing
  - Object identity metadata
    →Circular dependencies
- Easy to
  - write
  - parse

```json
{
  "@type": "User",
  "@id": 1,
  "name": "John Doe",
  "session": {
    "@type": "Session",
    "@id": 2,
    "active": true,
    "user": { "@ref": 1 }
  }
}
```

github.com/Modest-Project/PharoJackson

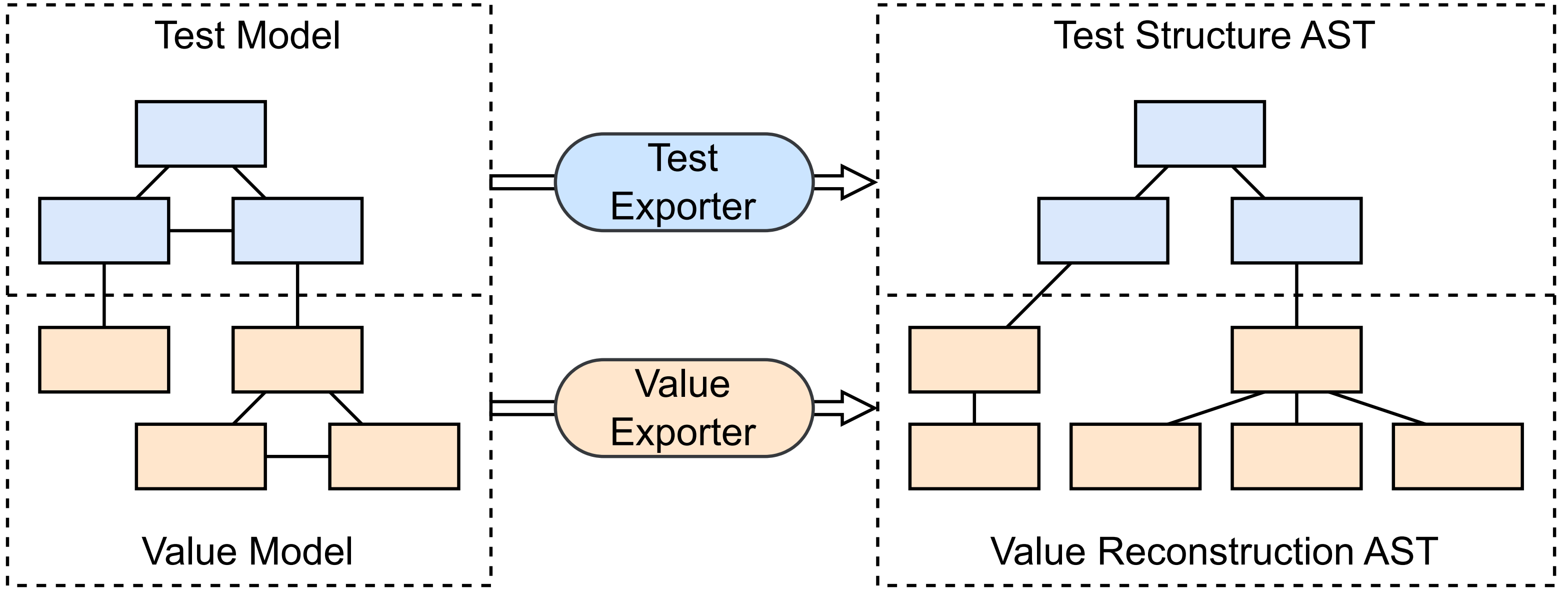
Building Test Model



1. Modeling
2. Tracing
3. Parsing
4. Building
5. **Exporting**

Exporting Test Model

- Use reflectivity to create packages, classes and methods
- Write the code using Pharo's AST



Reconstructing Values

```json
{
  "@type": "User",
  "@id": 1,
  "name": "John Doe",
  "session": {
    "@type": "Session",
    "@id": 2,
    "active": true,
    "user": { "@ref": 1 }
  }
}
```

```smalltalk
(user := User new)
  name: 'John Doe';
  session: (Session new
    active: true;
    user: user;
    yourself);
  yourself.
```

Example of a Generated Test (bis)

Existing test

```
self
 assert: (tokenizer tokenize: 'CK123J')
 equals: #( 'C' 'K123' 'J' ) asOrderedCollection
```

Generated test

```
"Arrange"
aString := 'CK123J'.
lbCTokenizer := LbCTokenizer new.
expected := OrderedCollection withAll: { 'C'. 'K123'. 'J' }.
"Act"
actual := lbCTokenizer tokenize: aString.
"Assert"
self assert: actual deepEquals: expected
```

Results

About target projects

Project	Tested Classes	Methods	Existing Tests	Executable Comments	Covered Methods	Mutation Coverage
DataFrame	1	187	275	0	144	59%
LabelContractor	10	64	31	18	44	56%

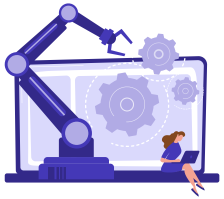
About generated tests

Project	Generated Tests	Passes	Fails	Mutation Coverage	Combined Mutation Coverage
DataFrame	144	114	30	43%	64%
LabelContractor	44	42	2	43%	59%

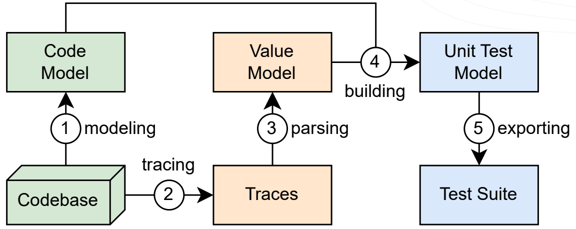
Conclusion

Our Test Generation Approach

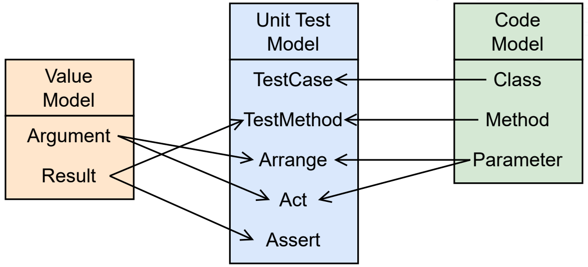
- Using software models and execution traces
  - static and dynamic analysis
- Our objective is to generate tests that are:
  - Relevant
  - Readable
  - Maintainable
  - Not requiring existing tests
  - Not contaminating



Approach Steps



Building Test Model



Example of a Generated Test (bis)

Existing test

```
self
  assert: (tokenizer tokenize: 'CK123J')
  equals: #( 'C' 'K123' 'J' ) asOrderedCollection
```

Generated test

```
"Arrange"
aString := 'CK123J'.
lbcTokenizer := LbcTokenizer new.
expected := OrderedCollection withAll: { 'C'. 'K123'. 'J' }.
"Act"
actual := lbcTokenizer tokenize: aString.
"Assert"
self assert: actual deepEquals: expected
```