IWST 2023

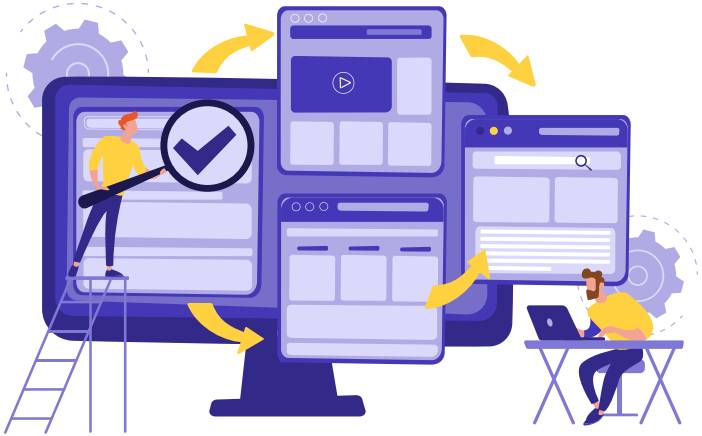
A Unit Test Metamodel for Test Generation

Gabriel Darbord[1]
Anne Etien[1]
Nicolas Anquetil[1]
Benoit Verhaeghe[2]
Mustapha Derras[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.



Legacy Software System Lack Tests

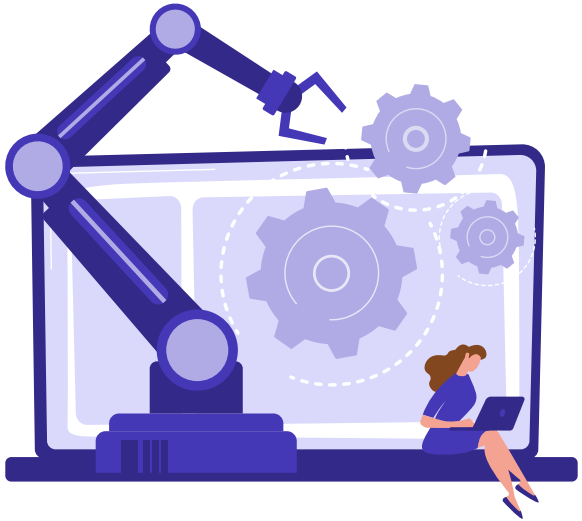
- In 2022, Berger-Levrault owns **150** software programs
  - Three-tier architectures (client, server, database)
  - Millions of lines of code
  - Different legacy technologies that can be up to 25 years old
- Severe lack of tests
  - Developers fear changes

Towards Automated Test Generation

- Berger-Levrault wants tests for their legacy software
- Impossible to write them manually
  - Time consuming
  - Difficult and error-prone
  - Lack of resources

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 relying on existing tests



Different Criteria

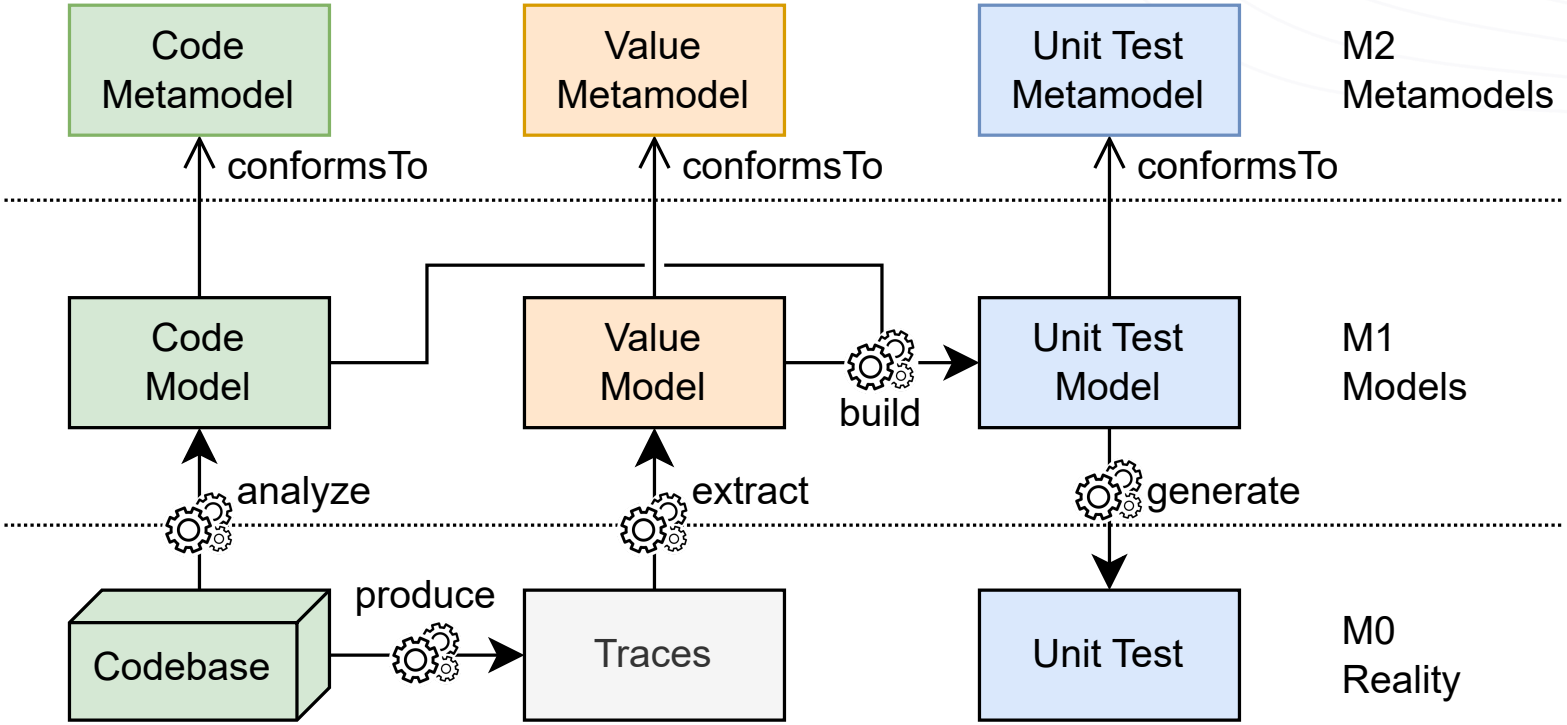
Paper	Relevant	Readable	Maintainable
J. Pires et al.	~	~	+
G. Fraser et al.	~	-	~
A. C. R. Paiva et al.	+	~	~
M. Tufano et al.	~	+	~

LLM approaches need to be trained on codebase and existing tests

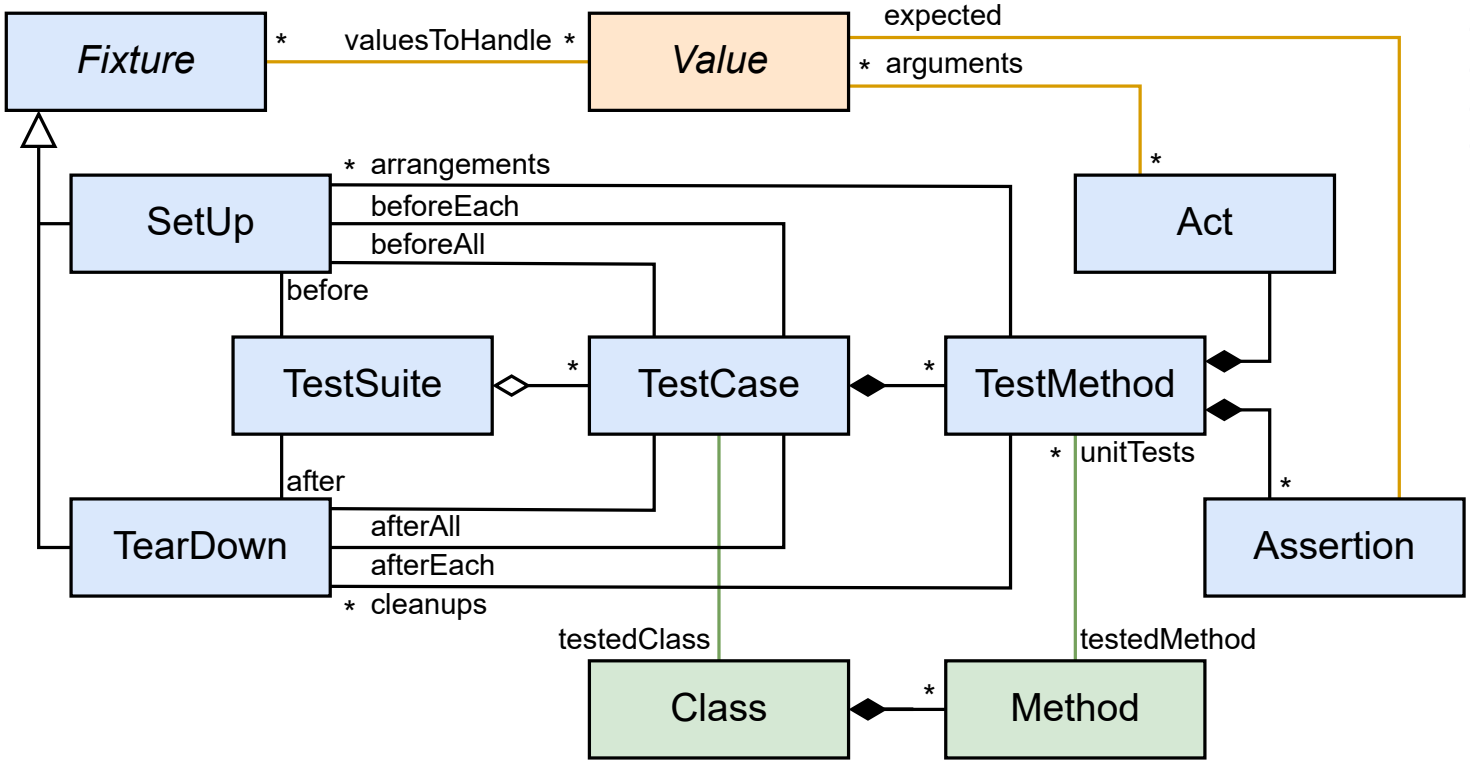
About the Moose Platform

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

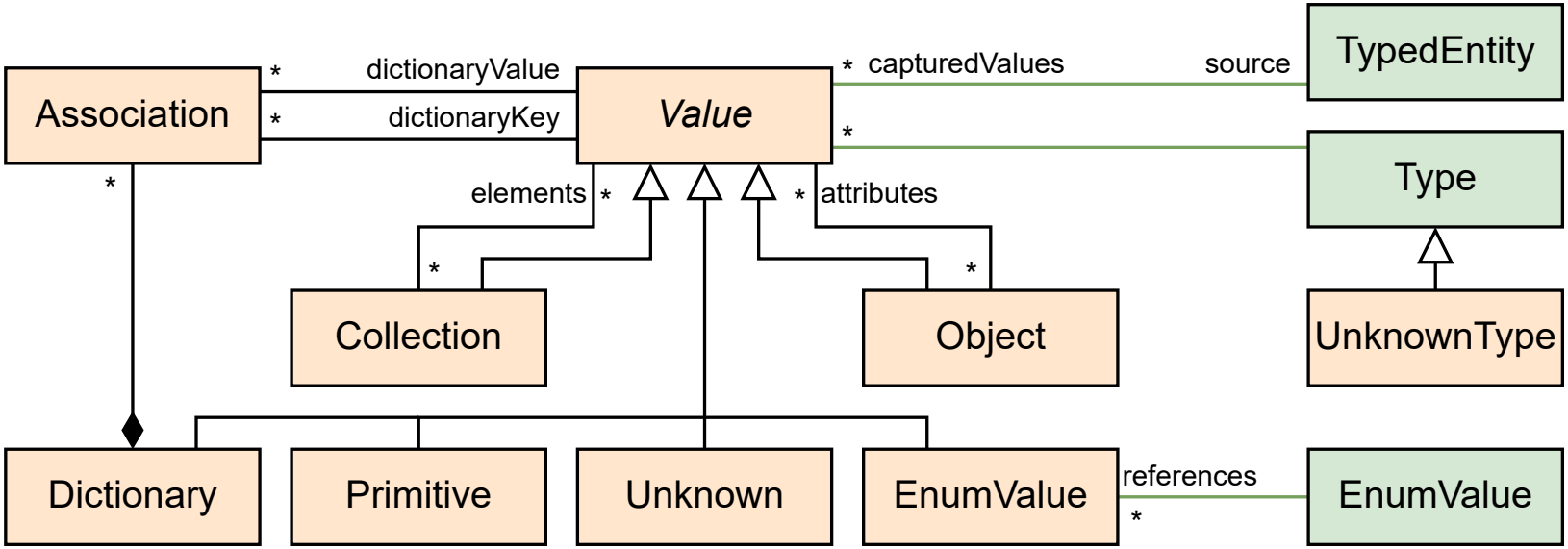
An Approach Based on Metamodels



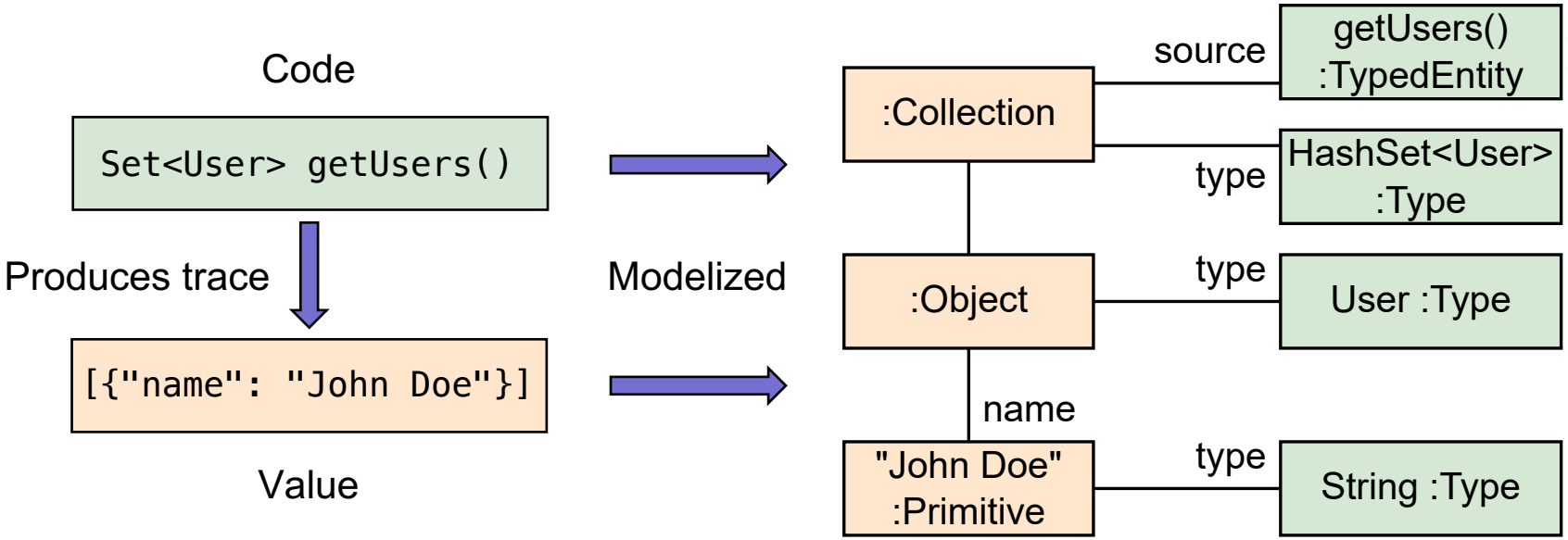
Unit Test Metamodel



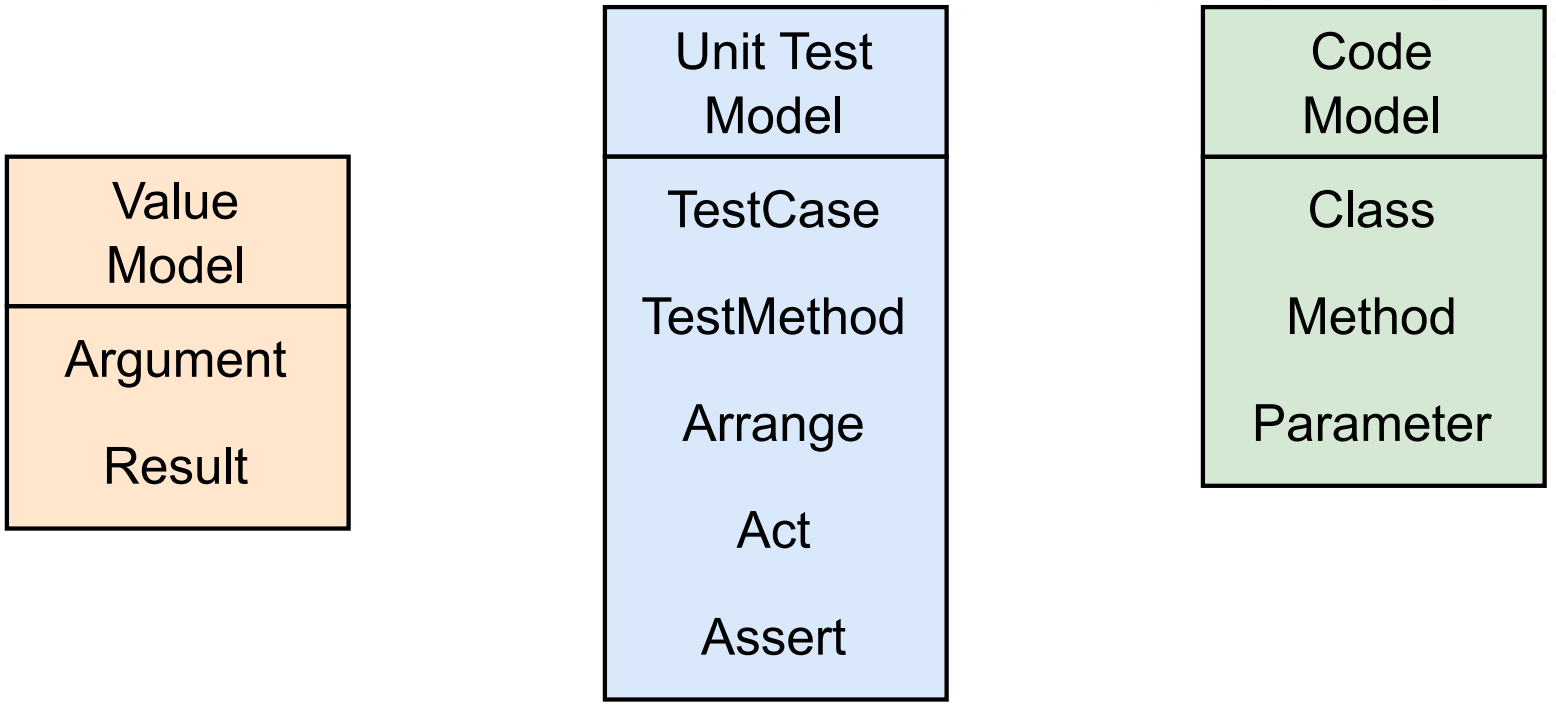
Value Metamodel



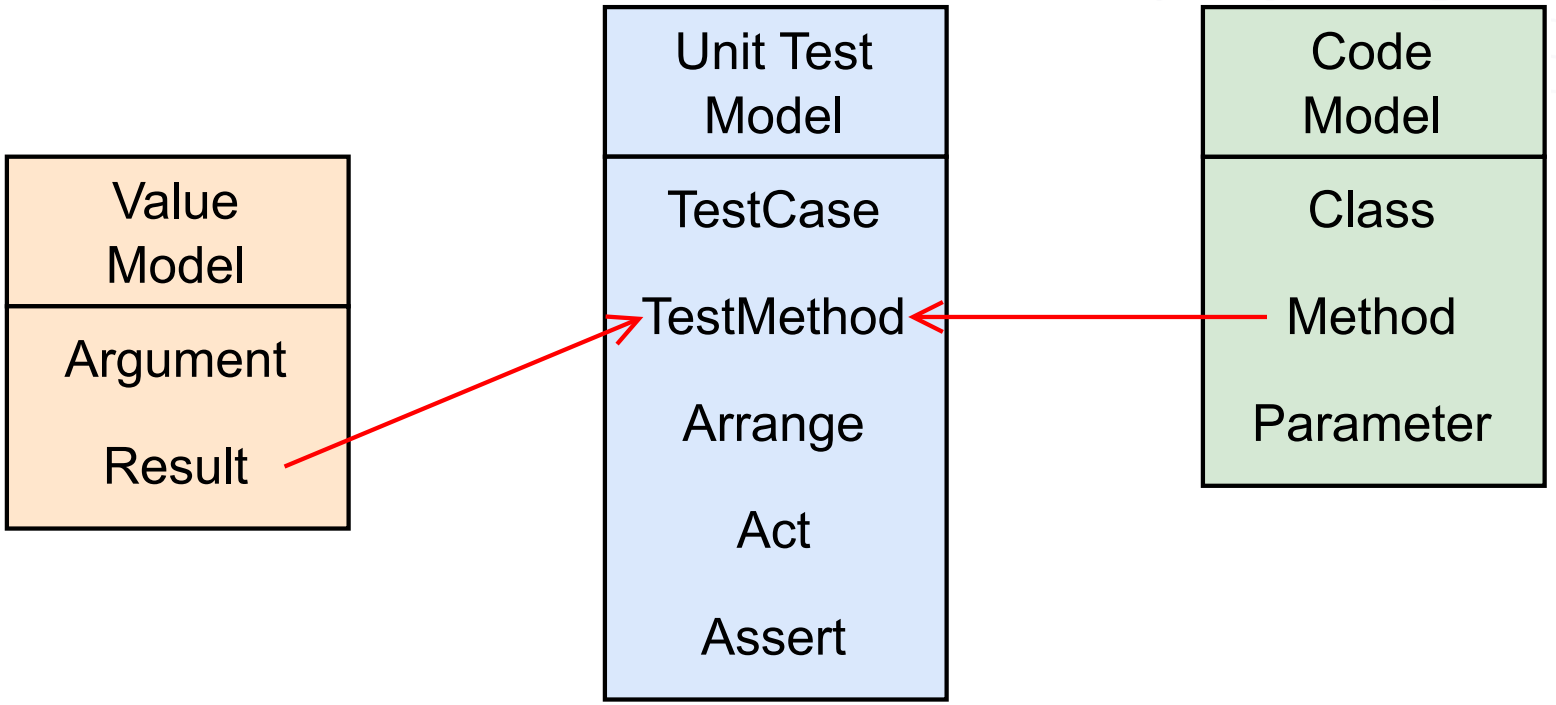
Example Value Model



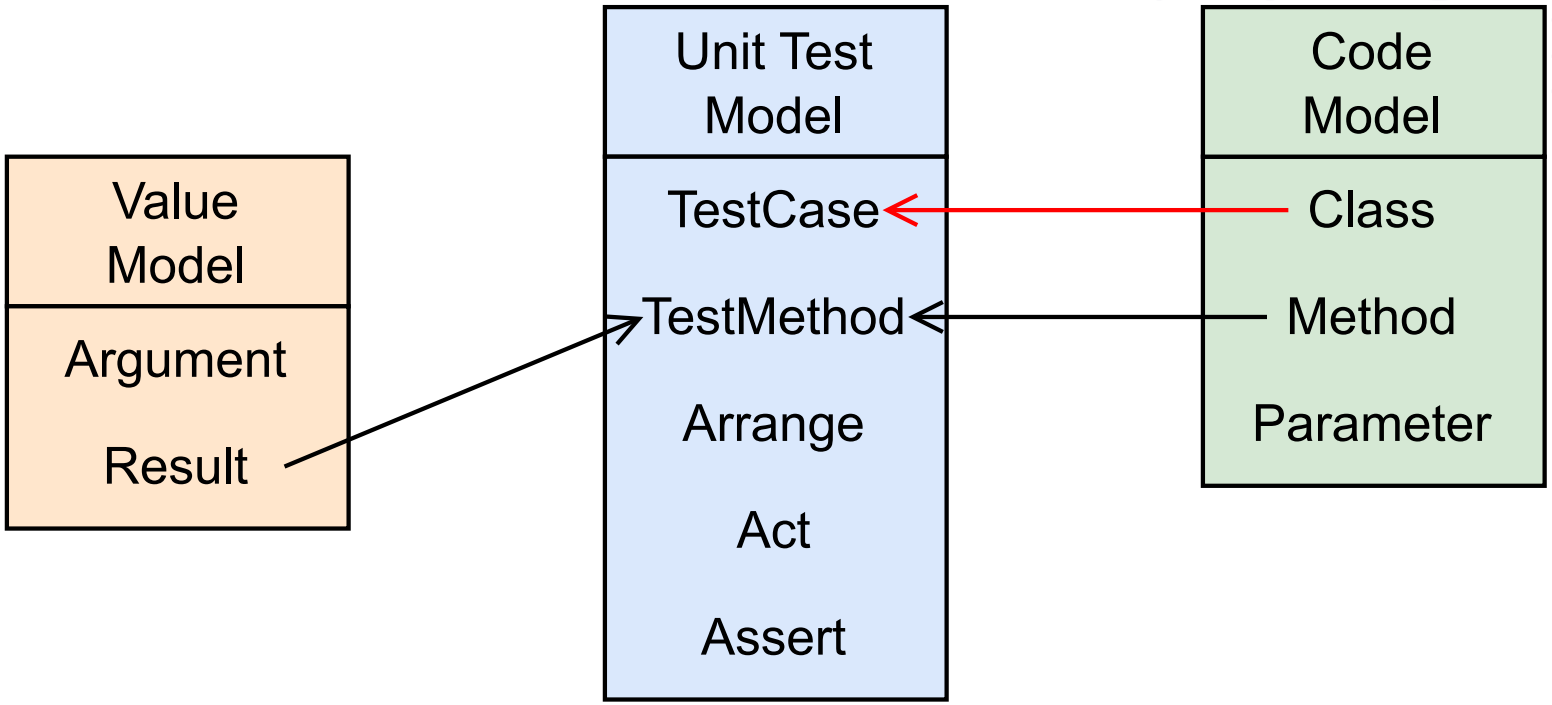
Build Unit Test Using Model Transformations



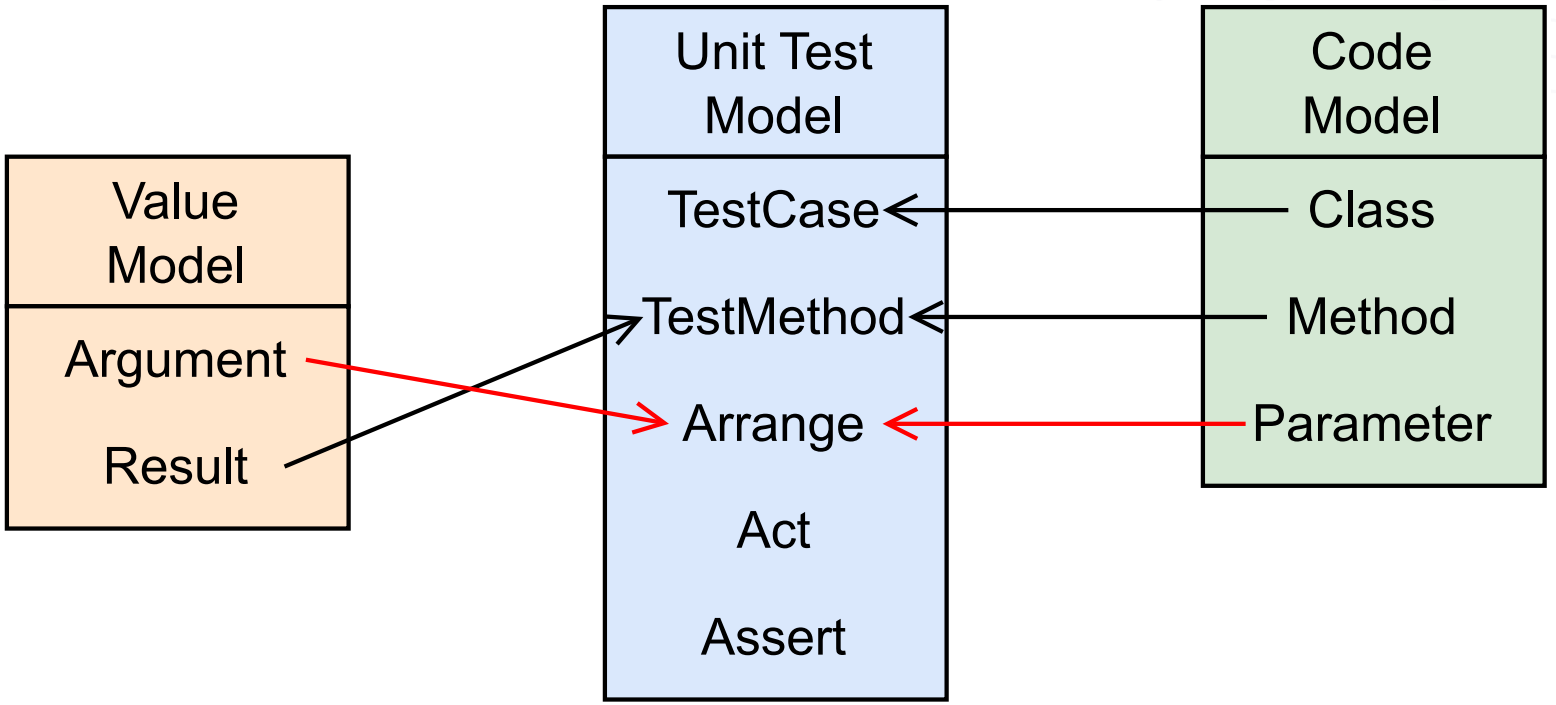
Build Unit Test Using Model Transformations



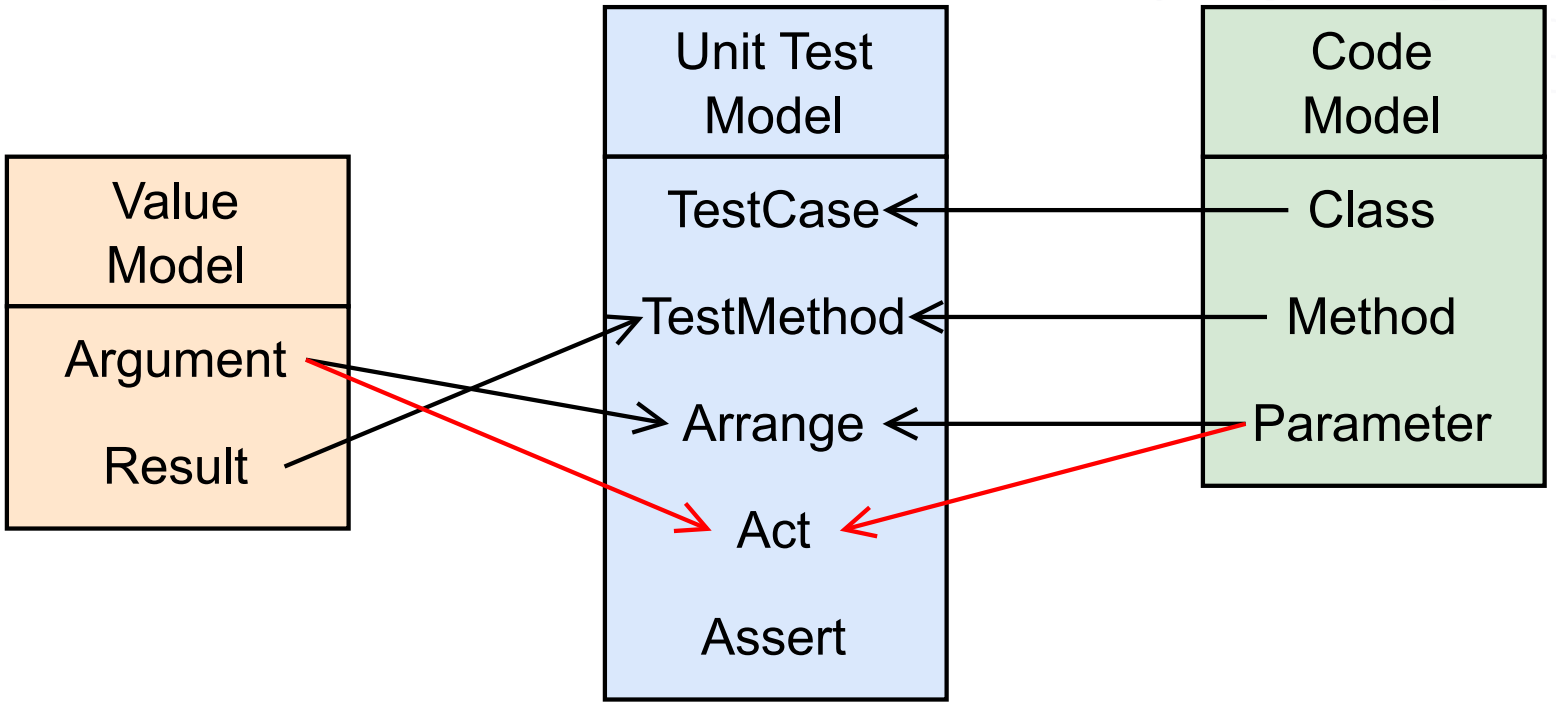
Build Unit Test Using Model Transformations



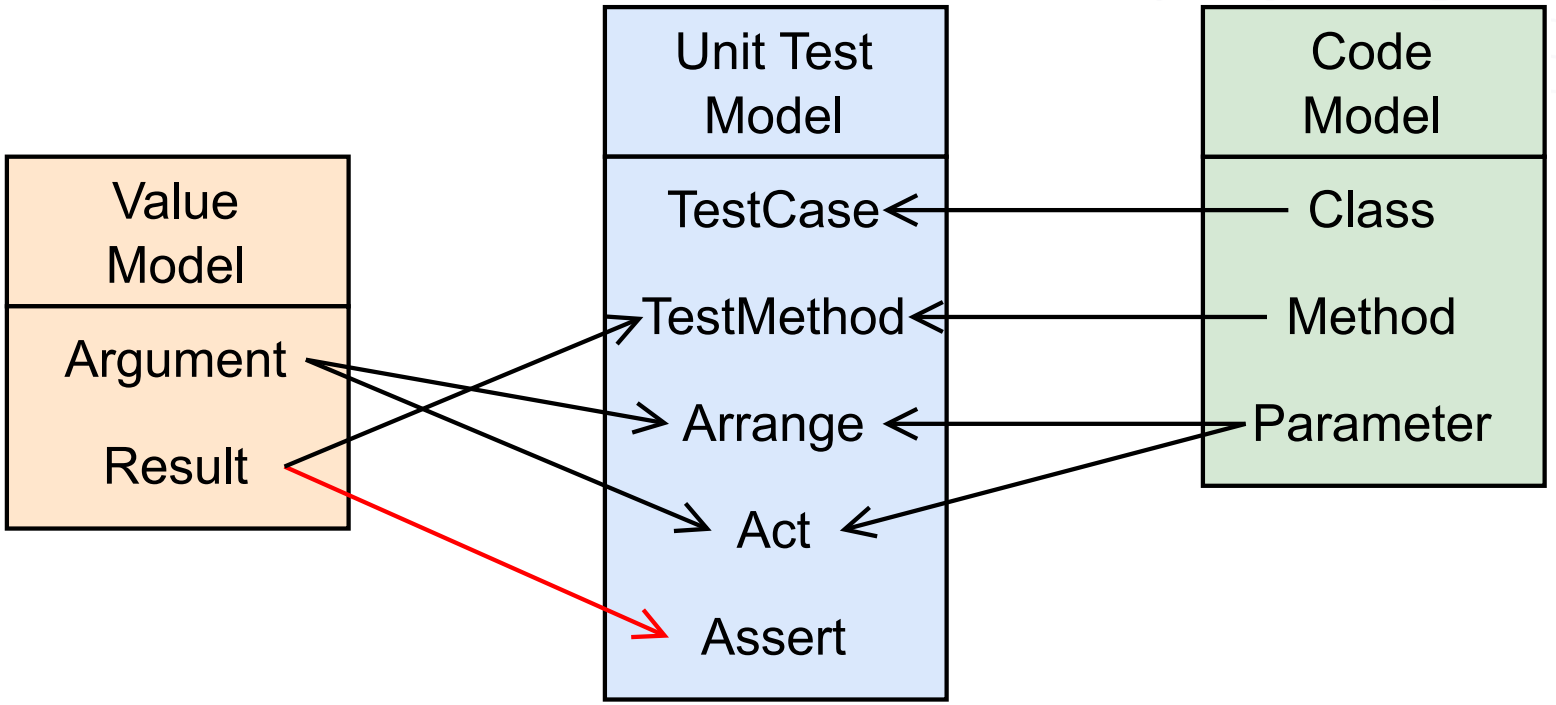
Build Unit Test Using Model Transformations



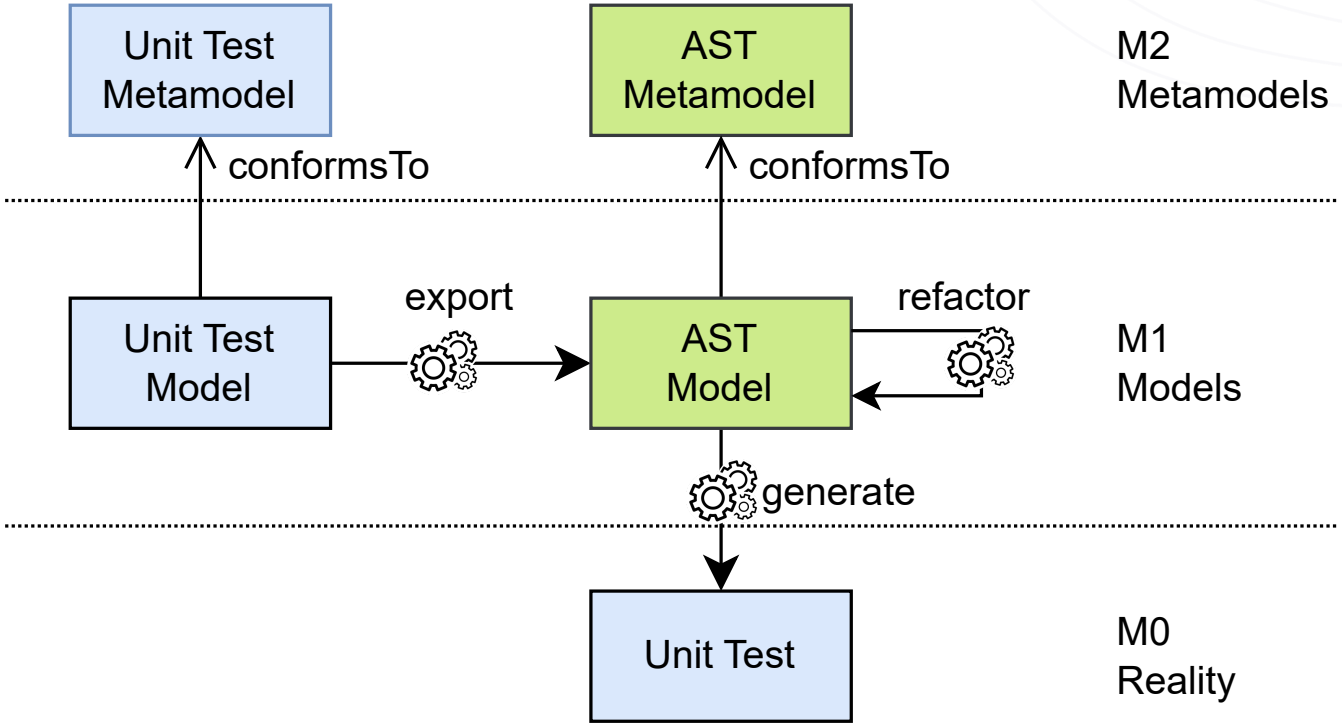
Build Unit Test Using Model Transformations



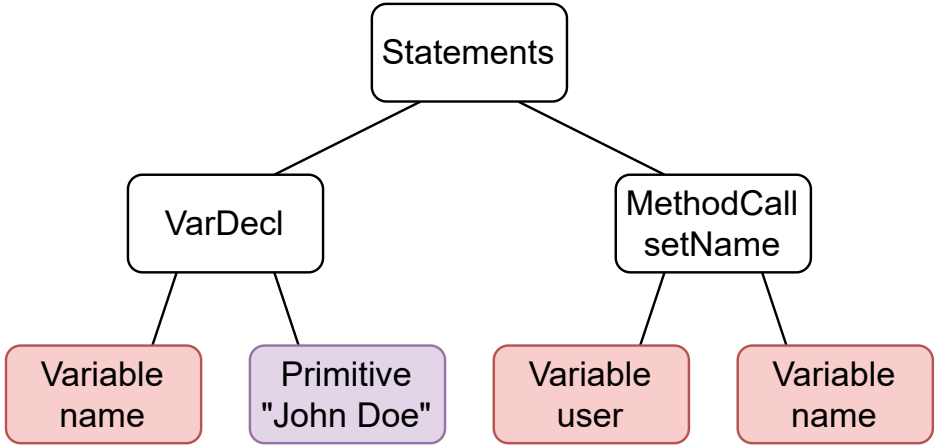
Build Unit Test Using Model Transformations



Export Code Using Abstract Syntax Trees

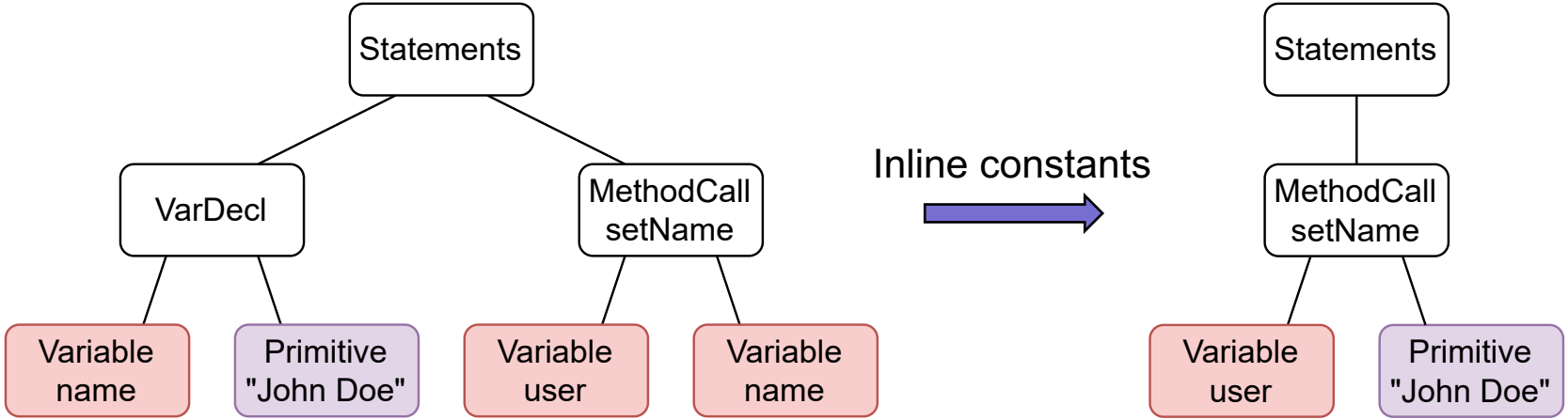


Refactoring Abstract Syntax Trees




```
String name = "John Doe";
user.setName(name);
```

Refactoring Abstract Syntax Trees




```
String name = "John Doe";
user.setName(name);
```

```
user.setName("John Doe");
```

Example of Generated Test for JUnit

Application code

```
class UserManager {
  Set<User> users;

  Set<User> usersByName(String name) {
    Set<User> result = new HashSet<>();
    for (User user: users)
      if (user.getName().equals(name))
        result.add(user);
    return result;
  }
}
```

Generated test code

```
class UserManagerTest {
  UserManager manager = new UserManager();
  @Test void testUsersByName() {
    Set<User> expected = new HashSet<>();
    User user = new User();
    user.setName("John Doe");
    expected.add(user);
    Set<User> actual = manager
                .usersByName("John Doe");
    assertEquals(expected, actual);
  }
}
```

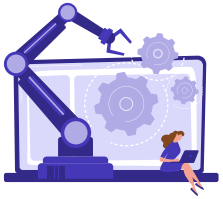
Future Work

- Pursue test quality
  - Relevant, readable, maintainable…
- Evaluate our approach in Pharo, thanks to our community
  - Compare existing tests to generated tests
- Generate unit tests for our industrial partner Berger-Levrault

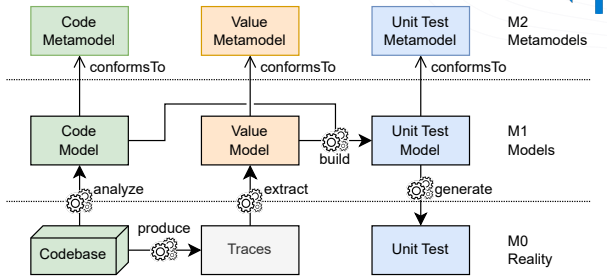
Conclusion

Our Test Generation Approach

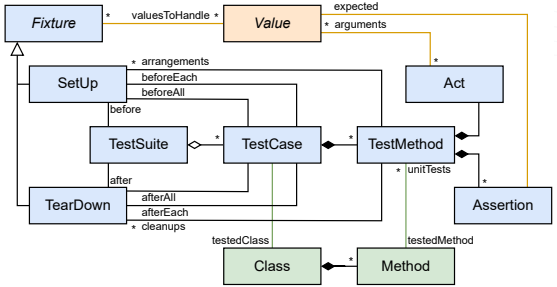
- Using software models and execution traces
  - static and dynamic analysis
- Our objective is to generate tests that are:
  - Relevant
  - Readable
  - Maintainable
  - Not relying on existing tests



An Approach Based on Metamodels



Unit Test Metamodel



Example of Generated Test for JUnit

Application code

```
class UserManager {
  Set<User> users;

  Set<User> usersByName(String name) {
    Set<User> result = new HashSet<>();
    for (User user: users)
      if (user.getName().equals(name))
        result.add(user);
    return result;
  }
}
```

Generated test code

```
class UserManagerTest {
  UserManager manager = new UserManager();
  @Test void testUsersByName() {
    Set<User> expected = new HashSet<>();
    User user = new User();
    user.setName("John Doe");
    expected.add(user);
    Set<User> actual = manager
                .usersByName("John Doe");
    assertEquals(expected, actual);
  }
}
```