

A Test Framework for Indico

Pedro Ferreira

CERN IT-UDS-AVC

22nd June 2010





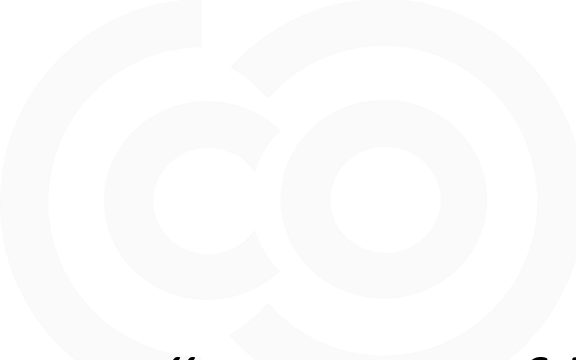
Contents

- Motivation
- Software Testing
- Indico Test Framework
- Conclusions

A normal day at work

- Pedro implements new feature X;
- Pedro makes N manual tests and they all succeed - *"it seems to work"*;
- Feature eventually gets deployed;
- Pedro finds out that feature X broke feature F, through a user report;
- Pedro fixes feature X so that it doesn't break feature F;
- Pedro does $2 * N$ manual tests and after some retries/changes, they all succeed - *"but now it is OK"*;
- The fix for feature X now breaks feature Y (that the $2 * N$ tests didn't cover);





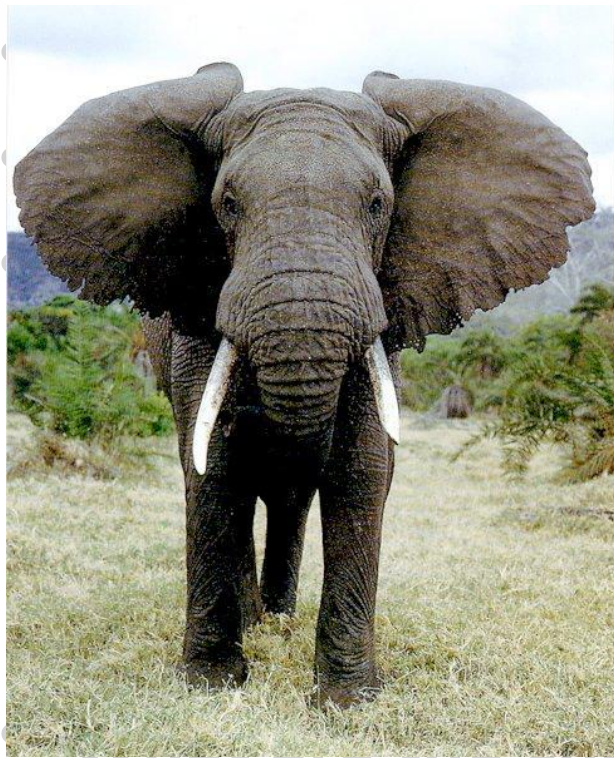
"Beware of bugs in the above code; I have only proved it correct, not tried it."

(Donald Knuth)


"Given enough eyeballs, all bugs are shallow."

(Eric S. Raymond)

Another common situation



What are the changes that need to be made to work correctly after the refactoring?



"If you want to refactor, the essential precondition is having solid tests."

(Martin Fowler)



Motivation

Software Testing

Indico Test Framework

Conclusions





Software Testing

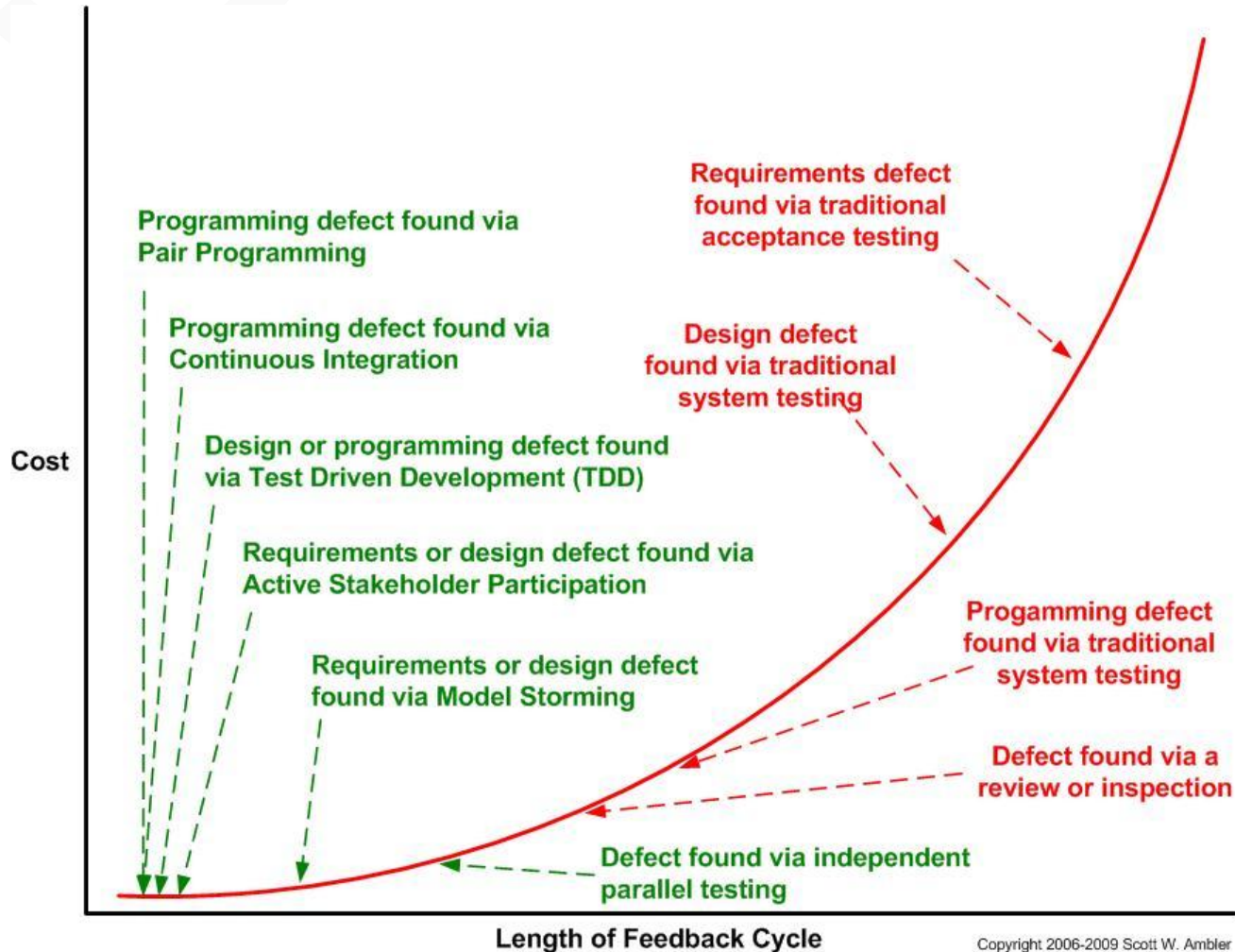
*"Software testing is an **investigation** conducted to provide stakeholders with information about the **quality** of the product or service under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the **risks** at implementation of the software."*

(Wikipedia)

Why should we test regularly?

- We want to find software bugs before they cause harm;
 - Users don't like bugs, bad for product image;
 - Bugs are costly (someone has to fix them);
- We want to be sure that modifications of the source code won't break already existing features;
 - Programmers should feel confident – always careful;
 - Tests reduce the costs of refactoring;
- Everyone else is doing it...

A word about cost...

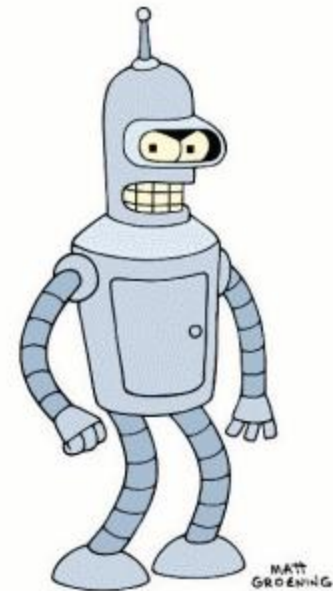


Copyright 2006-2009 Scott W. Ambler



Why should we automate testing?

- Computers are much better than human at performing repetitive tasks:
 - Computers don't (normally) fail at executing tasks;
 - Computers are much faster - we can afford doing it often;
- Computers can simulate some aspects of human behavior;
 - They're very bad at it, but they're seem to be good enough for us;
- It's much cheaper spending some time on a test infrastructure, than hiring someone to do manual tests;



What can we test?

- Code:
 - Code quality and "smells" (static code analysis)
 - Components (unit testing);
 - Code coverage;
 - ...
- User Interfaces:
 - Functional testing;
 - Usability testing;
 - ...
- System as a whole - Integration tests;
- Interfaces with other systems - System Integration tests;
- Performance;
- Security;

Tests don't solve everything

"Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence."

(Edsger Dijkstra)

"A fool with a tool is still a fool."

(?)



Motivation

Software Testing

Indico Test Framework

Conclusions



The Indico Test Framework

- An old idea, some previous attempts made;
 - Absence of an "official" test policy;
 - Pressure of deadlines, lack of manpower;
- September 2009 - the project starts:
 - Jeremy Nguyen Xuan, EPFL student
 - now CERN Fellow BE-CO;
 - >5 month work;
 - A first "market research";
 - Designing the test infrastructure;
 - Implementing the framework and associated systems;
 - Code name **IndiCop**;





What we needed

- Unit tests - **essential**;
 - For both Python and JavaScript;
- Source analysis - enforcing code standards and preventing common mistakes;
 - Again, both Python and JS;
- Code coverage - not essential, but still useful;
- Functional tests - for user interfaces;

- Integration server - for continuous integration;
 - We should integrate often;
 - All the tests would be performed and we'd be notified if a build failed;



What we found

Unit tests

UnitTest

py.test

Nosetests

pinocchio

Doctest

TestOOB

Functional tests

Twill

Selenium

Windmill

Coverage

coverage.py

figleaf

Source Analysis

PyChecker

Pylint

JS tests

js-test-driver (unit tests)

js-lint (source analysis)

NoseJS (run unit tests)

from Nguyen-Xuan, "Complete Analysis/Design/Implementation of a Test Suite for the Indico Project", March 2010

Unit Testing

- Python - *Nosetests*
 - Widely adopted, compatible with Python *stdlib*;
 - Python *stdlib* unittest was too basic;

```
python setup.py test --unit
```

- JavaScript - Google *js-test-driver*
 - Allows JS unit tests to be run on real browsers;
 - Still under development, but already very powerful;

```
python setup.py test --jsunit
```



Unit Testing

- OK/FAIL/ERROR
- Python - fake, clean database used - automatically launched by framework;
- Should all pass before a patch is sent for reviewing;
- Implementation of new features will require writing tests for that feature;

```
import md5, random, time
#50 Two consecutive set() operations over the same attribute ... ok
#51 Default value for non-existing attribute ... ok
#52 Getting the value of an existing attribute ... ok
#53 Return a DummyContext in case the attribute is not defined ... ok
#54 Thread safety ... ok
#55 getdefault() shouldn't fail, but return a DummyContext instead ... ok
#56 methods called on get() should do nothing ... ok
#57 __getitem__ should do nothing ... ok
#58 __setitem__ should be ignored ... ok
#59 get() shouldn't fail, but return a DummyContext instead ... ok
#60 has() will return false ... ok
#61 set() shouldn't fail, but return a DummyContext instead ... ok
#31 Default fossil ... ok
#32 Fossilize a dictionary of objects ... ok
#33 Fossilize list of fossilizable classes ... ok
#34 Method names should map to attribute names ... ok
#35 Serializing primitive objects such as strings, integers and floats ... ok
#36 'produce' attribute ... ok
#37 Serializing semi-structured data such as lists and dictionaries ... ok
#38 Complex fossilization example ... ok
#39 Complex fossilization example including conversion ... ok
#40 Non-fossilizable objects should throw an exception ... ok
#41 Simple fossilization example ... ok
#42 Fossilize composite objects with conversion ... ok
#43 Dynamic (added in runtime) fossils ... ok
#44 Fossil inheritance ... ok
#45 Fossilizing using the wrong type should throw an exception ... ok
#46 Some invalid names should raise an exception ... ok
#62 Makes sure a file wich is attached to a conference gets stored in ... ok
SKIP
/home/pferreir/indico/cds-indico/indico/MaKaC/services/interface/rpc/description.py:3: DeprecationWa
import sha
#66 testCreateCSBookingObject (indico.MaKaC.plugins.Collaboration.Vidyo.tests.python.unit.collaborat
#67 testObtainVidyoClass (indico.MaKaC.plugins.Collaboration.Vidyo.tests.python.unit.collaboration_t
#68 testAdminAPIConnectivity (indico.MaKaC.plugins.Collaboration.Vidyo.tests.python.unit.common_test
#69 testUserAPIConnectivity (indico.MaKaC.plugins.Collaboration.Vidyo.tests.python.unit.common_test
#70 testCommonOptions (indico.MaKaC.plugins.Collaboration.Vidyo.tests.python.unit.options_test.TestV
.....
Ran 34 tests in 6.064s

OK (SKIP=1)
** Python Unit Tests successful!
** report in /home/pferreir/indico/cds-indico/indico/tests/report/UnitTestRunner.txt
-- Stopping test DB
```

Unit Tests – Some Questions

- Existing code - what to cover?
 - Writing tests for code that will be refactored would be a waste of resources;
- How much, in general, should we cover?
 - The cost of maintaining tests shouldn't be very high;
- Granularity - How precise?
 - Feature-oriented testing (?)

Code Coverage

- Are we executing every single line of code?
- Performed during unit tests;
- figleaf/js-test-driver provide coverage reports;

```
python setup.py test --unit --coverage
```

```
python setup.py test --jsunit --coverage
```

```

8. ## modify it under the terms of the GNU General Public License as
9. ## published by the Free Software Foundation; either version 2 of the
10. ## License, or (at your option) any later version.
11. ##
12. ## CDS Indico is distributed in the hope that it will be useful, but
13. ## WITHOUT ANY WARRANTY; without even the implied warranty of
14. ## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
15. ## General Public License for more details.
16. ##
17. ## You should have received a copy of the GNU General Public License
18. ## along with CDS Indico; if not, write to the Free Software Foundation, Inc.,
19. ## 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.
20.
21. import MaKaC.webinterface.urlHandlers as urlHandlers
22. from MaKaC import conference
23. from MaKaC.i18n import _
24.
25. class NavigationEntry:
26.     _title = ""
27.     _url = None
28.     _parent = None
29.
30.     def getTitle(cls):
31.         return cls._title
32.     getTitle = classmethod( getTitle)
33.
34.     def getURL(cls, *arg):
35.         if cls._url is not None:
36.             return cls._url.getURL(*arg)
37.         return cls._url
38.     getURL = classmethod( getURL)
39.
40.     def getParent(cls, *arg):
41.         if cls._parent:
42.             return cls._parent()
43.         return None
44.     getParent = classmethod( getParent)
45.
46.
47. class NEConferenceProgramme( NavigationEntry ):
48.     _url = urlHandlers.UHConferenceProgram
49.     _title = "Scientific Programme"
50.
51. class NEConferenceCFA( NavigationEntry ):

```

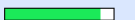


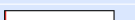

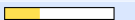






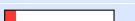





LCOV - code coverage report

Current view: directory

Test: builtConf.conf-coverage.dat

Date: 2010-06-14

	Found	Hit	Coverage
Lines:	11108	1687	15.2 %
Functions:	0	0	-

Directory	Line Coverage ↕	Functions ↕
/home/pferreir/indico/cds-indico/indico/tests/javascript/unit	 87.5 %	7 / 8
/home/pferreir/indico/cds-indico/indico/tests/javascript/unit/tests/Timetable	 100.0 %	17 / 17
indico/Admin	 4.2 %	5 / 118
indico/Collaboration	 7.4 %	50 / 675
indico/Common	 0.9 %	3 / 313
indico/Core	 12.0 %	75 / 624
indico/Core/Dialogs	 5.1 %	17 / 333
indico/Core/Interaction	 31.7 %	13 / 41
indico/Core/Widgets	 5.4 %	76 / 1417
indico/Display	 4.8 %	1 / 21
indico/Legacy	 0.4 %	4 / 1013
indico/Management	 2.4 %	34 / 1445
indico/MaterialEditor	 2.3 %	12 / 511
indico/Timetable	 25.1 %	457 / 1824
presentation/Core	 45.6 %	349 / 766
presentation/Data	 26.5 %	247 / 931
presentation/Ui	 40.9 %	184 / 450
presentation/Ui/Draw	 13.2 %	31 / 234
presentation/Ui/Extensions	10.4 %	14 / 135
presentation/Ui/Widgets	27.6 %	92 / 333

Generated by: LCOV version 1.7



Source Code Analysis

- Pylint and JSLint
- Plain text and HTML (Pylint only) reports provided;
- Different kinds of warnings/errors:
 - PEP8-compliance (plus our own extensions);
 - Unused/undeclared variables;
 - Long lines;
 - Code "smells" (i.e. methods that could be static);
 - ...
- Results have to be taken with a grain of salt;
 - False positives;
 - Situations where theoretically "bad code" is needed;
- It will be a long time till the whole Indico code base is at a satisfactory state;

Functional Testing

- Testing user interfaces is a tedious job;
- Nothing can test a user interface better than a human;
- We now rely a lot on JavaScript
 - The behavior has to be checked across different browsers;
- Solution:
 - Selenium RC - automated tests on different browsers;
 - Selenium Grid - tests can be processed by different machines, as grid jobs;



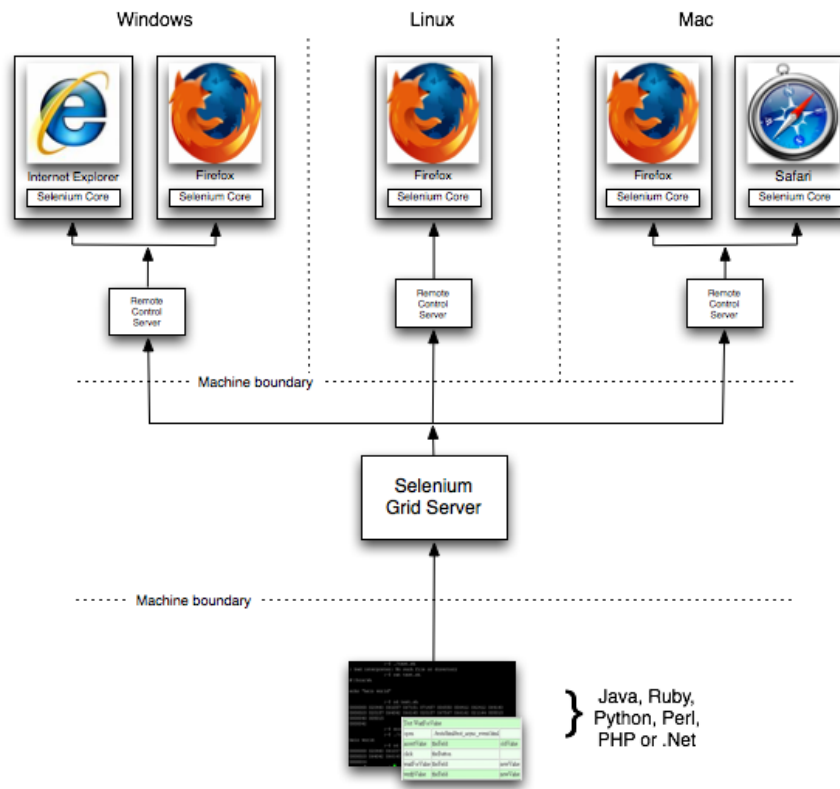
Functional Testing

- Tests are specified as sequences of actions that the user would follow;
 - Link/button clicks, waiting for a specific message to appear on the screen, page reloading, etc...
- The Selenium IDE Firefox plugin allows us to record test sequences easily;
- A test database is used (fake user created);

```
python setup.py test --functional
```

```
python setup.py test --grid
```

Functional Testing – Selenium Grid



Functional Testing- Demo

- Video [here](#)

Functional Testing - Problems

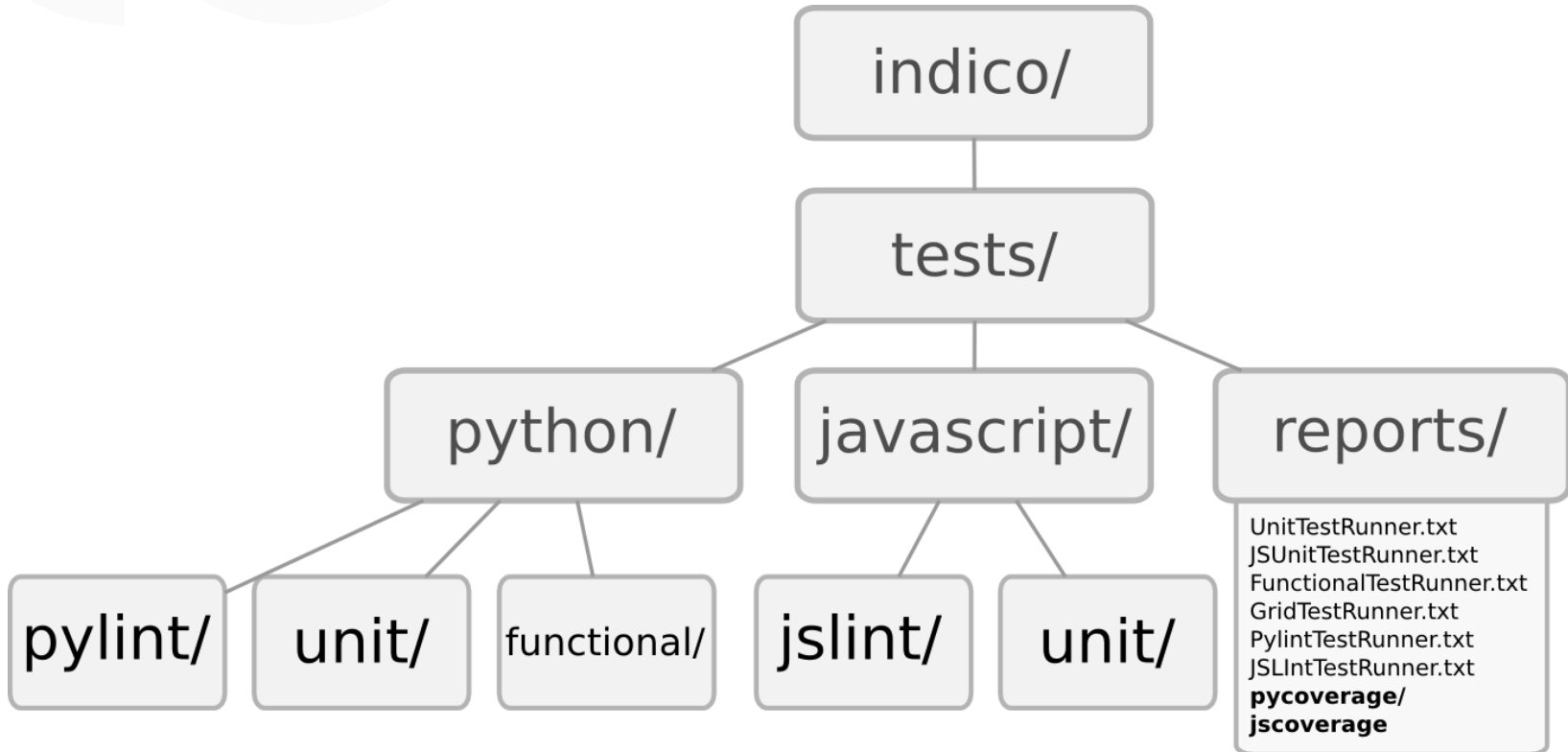
- `mod_python`
 - Creating several instances of Indico per machine is possible but hard - WSGI will make it easier;
 - For now, we have to make sure tests don't interfere with each other;
 - We need to restart apache between test runs (database switches);
- Speed - quite slow, compared to unit tests;
- AJAX can be hard to test, sometimes;
- Same problems as unit testing:
 - Do we want to cover the whole app?
 - Granularity - how much should a single test cover?
 - How much of the existing interface do we want to cover?

Running all the tests

```
python setup.py test
```

1. Python Unit
2. Pylint
3. JSUnit
4. JSLint
5. Functional (local)
6. Grid

Directory Structure

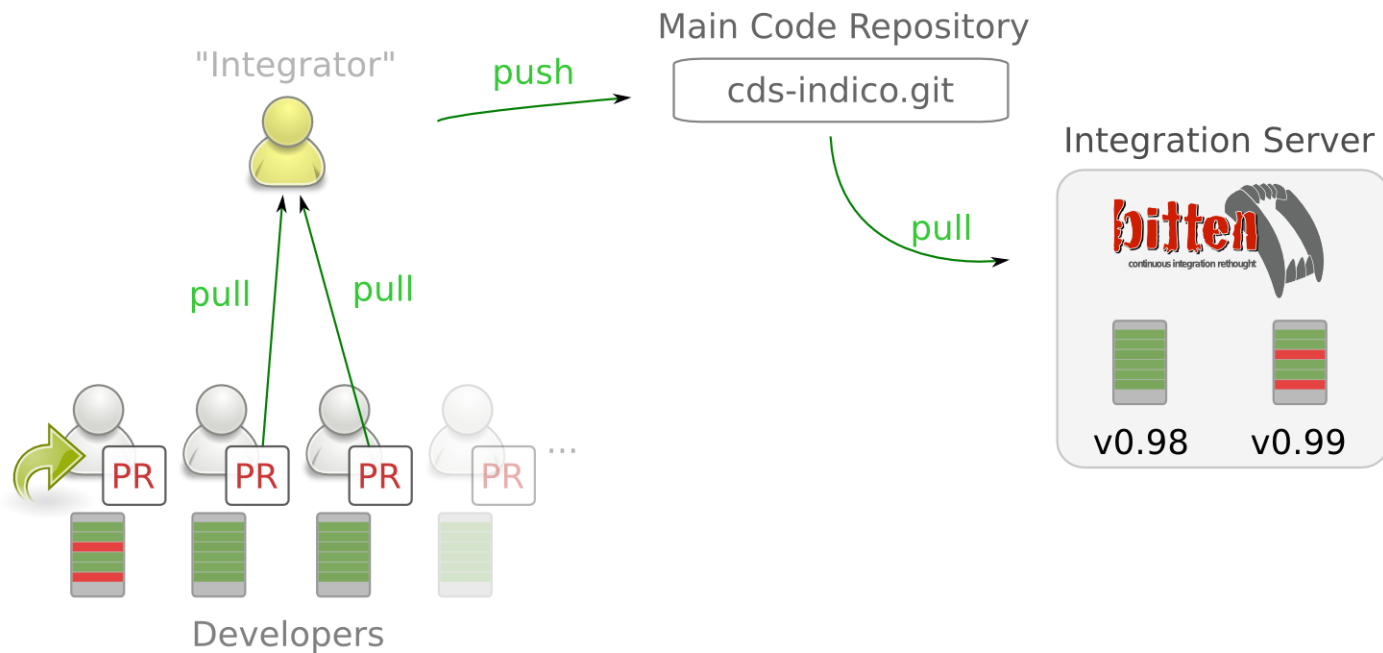


Integration Server

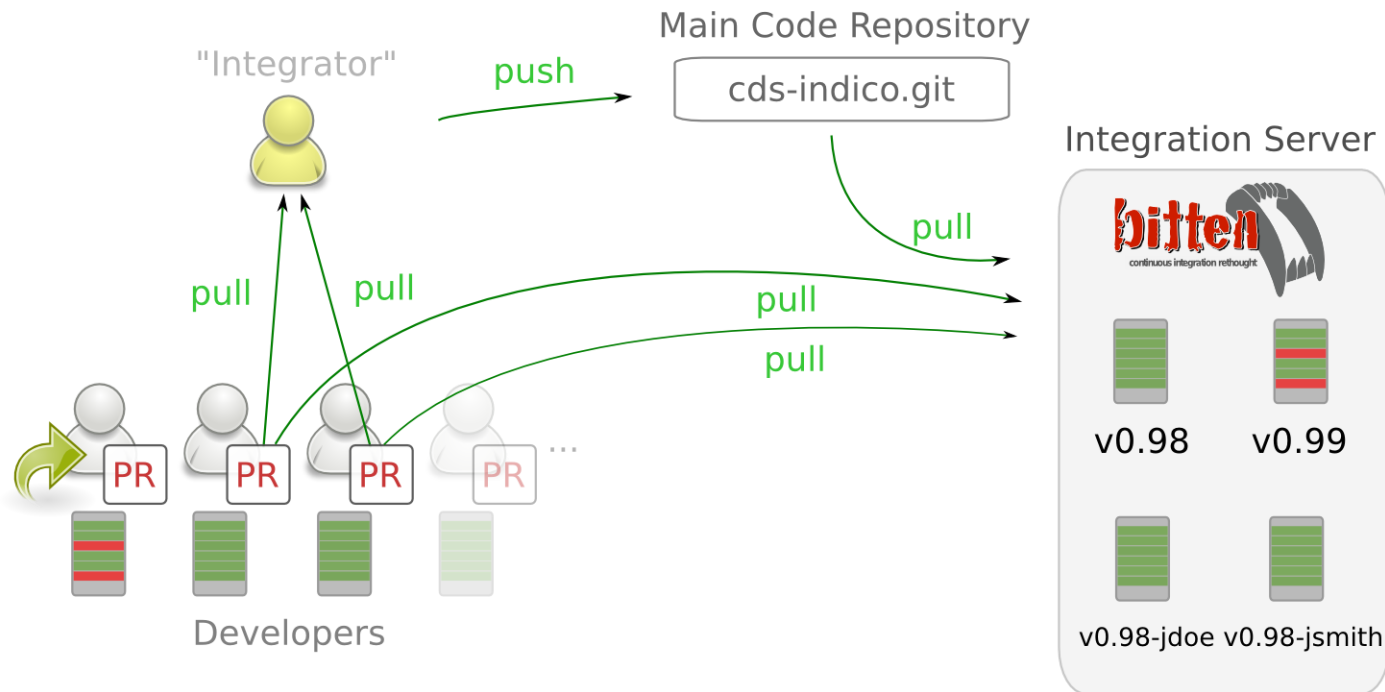
- Last step;
- We want to have tests executed each time there is a change in the repository;
 - Continuous integration;
- Bitten was chosen for this;
 - Open source;
 - Widely adopted by the Python community;
 - Based on Trac (which we already use);



Integration Server



Integration Server



Integration Server

- Each time there is a change in the Git repository, the integration server will execute several tests;
 1. Fetch Indico from the Git repo;
 2. Package and install Indico in a virtual environment (virtualenv);
 3. Configure Indico;
 4. Run unit tests (Python and JS);
 5. Run functional tests;
- Bitten [web interface](#) provides detailed feedback on the different steps;



Motivation

Software Testing

Indico Test Framework

Conclusions



The Full Picture

Ticket tracking / Wiki /
Repo browser



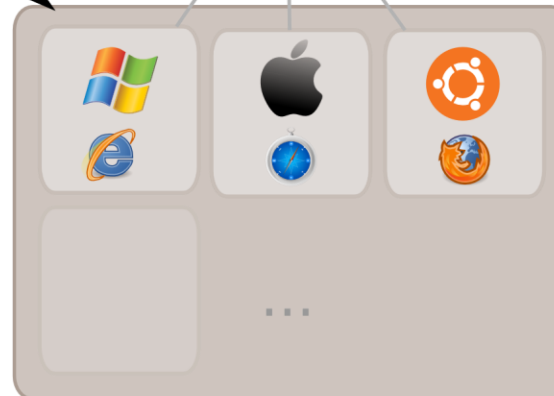
Version Control Server



Selenium Grid Hub



Test Machine "Grid"



Can act as both selenium grid nodes
and "bitten slaves"



Conclusions

- The framework is ready, but the most important is still to come - writing tests, and writing them well;
 - Test writing guidelines already defined (wiki, cheatsheet...), well-defined and easily extensible framework structure;
- This is a valuable tool that will for sure make development much less painful for us;
 - Mandatory tool for code contribution;

Acknowledgements

- Jeremy Nguyen-Xuan
 - for all the extra work he put into Bitten/Trac, and the availability for answering my questions;
- Everyone at *#trac* and *#bitten*, irc.freenode.org
 - for the help concerning the integration of Bitten with Git;
- J. Hampton
 - for the “semi-official” Bitten patch for Trac 0.12;



Questions?

