Test Design

Analyse et Vérification de Logiciel

Guille Polito — CRIStAL — CNRS — RMoD - Inria Lille
guillermo.polito@inria.fr
@guillep

Automated Testing Refreshment

```
SetTest >> testAdd

  | aSet |
  "Context"
  aSet := Set new.

  "Stimulus"
  aSet add: 5.
  aSet add: 5.

  "Check"
  self assert: aSet size equals: 1.
```

Automated Testing Refreshment

```
SetTest >> testAdd

  | aSet |
  "Context"
  aSet := Set new.

  "Stimulus"
  aSet add: 5.
  aSet add: 5.

  "Check"
  self assert: aSet size equals: 1.
```

in *this context*

Automated Testing Refreshment

```
SetTest >> testAdd

  | aSet |
  "Context"
  aSet := Set new.

  "Stimulus"
  aSet add: 5.
  aSet add: 5.

  "Check"
  self assert: aSet size equals: 1.
```

in *this context*
  when *this happens*

Automated Testing Refreshment

```
SetTest >> testAdd

  | aSet |
  "Context"
  aSet := Set new.

  "Stimulus"
  aSet add: 5.
  aSet add: 5.

  "Check"
  self assert: aSet size equals: 1.
```

in *this context*
  when *this happens*
    then *this should happen*

Why testing?

Why testing?

- **Increase quality!**

- **Detect regressions**: *Wait, this was working before!*
- **Trust changes**: *I'll refactor/change this piece of critical code…*

- ***Murphy's law:*** *Anything that can go wrong will go wrong*

Kinds of testing

- **Unit tests:** low level, single-component
- **Integration testing:** how different modules work together
- **Functional testing:** focus on the business requirements of an application
- **Acceptance testing:** verify minimal business requirements
- **Performance testing:** behaviours the system under significant load
- **Smoke testing:** check that the system *does not fail*

What is a good test?

What is a good test?

“A good test is a test that catches bugs”

- me

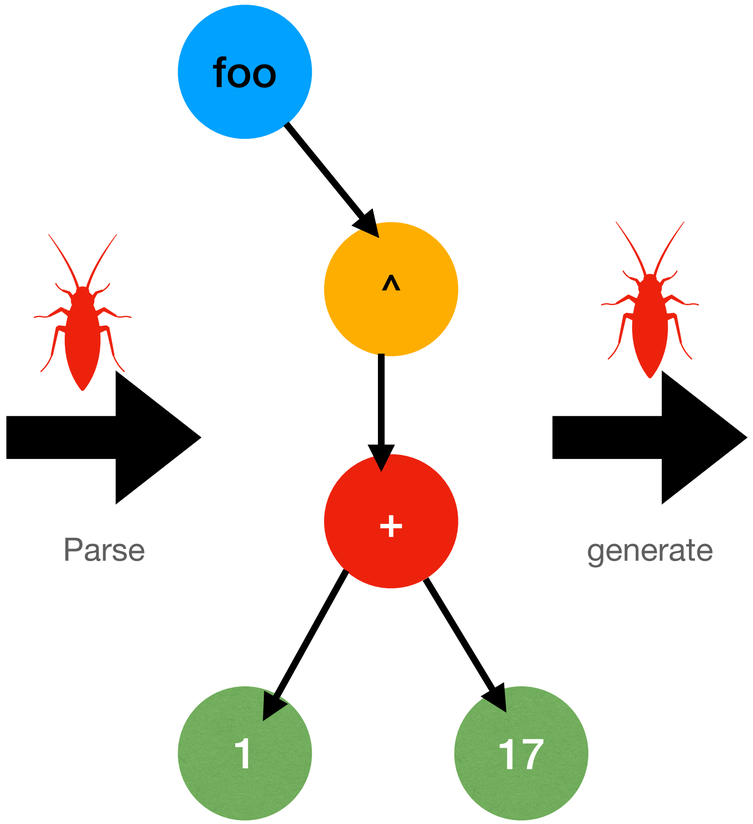
Tests that catch bugs

- check extreme cases (e.g. null, 0, empty, bigger than the collection…)
- check complex cases (e.g. exceptions)
- check different execution paths

Case Study 1: Compiler

Source code

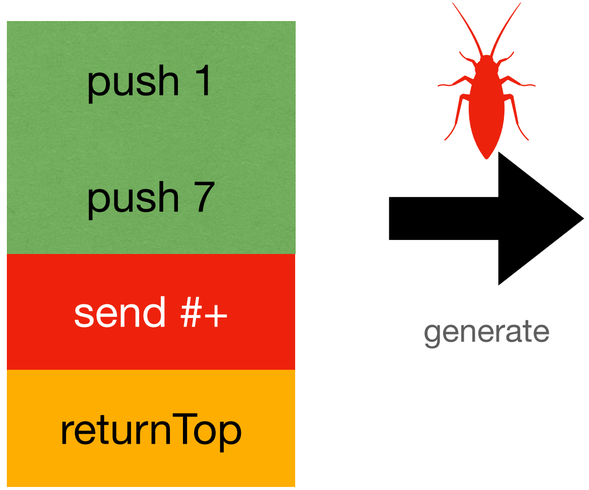
```
MyClass >> foo
    ^ 1 + 17
```



Parse

Intermediate Representation 1: ASTs

generate



Intermediate Representation 2
Linear Bytecode-like IR

generate

Target Code

```
16r76
16r20
16rB0
16r7C
```

Compiler test example

```
testPushConstantZeroBytecodePushesASmallIntegerZero

  self compile: [ compiler genPushConstantZeroBytecode ].
  self runGeneratedCode.

  self assert: self popAddress equals: (memory integerObjectOf: 0)
```

Compiler test example

Insights: Black box testing

Black box testing

=> depend only on **observable behaviour**

=> **reusable** in different backends

=> more **resistant to changes** in the implementation

```
testPushConstantZeroBytecodePushesASmallIntegerZero

  self compile: [ compiler genPushConstantZeroBytecode ].
  self runGeneratedCode.

  self assert: self popAddress equals: (memory integerObjectOf: 0)
```

Compiler test example

Insights: Cross-compile / Cross-execute

http://www.unicorn-engine.org

Use a machine simulator

=> **hardware independent:** test and debug in any machine any backend

=> **parametrizable tests** run the same test with multiple backends

```
testPushConstantZeroBytecodePushesASmallIntegerZero

  self compile: [ compiler genPushConstantZeroBytecode ].
  self runGeneratedCode.

  self assert: self popAddress equals: (memory integerObjectOf: 0)
```

Compiler test example

Insights: Start Small

- First: The **simplest** test you can write for the **simplest** functionality
- Second: The **next simplest** test you can write for the **next simplest** functionality

=> The first focus is in understanding **how to** better write the **tests**

```
testPushConstantZeroBytecodePushesASmallIntegerZero

  self compile: [ compiler genPushConstantZeroBytecode ].
  self runGeneratedCode.

  self assert: self popAddress equals: (memory integerObjectOf: 0)
```

Compiler test example

Insights: Invest in infrastructure

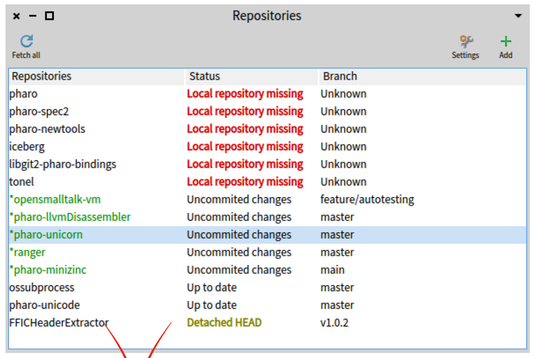
- **Refactor**
- **Clean**
- **Create Reusable Components**

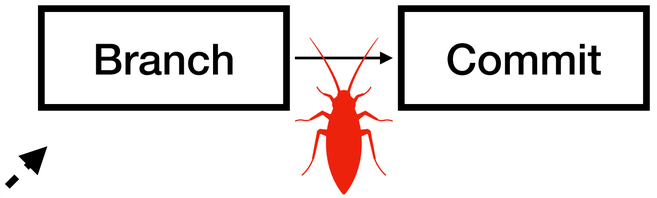
```
testPushConstantZeroBytecodePushesASmallIntegerZero

  self compile: [ compiler genPushConstantZeroBytecode ].
  self runGeneratedCode.

  self assert: self popAddress equals: (memory integerObjectOf: 0)
```

Case Study 2: Git Client



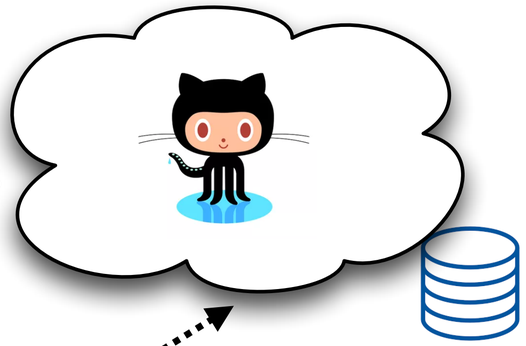


LibGit

Git Client

Git Repository

Remote repositories



Working Copy

Index

Repository

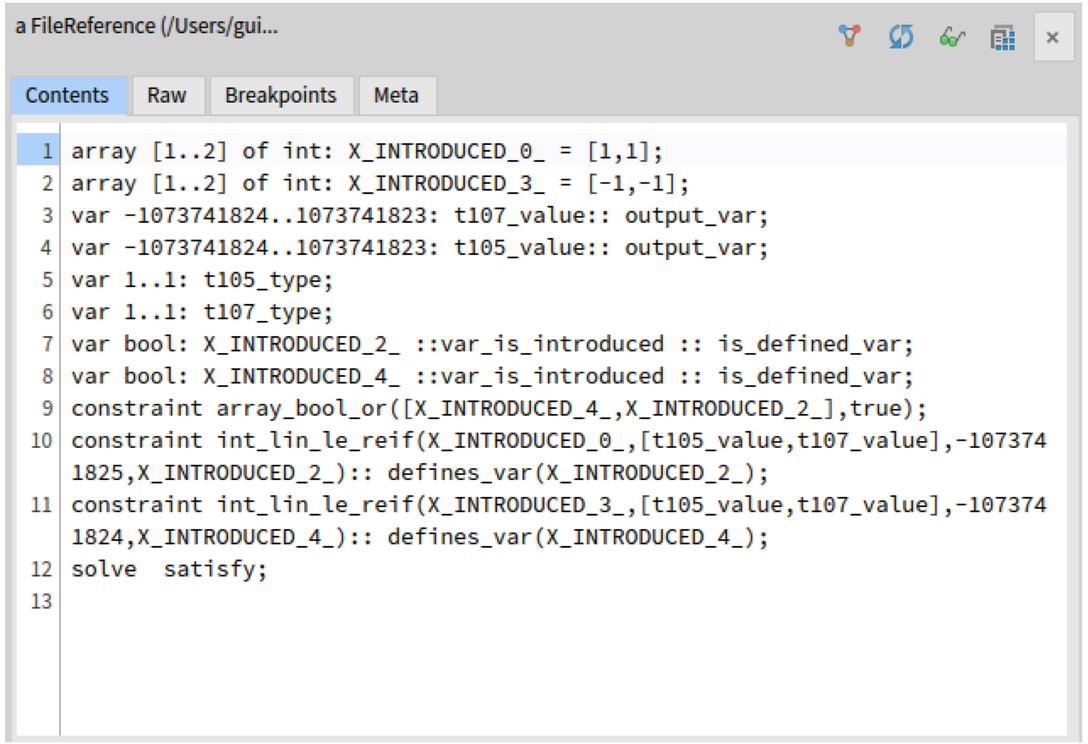
What you see

A “hidden” staging area (sort of)

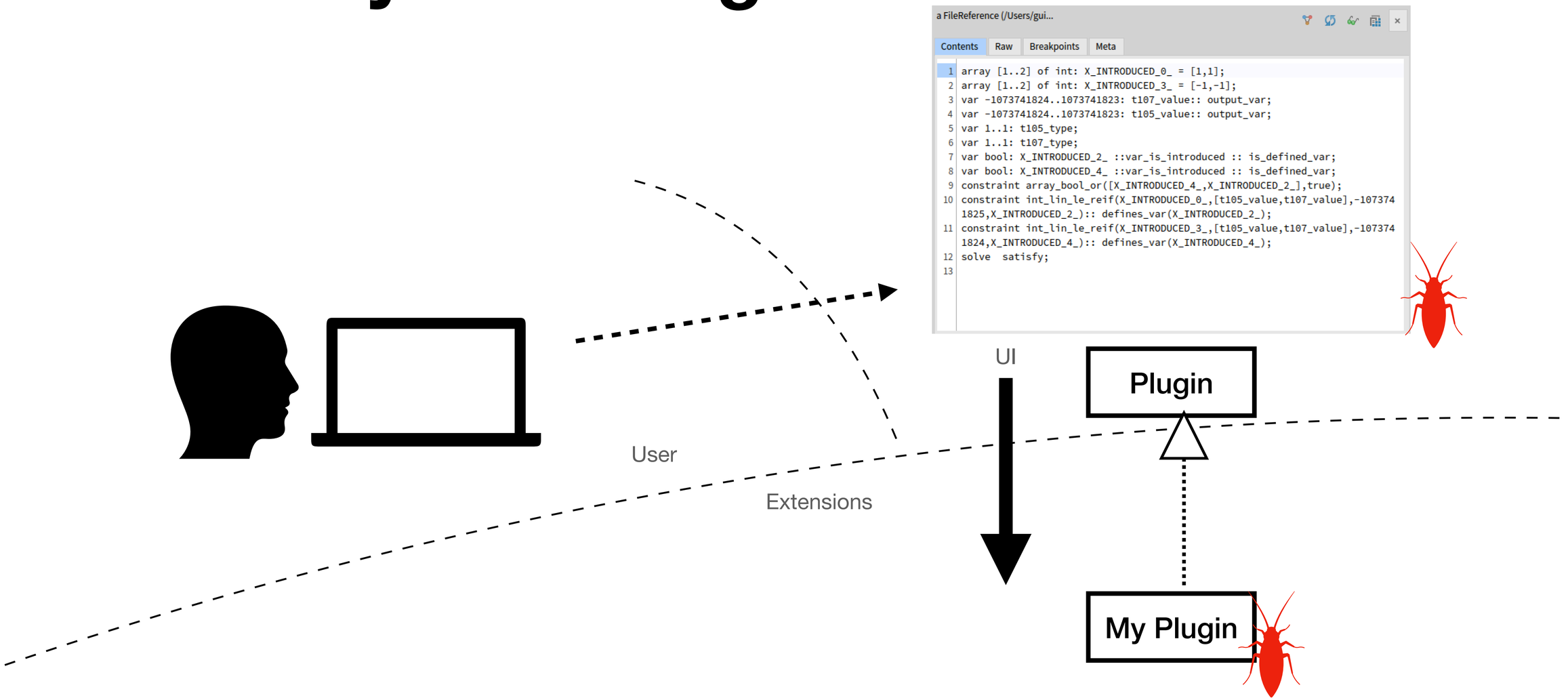
Where **commits** are

Case Study 3: UI Plugins

```
File >> gtInspectorContentsIn: composite
  <gtInspectorPresentationOrder: 5>
  composite text
    title: ‘Contents';
    format: #asText;
    display: [ … ]
```



Case Study 3: UI Plugins



Tests that are challenging to write

- non-deterministic behavior
- user-interactions
- external interactions

Conclusion

- *Anything that can go wrong will go wrong!*
- *Tests give freedom to change without fear*

- *Testing Guidelines*
  - *Start simple, then “what happens if…?”*
  - *Make tests resistant to implementation changes*
  - *Tests are code!*