

M0.1 - Testing Refreshment

Guillermo Polito

Automated Tests

```
SetTest >> testAdd
```

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

```
"Stimuli"  
aSet add: 5.  
aSet add: 5.
```

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

Automated Tests - Context

SetTest >> testAdd

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

in this context

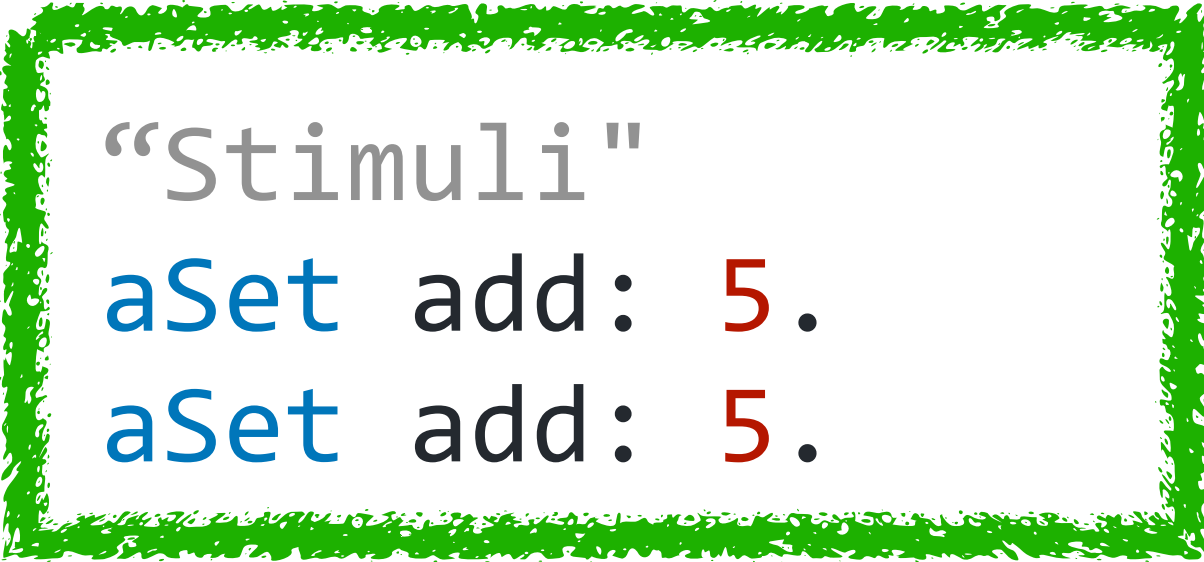
```
"Stimuli"  
aSet add: 5.  
aSet add: 5.
```

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

Automated Tests - Stimuli

SetTest >> testAdd

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



```
"Stimuli"  
aSet add: 5.  
aSet add: 5.
```

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

in this context
when this happens

Automated Tests - Assertions

SetTest >> testAdd

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

```
"Stimuli"  
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:** behaviors 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

Rule of Thumb: the RIGHT BICEP principle

- **Right** - Check if the results are right
- **B** - Check boundary cases
- **I** - Check Inverse conditions
- **C** - Cross-check results with other sources
- **E** - Check error conditions
- **P** - Check performance

Set Example - Right

```
SetTest >> testAdd
```

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

```
"Stimuli"  
aSet add: 5.  
aSet add: 5.
```

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

- Elements are added
- Duplicated elements are ignored
- Iteration yields all elements

Set Example - Boundary

```
SetTest >> testDoEmpty
```

```
| aSet |  
"Context"
```

```
aSet := Set new.
```

```
c := 0.
```

```
"Stimuli"
```

```
aSet do: [:e | c := c + 1 ].
```

```
"Check"
```

```
self assert: c equals: 0.
```

- Set starts empty
- Iteration works on empty collection
- Upper bounds?
 - Sets in Pharo are bound by the memory...

Set Example - Inverse Relationships

```
SetTest >> testRemoveTwice
```

```
| aSet |  
"Context"  
aSet := Set new.  
aSet add: 5.  
aSet add: 5.  
  
"Stimuli"  
aSet remove: 5.  
  
"Check"  
self  
  should: [ aSet remove: 5 ]  
  raise: NotFound
```

- Add two times an element, remove it twice

Set Example - Cross check

```
SetTest >> testCrossSet
```

```
| aSet aTreeSet |
```

```
"Context"
```

```
aSet := Set new.
```

```
aTreeSet := TreeSet new.
```

```
"Stimuli"
```

```
aSet add: 5; add: 5.
```

```
aTreeSet add: 5; add: 5.
```

```
"Check"
```

```
self
```

```
assert: aSet size
```

```
equals: aTreeSet size
```

- Compare different set implementations

Set Example - Error conditions

```
SetTest >> testAnyEmptySet
```

```
| aSet aTreeSet |
```

```
"Context"
```

```
aSet := Set new.
```

```
aTreeSet := TreeSet new.
```

```
"Check"
```

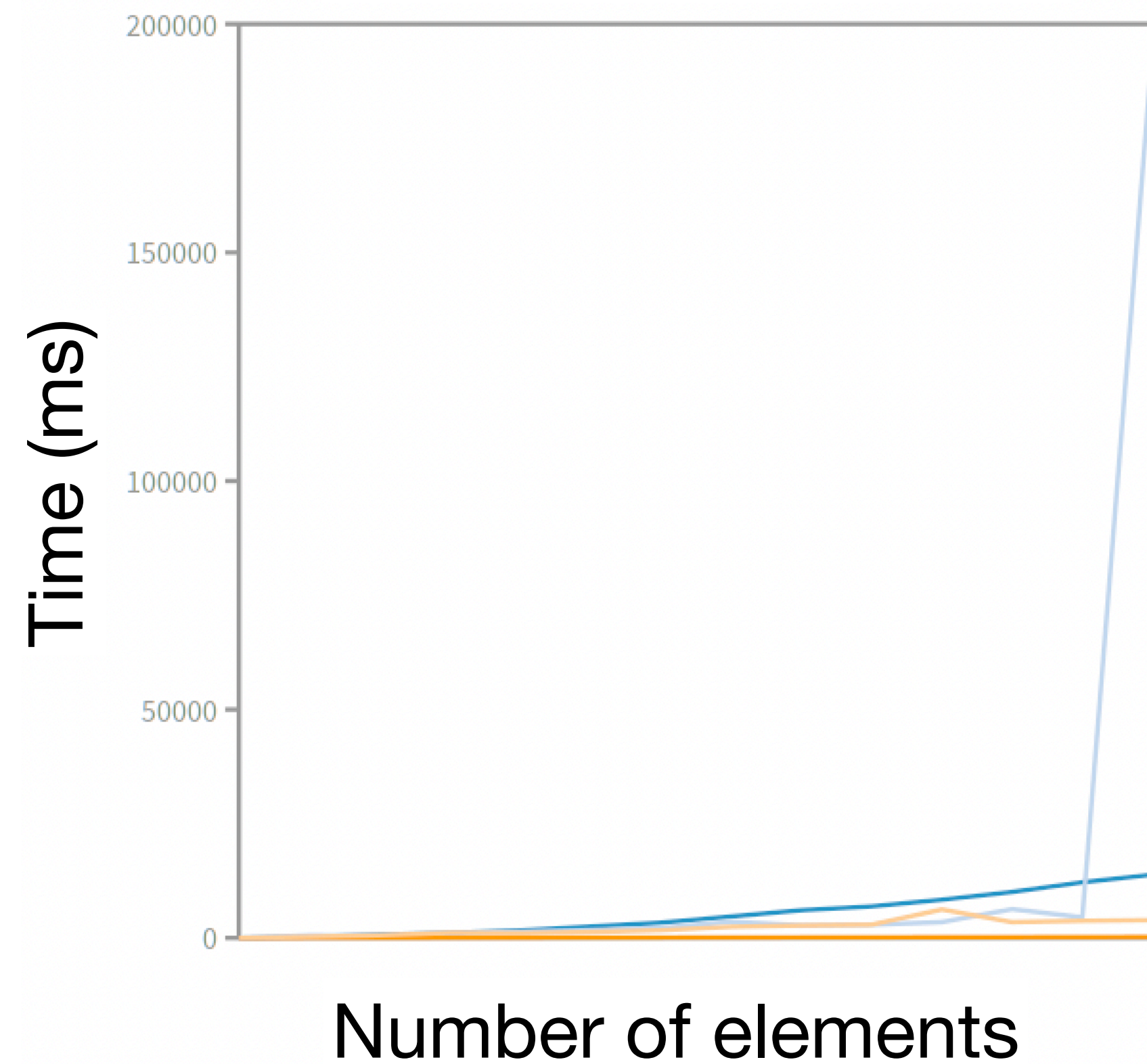
```
self
```

```
should: [ aSet anyOne ]
```

```
raise: CollectionIsEmpty
```

- Accessing elements in an empty set
- Remove an element that never was there

Set Example - Performance



- Scalability to many elements
 - adding
 - removing
- Assert => set performance expectations

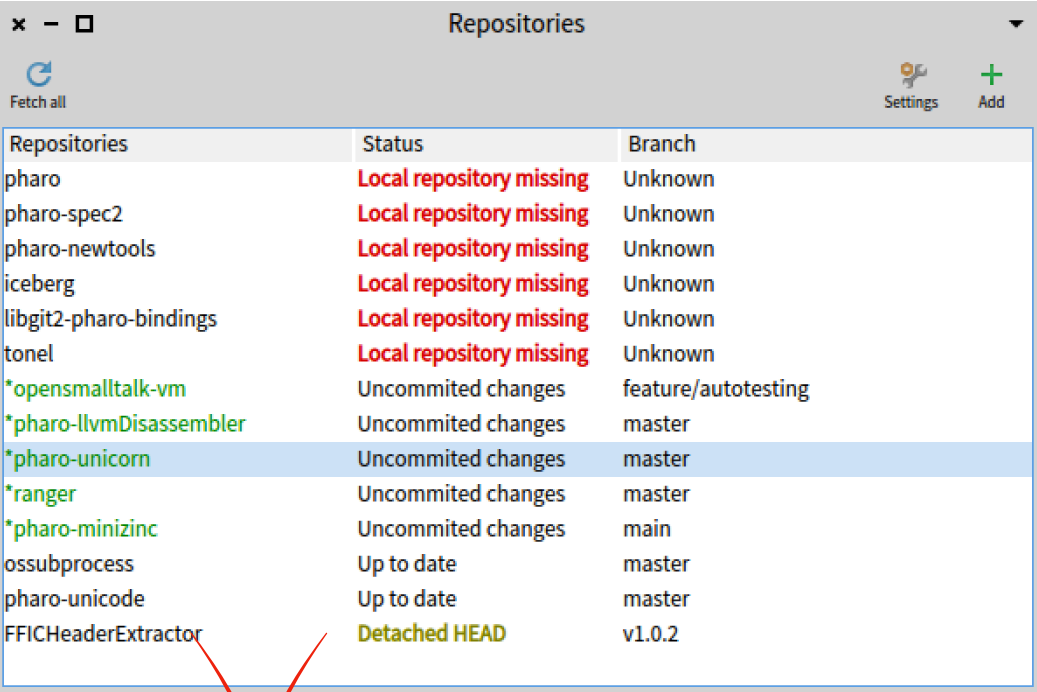
Challenging tests

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

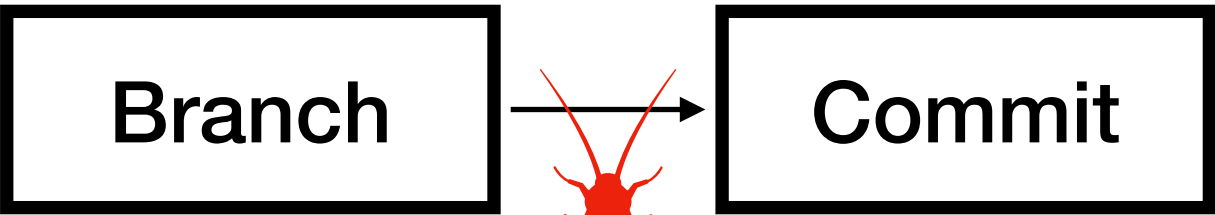


- **non-deterministic => deterministic**
 - Mocks
 - Control random seeds
 - Simulations

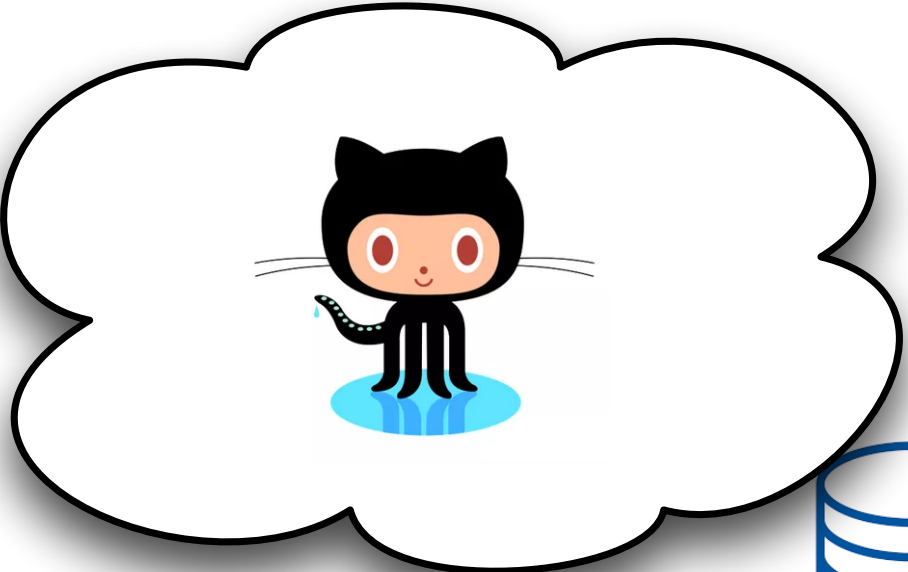
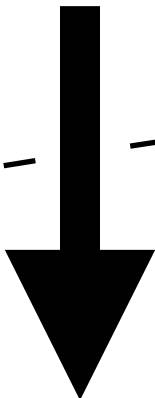
Case Study: Git Client



Repositories	Status	Branch
pharo	Local repository missing	Unknown
pharo-spec2	Local repository missing	Unknown
pharo-newtools	Local repository missing	Unknown
iceberg	Local repository missing	Unknown
libgit2-pharo-bindings	Local repository missing	Unknown
tonel	Local repository missing	Unknown
*opensmalltalk-vm	Uncommitted changes	feature/autotesting
*pharo-llvmDisassembler	Uncommitted changes	master
*pharo-unicorn	Uncommitted changes	master
*ranger	Uncommitted changes	master
*pharo-minizinc	Uncommitted changes	main
ossubprocess	Up to date	master
pharo-unicode	Up to date	master
FFICHeaderExtractor	Detached HEAD	v1.0.2



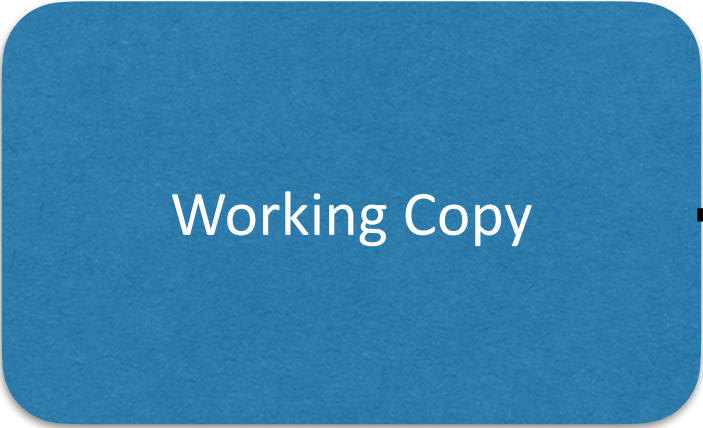
LibGit



Git Client

Git Repository

Remote
repositories



What you see



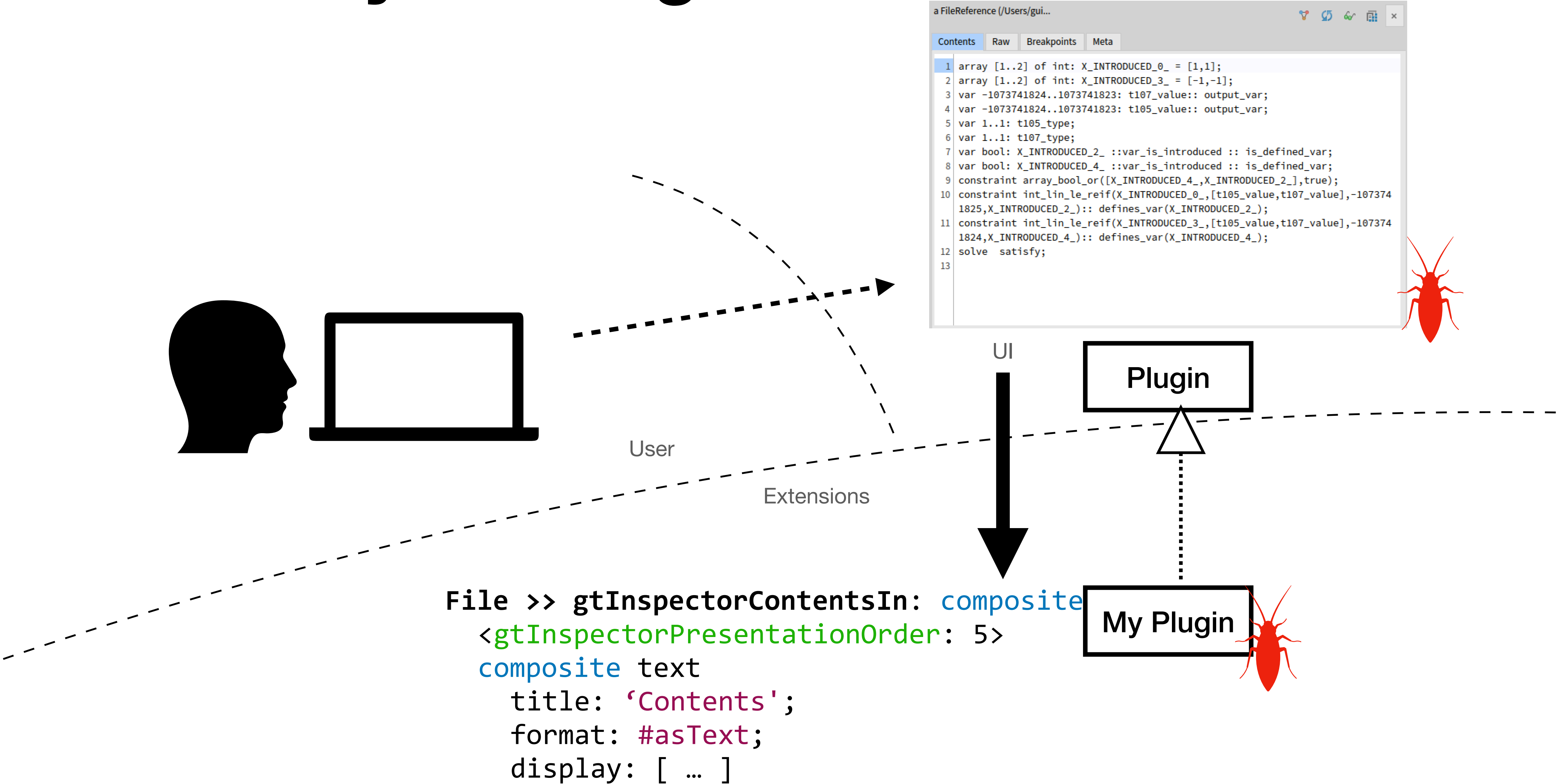
A “hidden”
staging area
(sort of)



Where
commits are



Case Study: UI Plugins



Material

- Learning Pharo: [Mooc](https://mooc.pharo.org/)
 - Week 1 Lecture 4: 🐥 Pharo Object Model in a Nutshell
 - Week 1 Lecture 5: 🐥 Lecture Pharo Syntax in a Nutshell
 - Week 1 Lecture 6: 🐥 Lecture Class and Method Definitions
- Introduction to SUnit unit testing
 - Pharo by example book: <https://books.pharo.org/updated-pharo-by-example/>
 - [Mooc](https://mooc.pharo.org/) Week 5 lecture 6: 🐥 SUnit: Unit Tests in Pharo
- Fuzzing Book, introduction to software testing: https://www.fuzzingbook.org/html/Intro_Testing.html