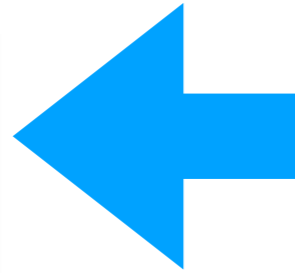


The future of testing in Pharo

Julien Delplanque

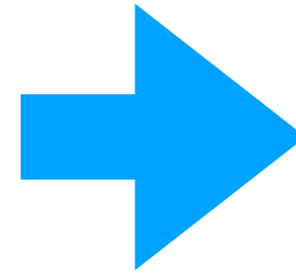
julien.delplanque@inria.fr

About me



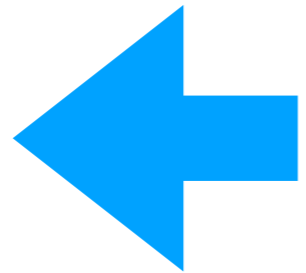
This is me on discord.

This is me on github
and twitter.



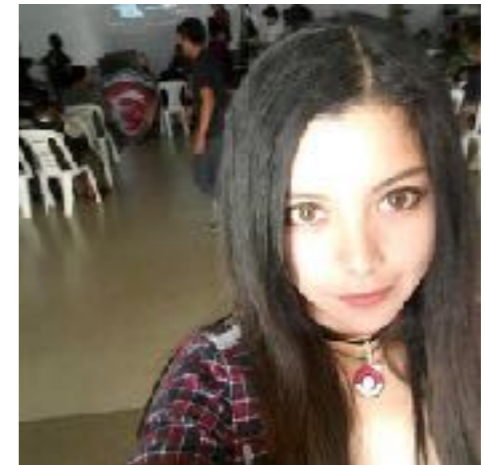
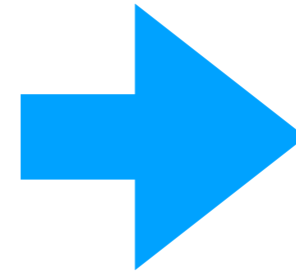
- 2nd year of PhD in RMoD team
- Hacking Pharo around many aspects... just for fun :-)
- Hit by testing topic « by accident »

The one who work on this topic right now



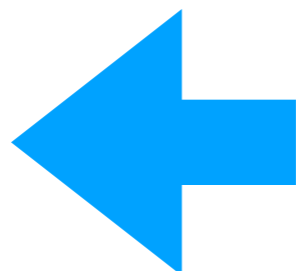
This is Dayne on discord.

This is Dayne on github.

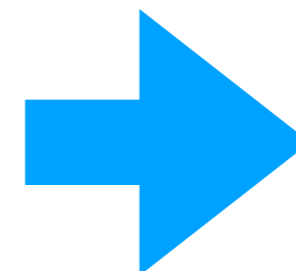


- 2nd year of Master
- Doing an internship in RMoD team for 6 months
- Master thesis on the enhancements of tests in Pharo

Guille



Her other mentors on Github



Steph

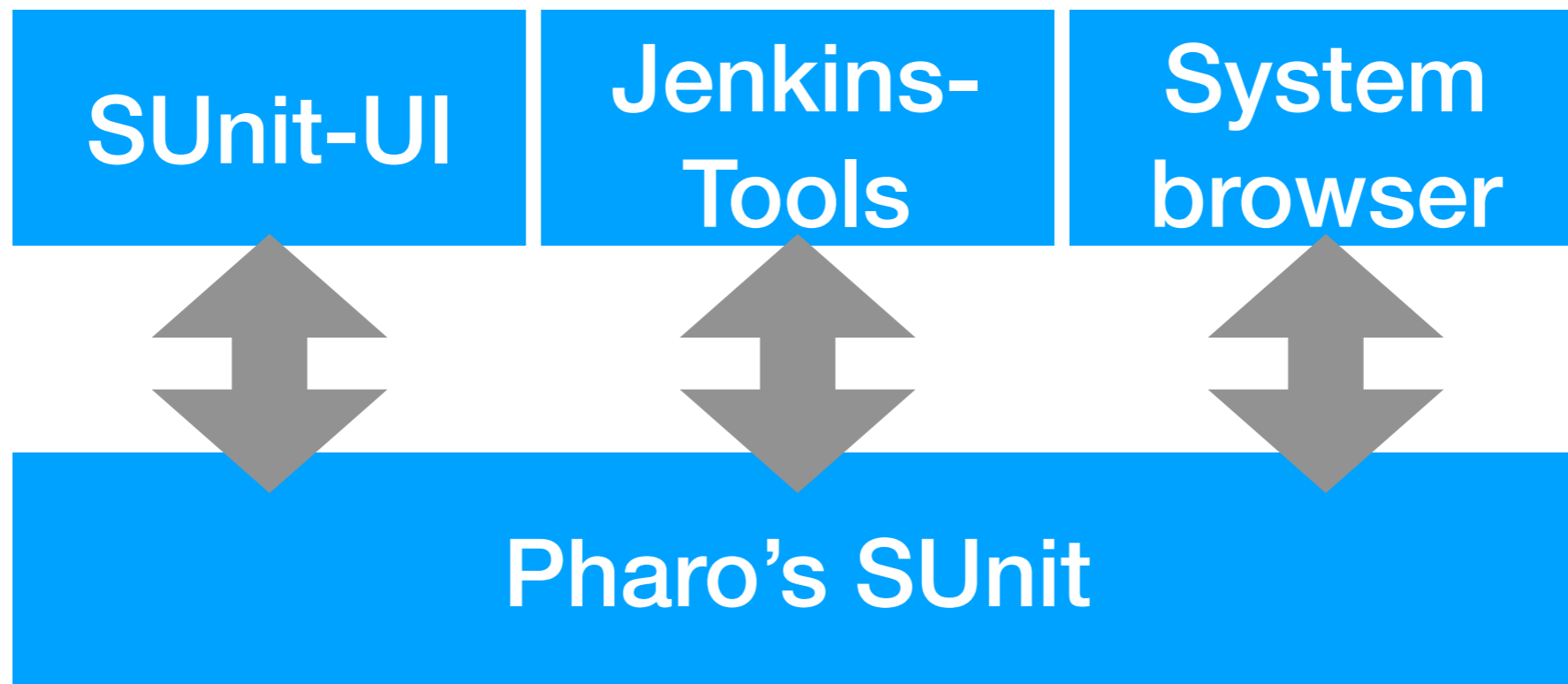


Questions this talk addresses

- What infrastructure Pharo provides for testing?
- How can we enhance testing experience?

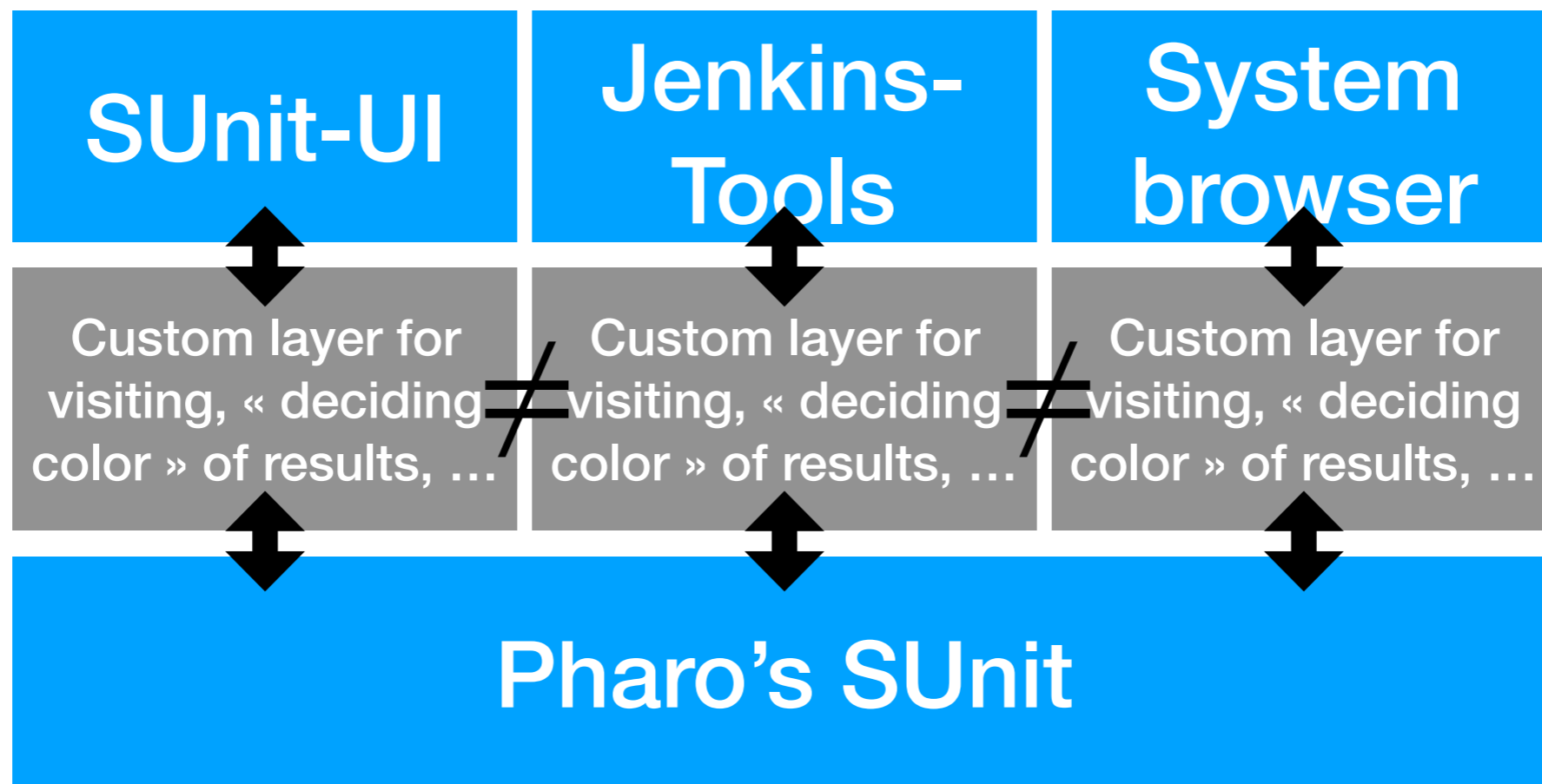
**What infrastructure Pharo
provides for testing?**

What infrastructure Pharo provides for testing?

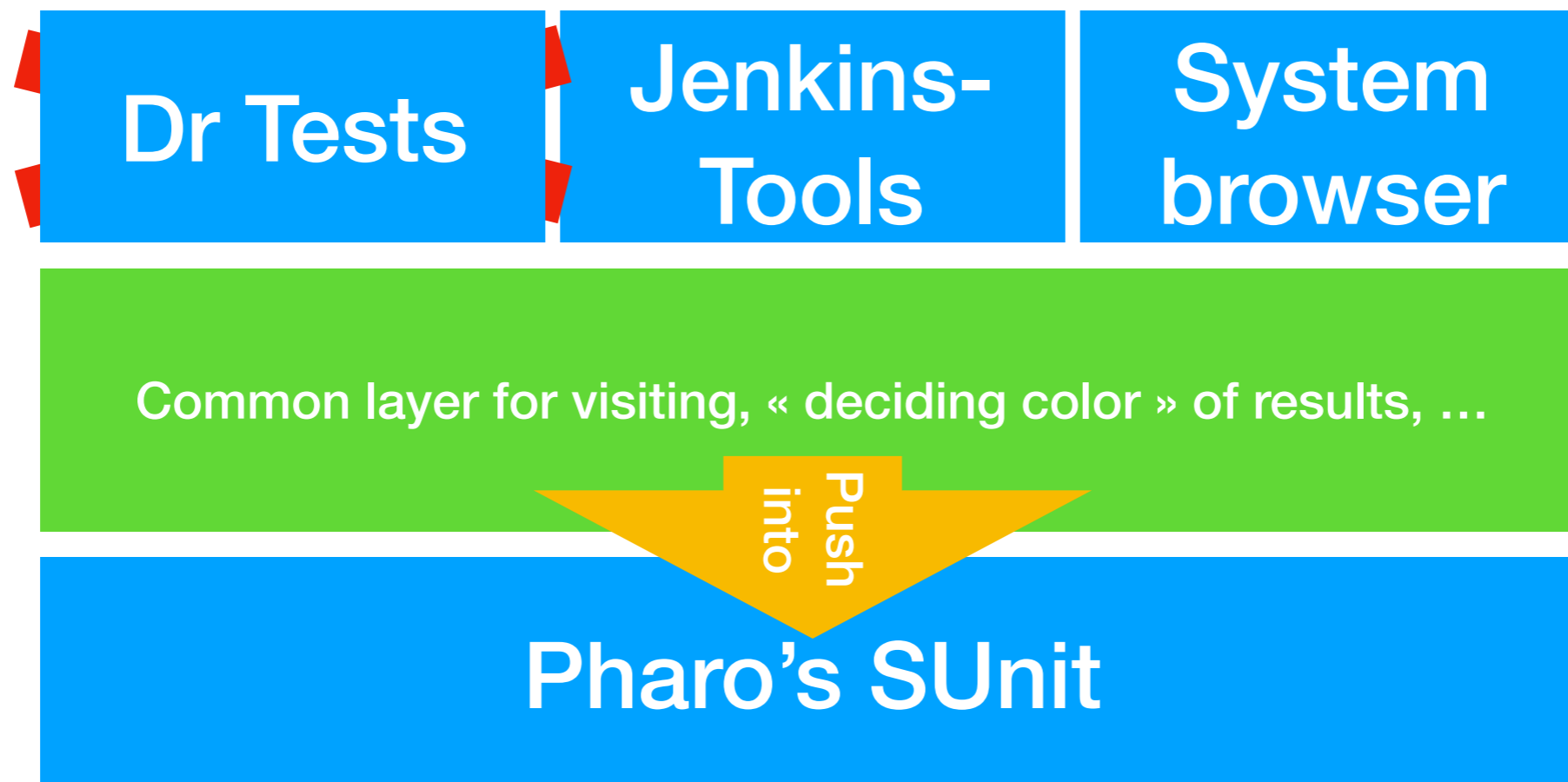


What infrastructure Pharo provides for testing?

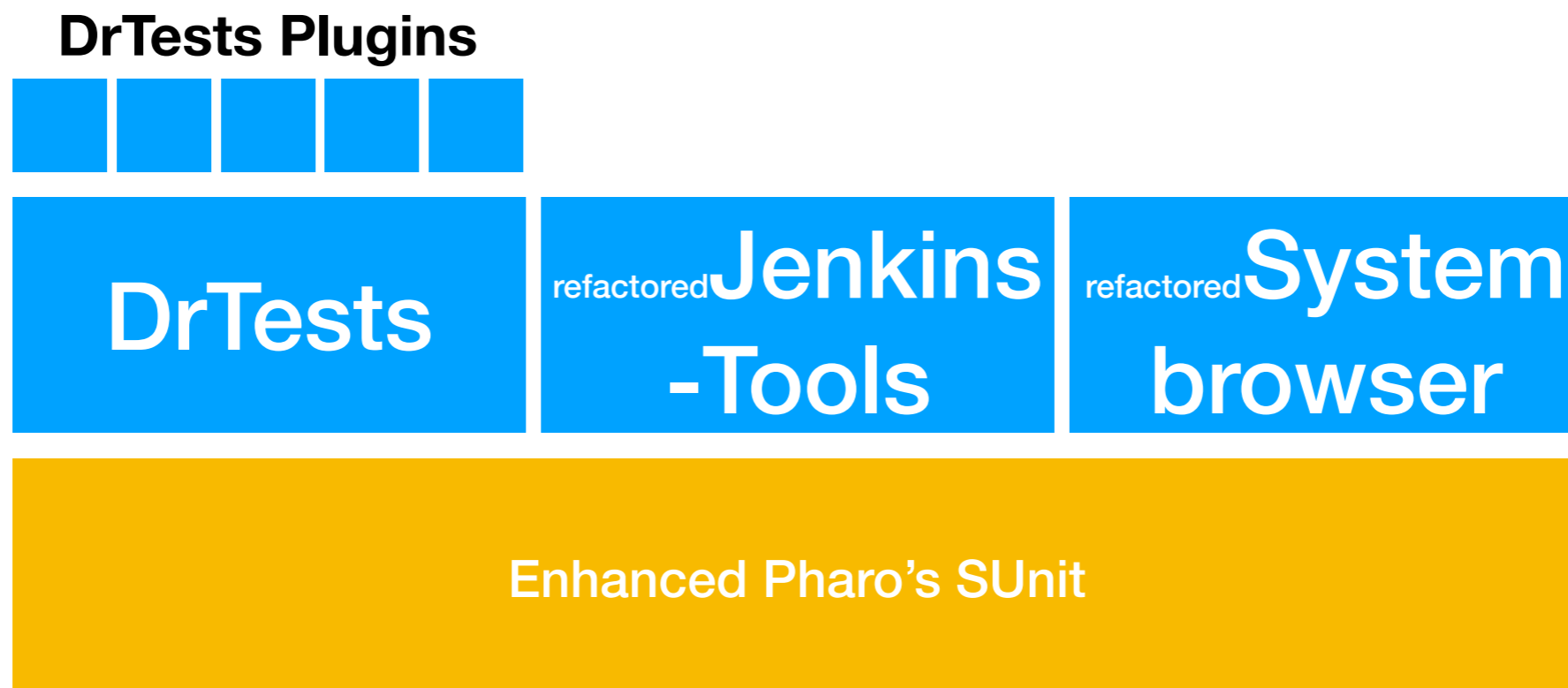
In fact,



The infrastructure we want



The infrastructure we want



**How can we enhance
testing experience?**

What can we do with tests?

		Managed by SUnit-UI			Parametrisable test	Mutation testing	Run examples in comment	Rotten Tests Finder	...
		Runner	Coverage	Profile					
Input		Test cases	Test cases	Test cases	Test cases + parameters	Test cases + mutations	Comments	Test cases	...
Output		Results of tests run	Percentage of methods covered	Time taken for each test run	Results of tests run depending on parameter	Mutants to be killed	Comments containing failing examples	Rotten tests	...

TestRunner UI

The image shows a screenshot of the TestRunner application window. The window is titled "Test Runner" and contains several components:

- Packages containing tests:** A tree view on the left showing a hierarchy of packages under "Network-". The selected package is "Network-Tests-UUID".
- TestCases:** A list of test cases under the selected package. The selected test case is "TestCaseA|TestCaseB".
- Results summary:** A red bar at the top right displaying the summary: "166 run, 164 passes, 2 skipped, 0 expected failures, 1 failures, 1 errors, 0 unexpected passes".
- Test Output:** A text area below the summary showing the output of the test cases, including "Base64Tests>>#testRangeEncoding" and "Base64Tests>>#testPrimesEncoding".
- Buttons:** A row of buttons at the bottom: "Run Selected", "Run Profiled", "Run Coverage", "Run Failures", "Run Errors", and "File out results".

Annotations with lines pointing to the UI elements:

- "Run tests" points to the "Run Selected" button.
- "Profile test execution" points to the "Run Profiled" button.
- "Analyse code coverage" points to the "Run Coverage" button.
- "Failed tests" points to the "Run Failures" button.
- "Re-run failures or errors only" points to the "Run Errors" button.
- "Export results" points to the "File out results" button.
- "Errors" points to the "Run Errors" button.

TestRunner UI: coverage

The screenshot displays the TestRunner interface. At the top, a window titled "Test Runner" contains a sub-window "Not Covered Code (30% Code Coverage) [278]". This sub-window lists 15 items, each with a class name, a method name, and a package name. The first item is highlighted in blue.

Class	Method	Package
ConnectionQueue	connectionCount	Network Kernel
ConnectionQueue	destroy	Network-Kernel
ConnectionQueue	getConnectionOrNil	Network-Kernel
ConnectionQueue	getConnectionOrNilLenient	Network-Kernel
ConnectionQueue	initPortNumber:queueLength:	Network-Kernel
ConnectionQueue	isValid	Network-Kernel
ConnectionQueue	listenLoop	Network-Kernel
ConnectionQueue	oldStyleListenLoop	Network-Kernel
ConnectionQueue class	portNumber.queueLength:	Network-Kernel
ConnectionQueue	pruneStaleConnections	Network Kernel
ConnectionRefused	host	Network-Kernel
ConnectionRefused	host:port:	Network-Kernel
ConnectionRefused class	host:port:	Network-Kernel

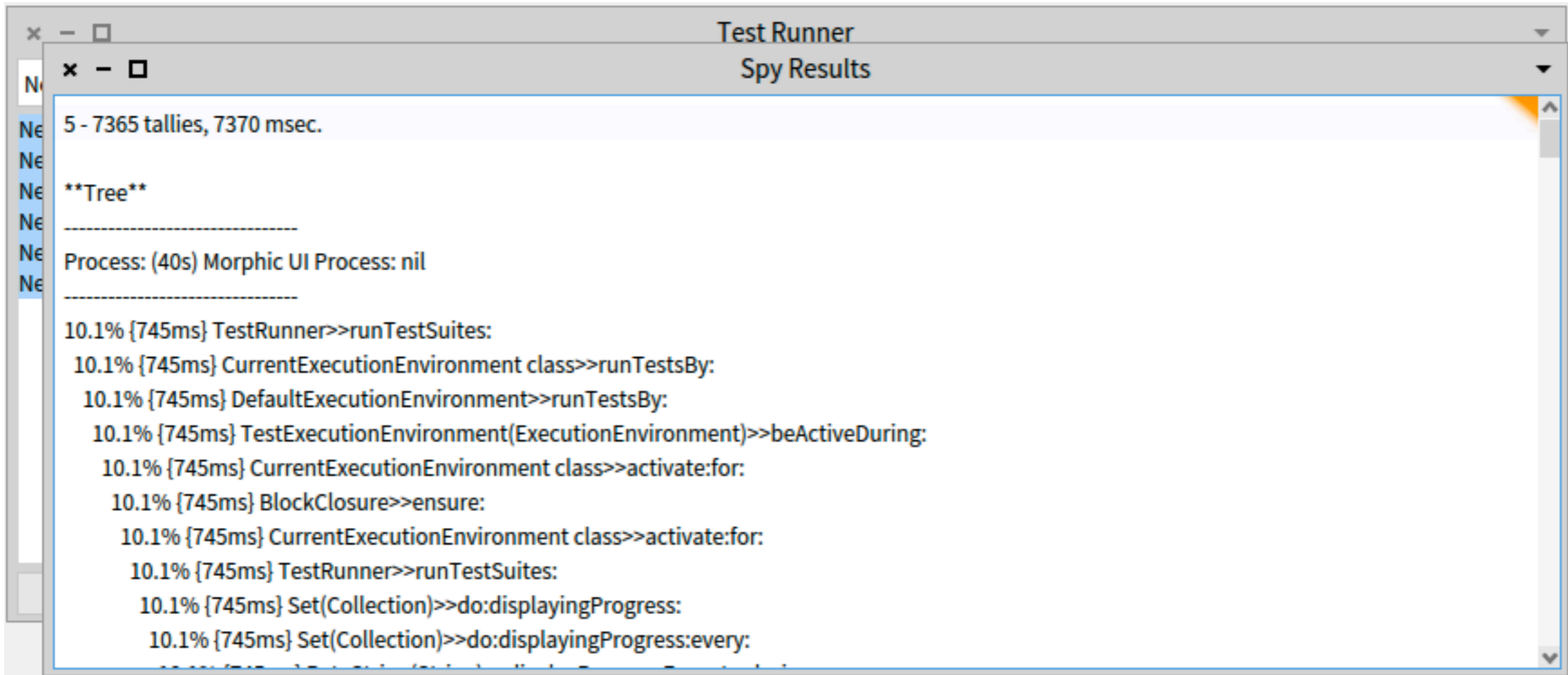
Below the list is a "Filter..." input field. The main interface shows a "Current image" dropdown set to "Flat", with buttons for "Browse", "Class refs.", "Implementors", and "Senders". A tab labeled "connectionCount x" is active, showing the following code:

```
connectionCount
  "Return an estimate of the number of currently queued connections. This is only an estimate since a
  new connection could be made, or an existing one aborted, at any moment."

  self pruneStaleConnections.
  ^accessSema critical: [connections size]
```

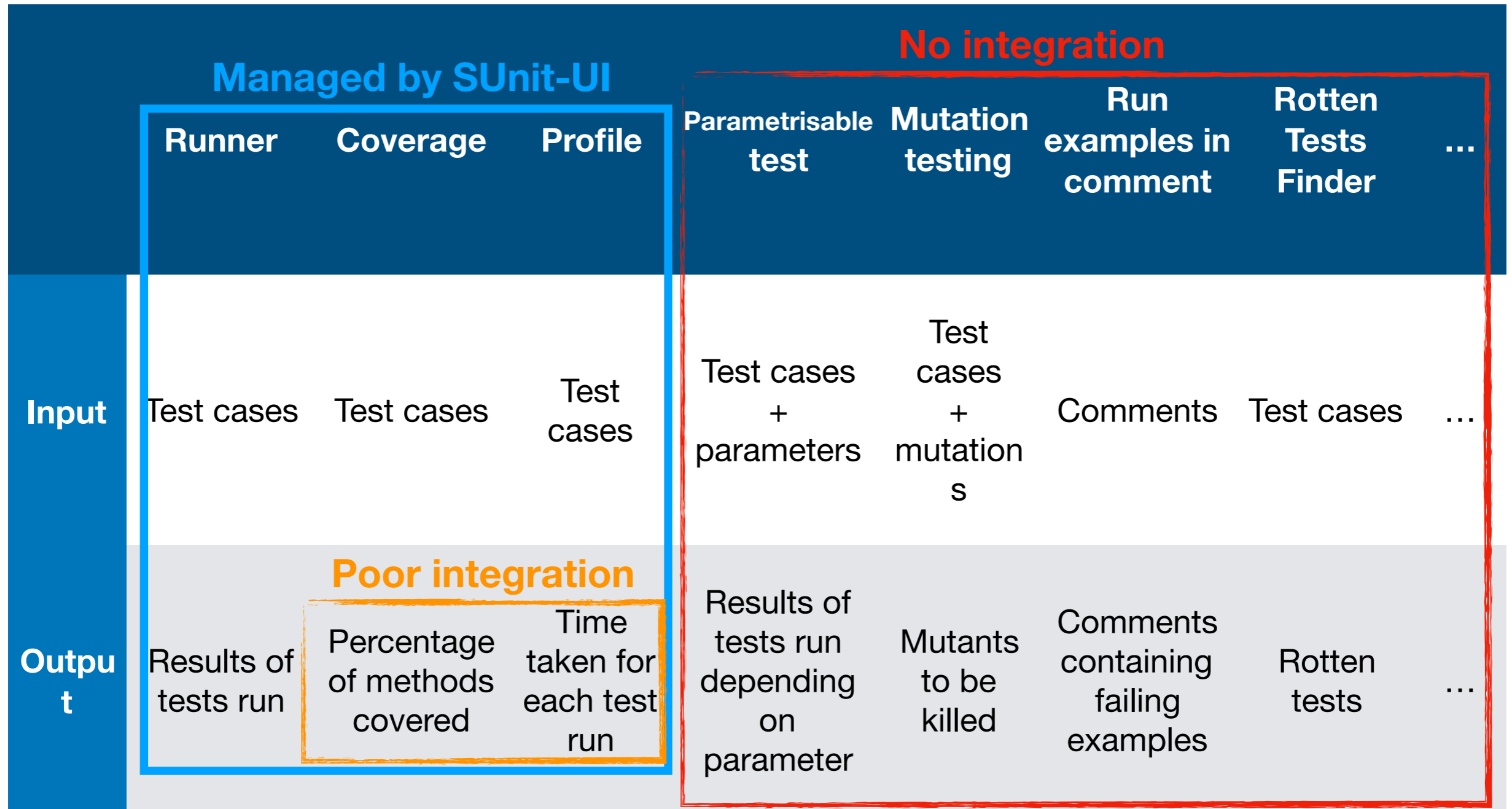
The status bar at the bottom indicates "1/89 [1]" and "public extension F +L W".

TestRunner UI: profile



```
5 - 7365 tallies, 7370 msec.  
  
**Tree**  
-----  
Process: (40s) Morphic UI Process: nil  
-----  
10.1% {745ms} TestRunner>>runTestSuites:  
  10.1% {745ms} CurrentExecutionEnvironment class>>runTestsBy:  
    10.1% {745ms} DefaultExecutionEnvironment>>runTestsBy:  
      10.1% {745ms} TestExecutionEnvironment(ExecutionEnvironment)>>beActiveDuring:  
        10.1% {745ms} CurrentExecutionEnvironment class>>activate:for:  
          10.1% {745ms} BlockClosure>>ensure:  
            10.1% {745ms} CurrentExecutionEnvironment class>>activate:for:  
              10.1% {745ms} TestRunner>>runTestSuites:  
                10.1% {745ms} Set(Collection)>>do:displayingProgress:  
                  10.1% {745ms} Set(Collection)>>do:displayingProgress:every:
```

What can we do with tests?



Proposal: Dr Tests

Power-up testing experience in Pharo by:

Developing and promoting DrTests as the new UI for testing

- ▶ Written in Spec
- ▶ Extensible via plugins
- ▶ Provides good model to configure, run and gather results from plugins

Dr Tests

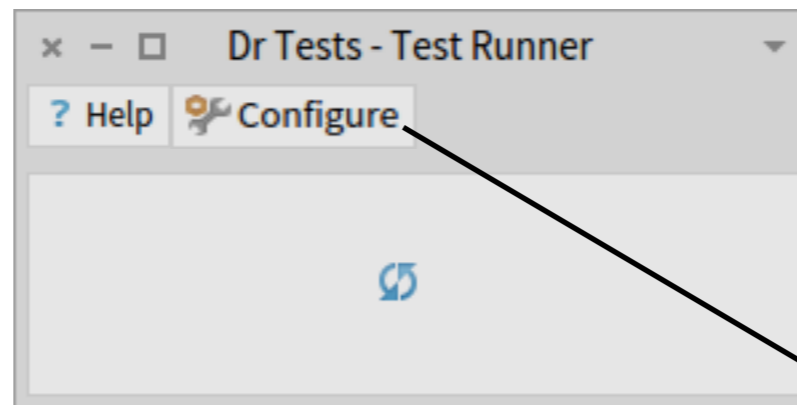
The screenshot shows the 'Dr Tests - Tests Runner' window. It is divided into several sections:

- Plugin selected:** A dropdown menu at the top left shows 'Tests Runner' selected.
- Packages under analysis:** A list on the left side shows various packages, with 'RottenTestsFinder-Tests' selected.
- Plugin input:** A list in the middle shows 'Test cases (7 selected)', including 'RTFFakeTestClass', 'RTFFakeTestSuperClass', and others.
- Results tree:** A panel on the right shows 'Results:' with 'Errors(0)', 'Failures(1)', 'Skipped tests(0)', and 'Passed tests(17)'. A 'Re-run' button is visible next to the failure.
- Plugin-defined action(s):** A 'Browse test' button is located at the bottom right of the results panel.
- Logging label:** A status bar at the bottom left displays '2018-07-02 14:45: Tests finished.'
- Start plugin:** A 'Run Tests' button is located at the bottom center.

Annotations with vertical lines point from the labels to the corresponding UI elements in the screenshot.

Mini Dr Tests

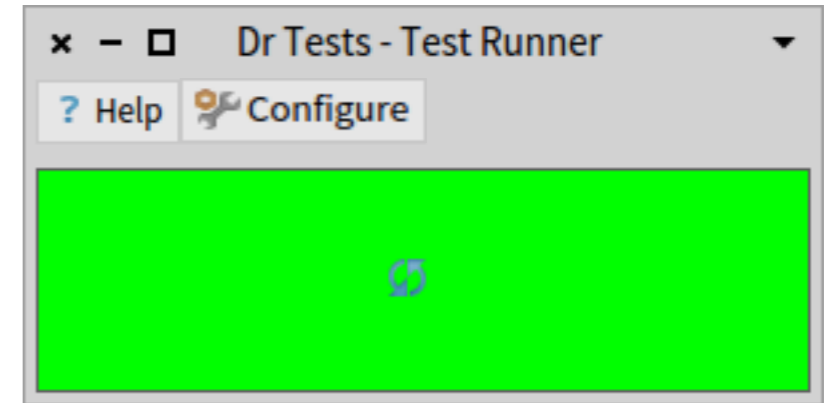
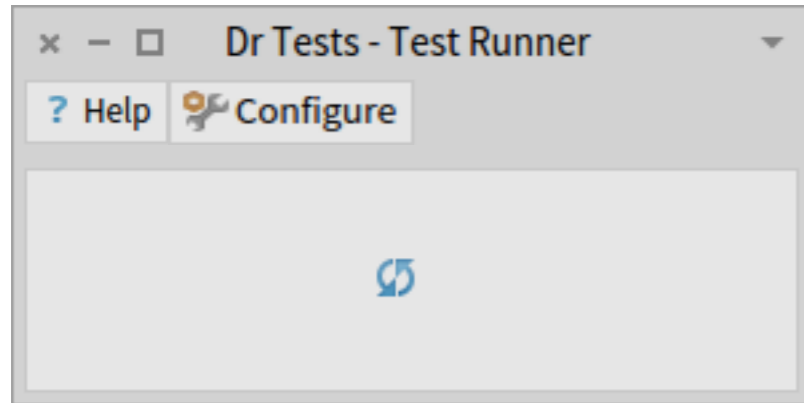
➔ **Simplified view to use a plugin once it is configured**



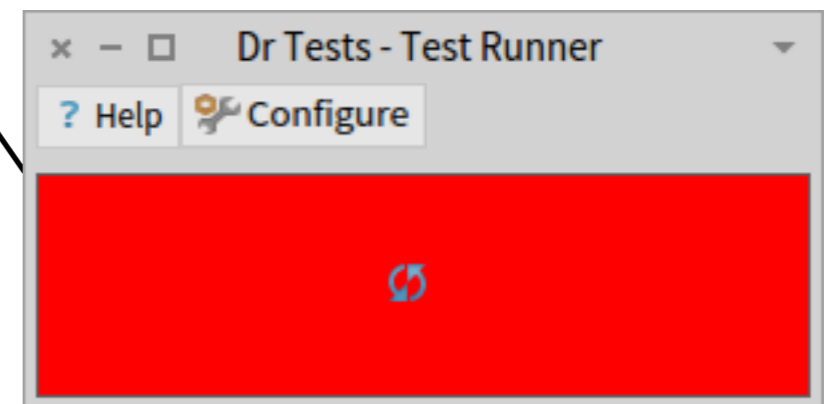
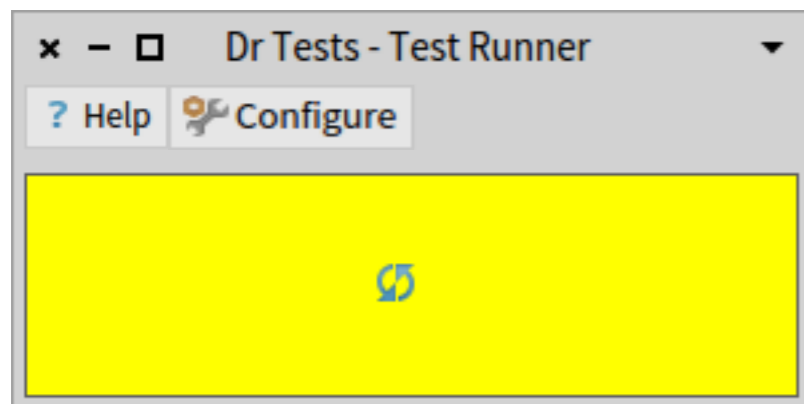
Button to re-run according to configuration

Go back to normal UI to
configure plugins, input, etc.

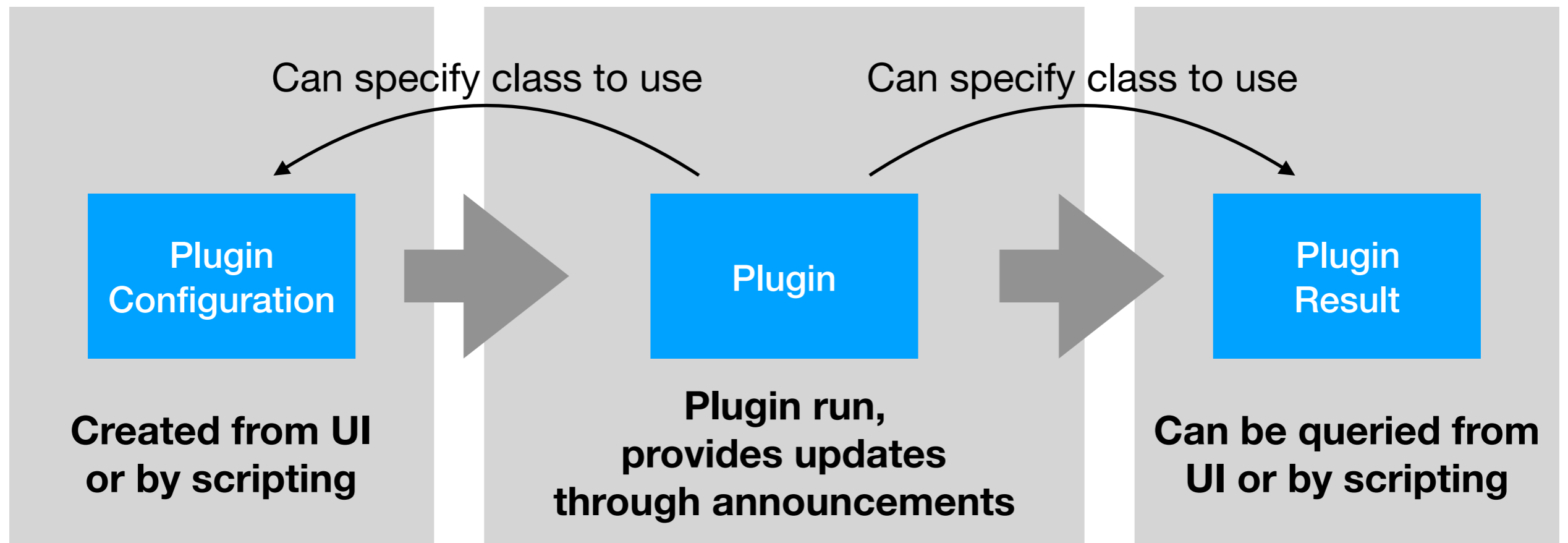
Mini Dr Tests



Button's color gives visual feedback about last result of plugin



Dr Tests model



Conclusion

Vision

→ **We want a testing ecosystem able to evolve**

★ Enhanced SUnit

★ Uniform API for SUnit clients

★ Plugin-based testing UI = **Dr Tests**

★ More tools to handle tests = **Dr Tests plugins**

What's next?



Dayne is working on Dr Tests and SUnit enhancements

Clothilde *will* work on Smart Tests soon



We can test it!



You can help the effort!



@juldelplanque 

We are interested in your inputs and contributions!



julindelpunque 



[juliendelpunque/DrTests](#)