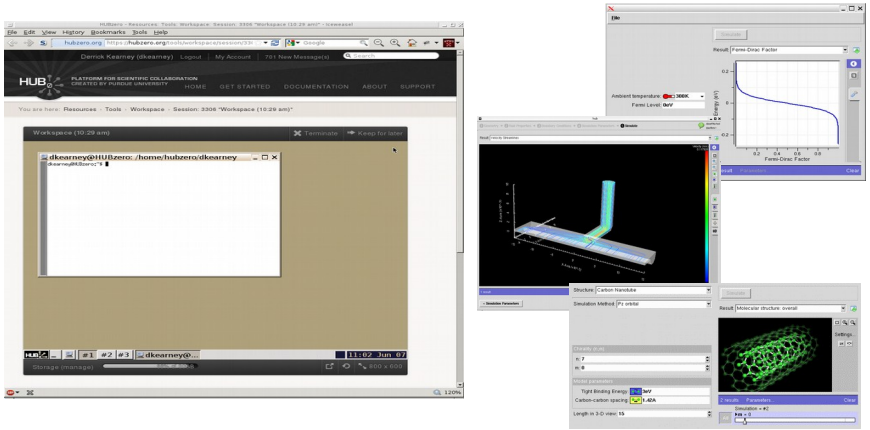
HUBcheck:
Check the hub

Master's Final Examination
April 3, 2014

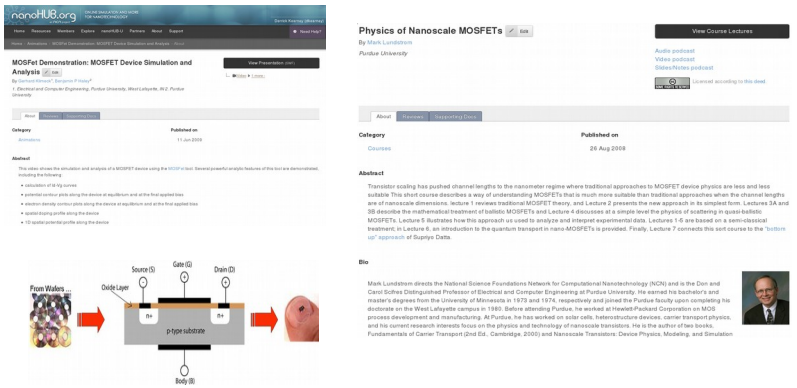
Derrick Kearney
Purdue University

What is a hub?

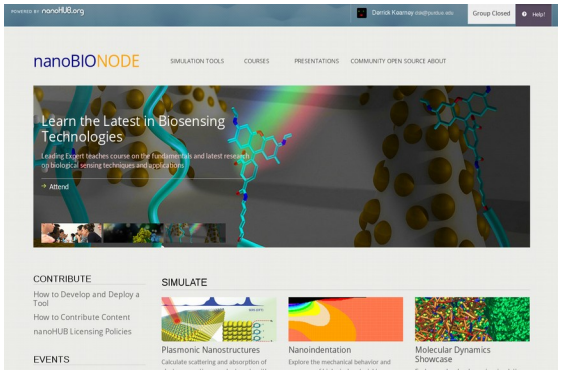
Simulation Tools



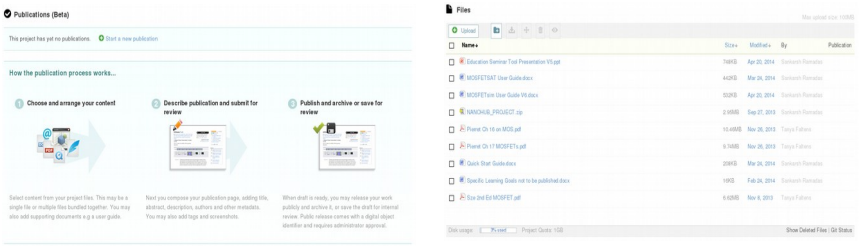
Content Sharing



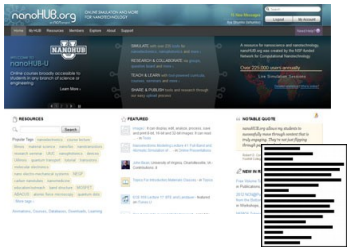
Collaboration



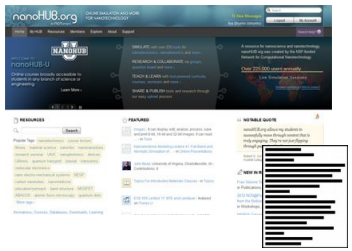
Data Management

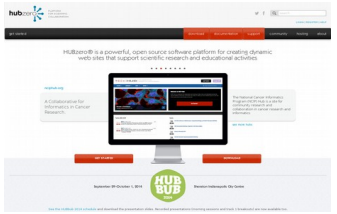


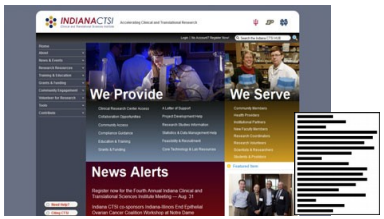
HUBzero History

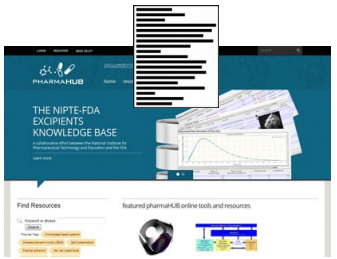


HUBzero History





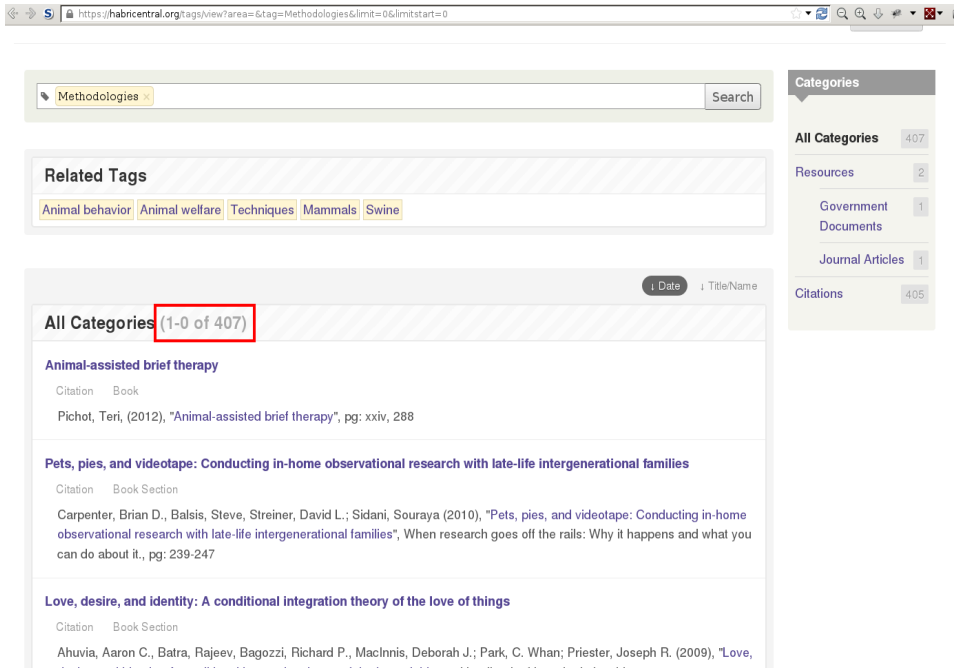


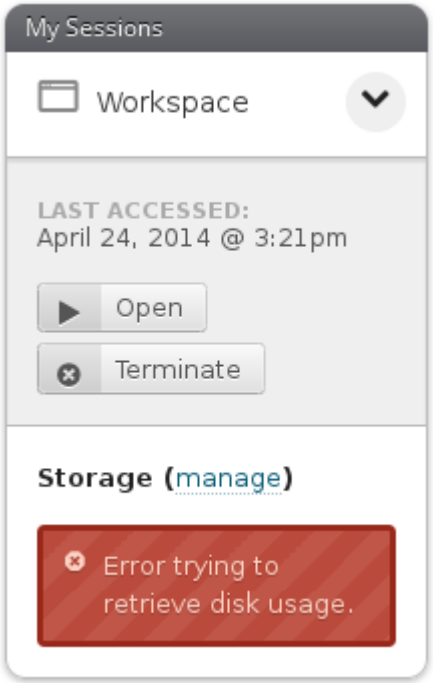


HUBzero History

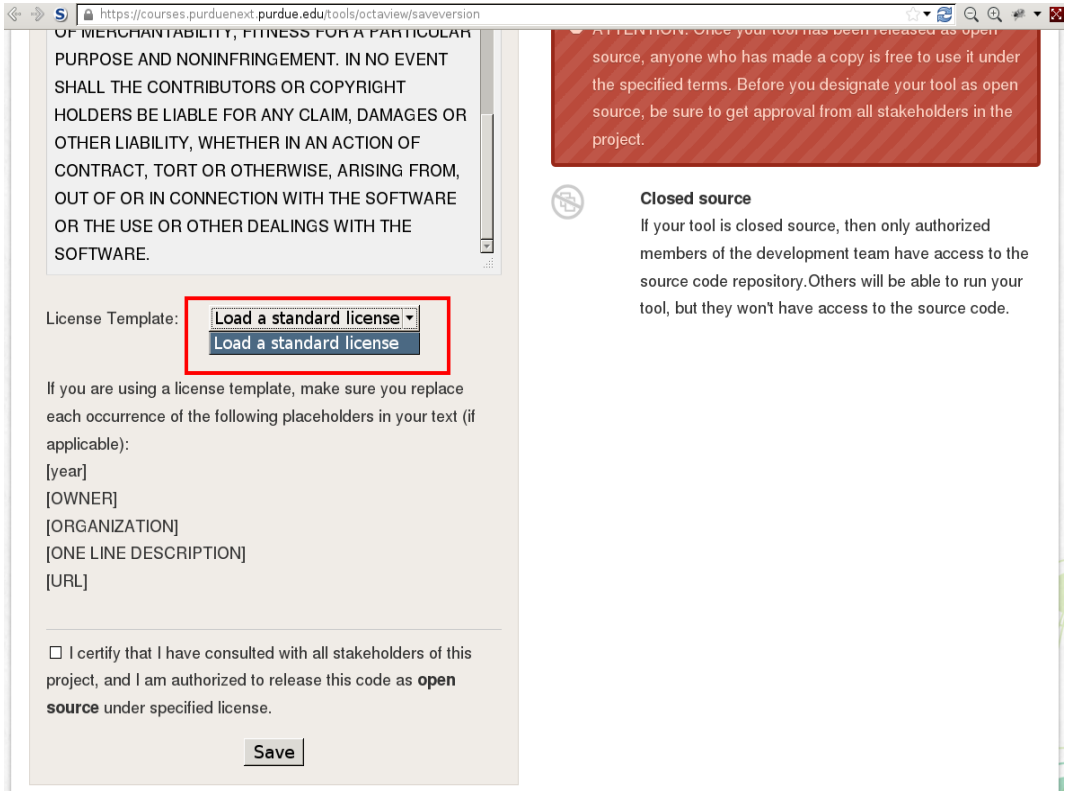
HUBzero History

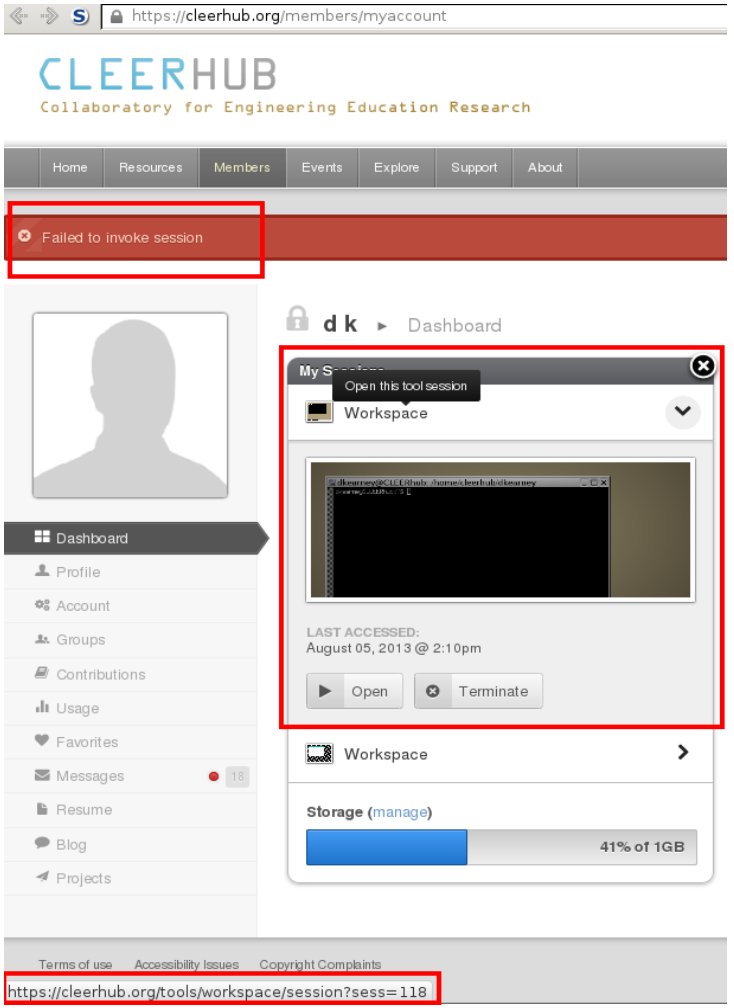
Types of Errors



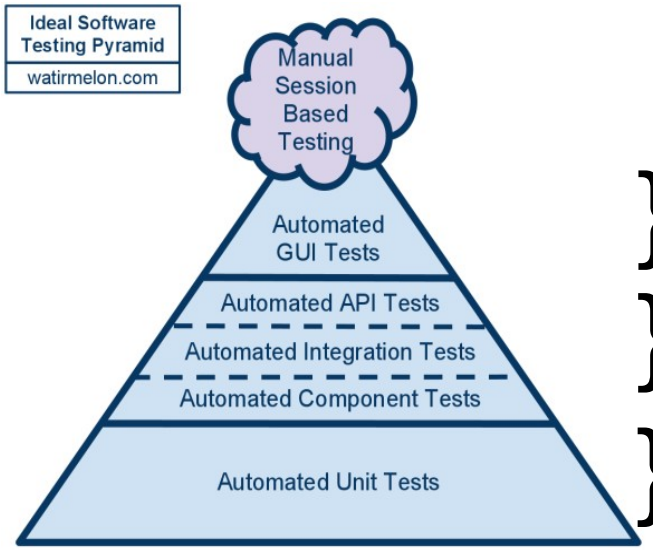


Types of Errors





Testing on the hub

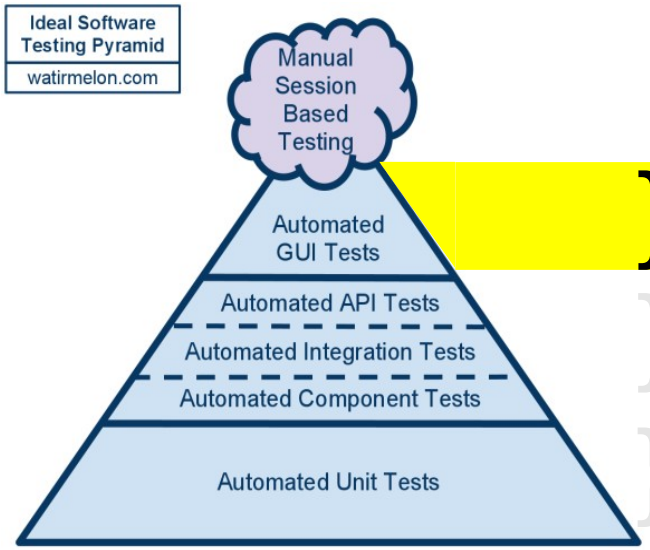


} User InterfaceTests

} Service Layer Tests

} Unit Tests

Testing on the hub

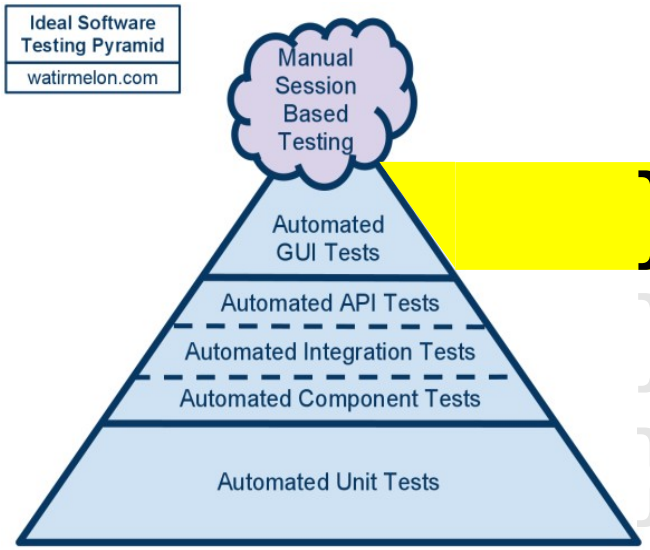


} User InterfaceTests

} Service Layer Tests

} Unit Tests

Testing on the hub



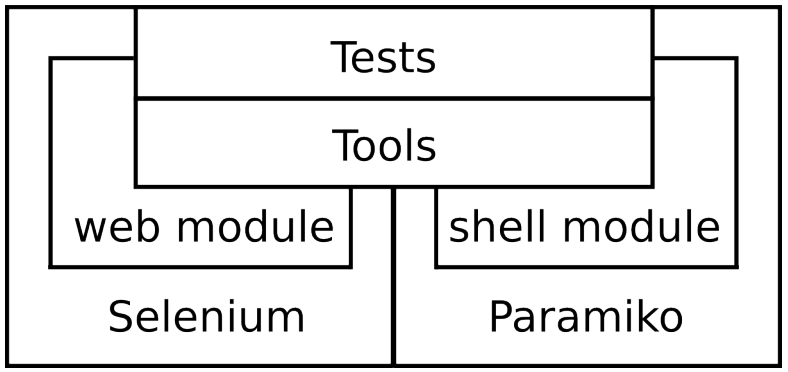
} User Interface Tests

} Service Layer Tests

} Unit Tests

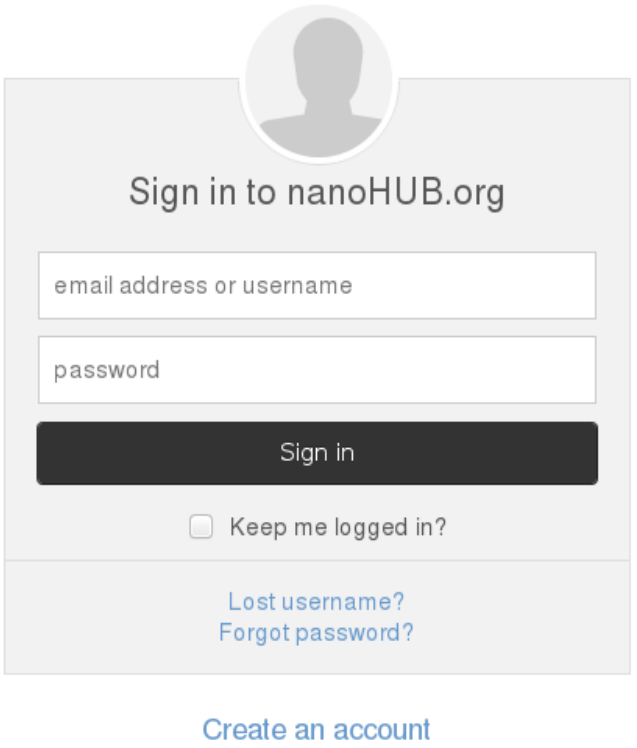
\+ SSH

HUBcheck is...

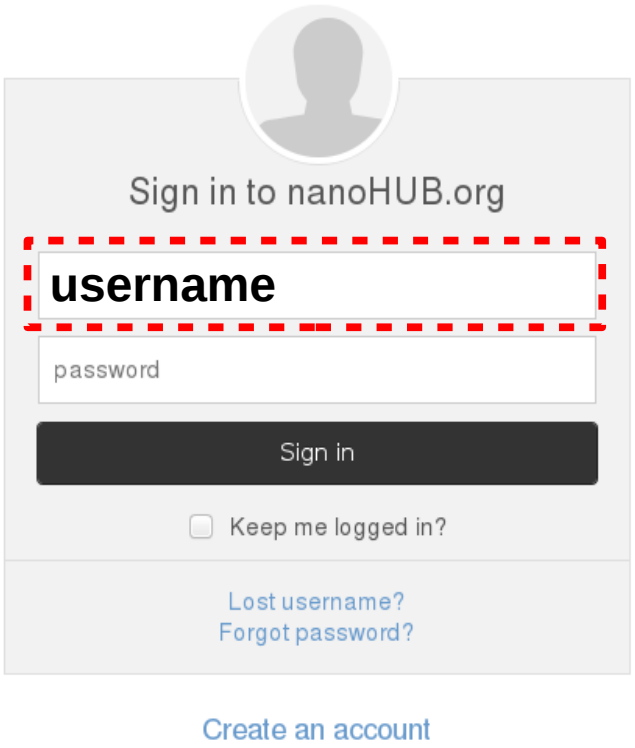


Web Browser Automation + Shell Automation

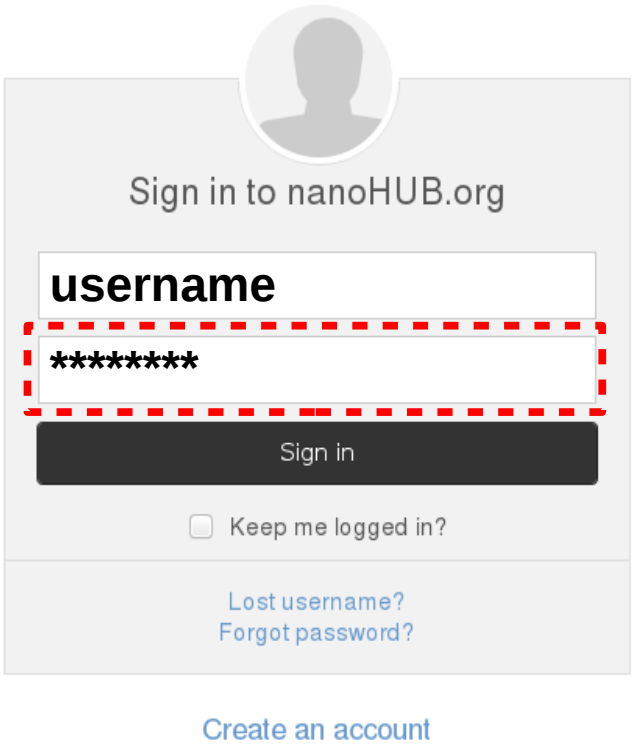
Accessing the hub website



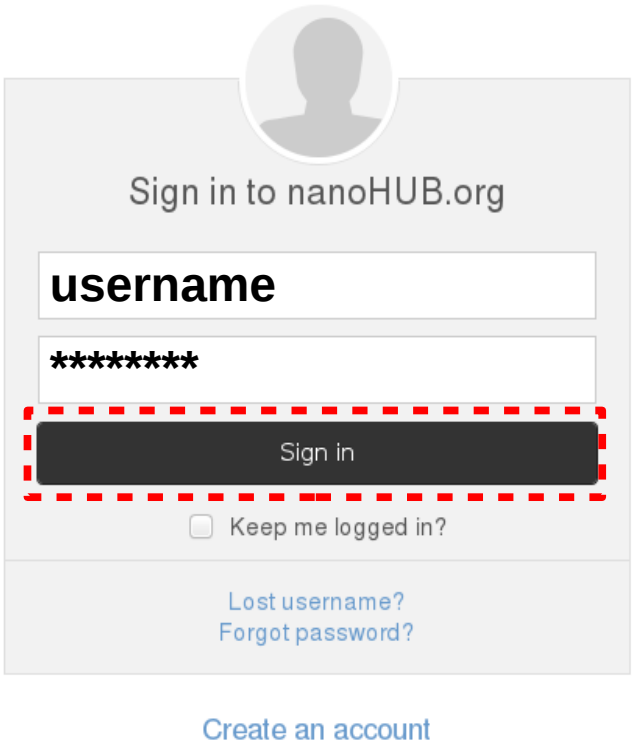
Accessing the hub website



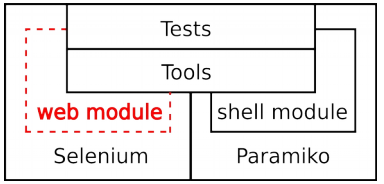
Accessing the hub website



Accessing the hub website

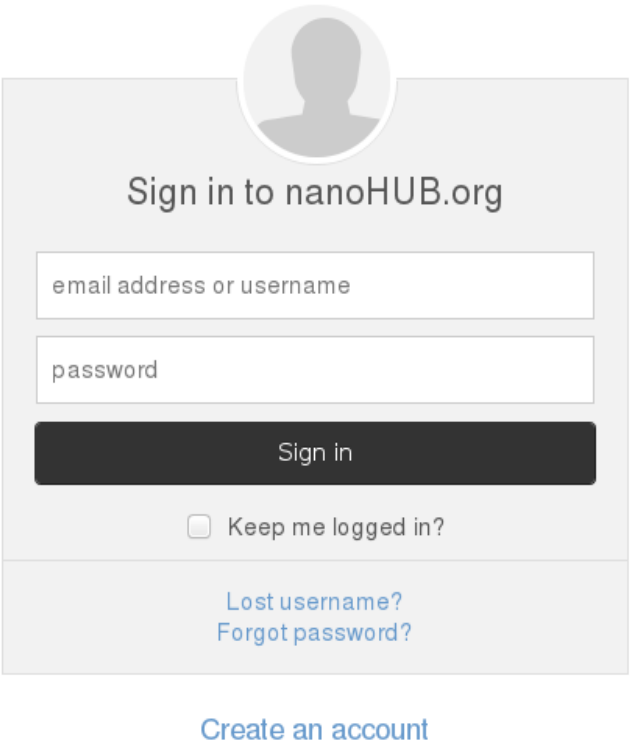


Web Automation

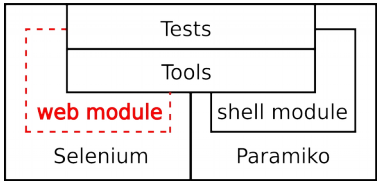


HUBcheck Page Object

```
class LoginPage(BasePageWidget):
    def __init__(self, locator):
        ...
        self.username = Text(self,...)
        self.password  = Text(self,...)
        self.submit      = Button(self,...)
        ...

    def login_as(self,username,password):
        self.username.value = username
        self.password.value = password
        self.submit.click()
```

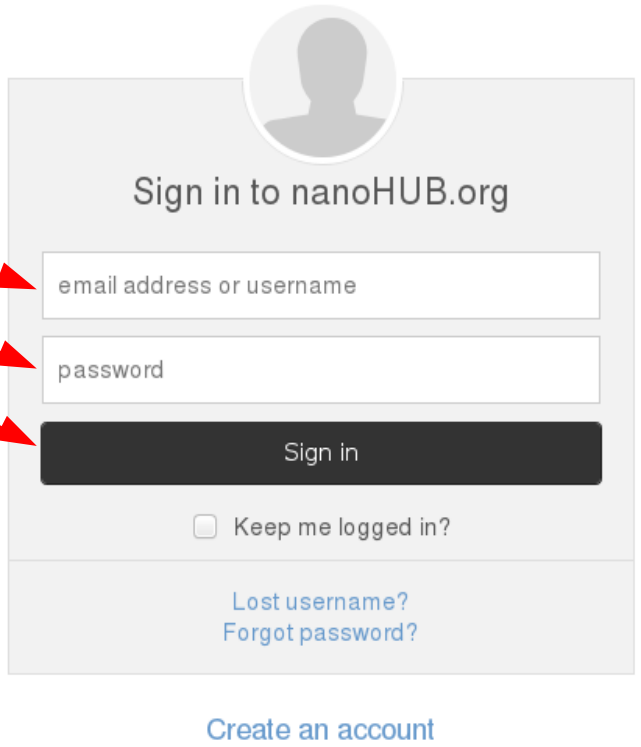


Web Automation

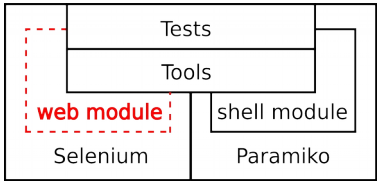


HUBcheck Page Object

```
class LoginPage(BasePageWidget):
    def __init__(self, locator):
        ...
        self.username = Text(self,...)
        self.password  = Text(self,...)
        self.submit      = Button(self,...)
        ...

    def login_as(self,username,password):
        self.username.value = username
        self.password.value = password
        self.submit.click()
```

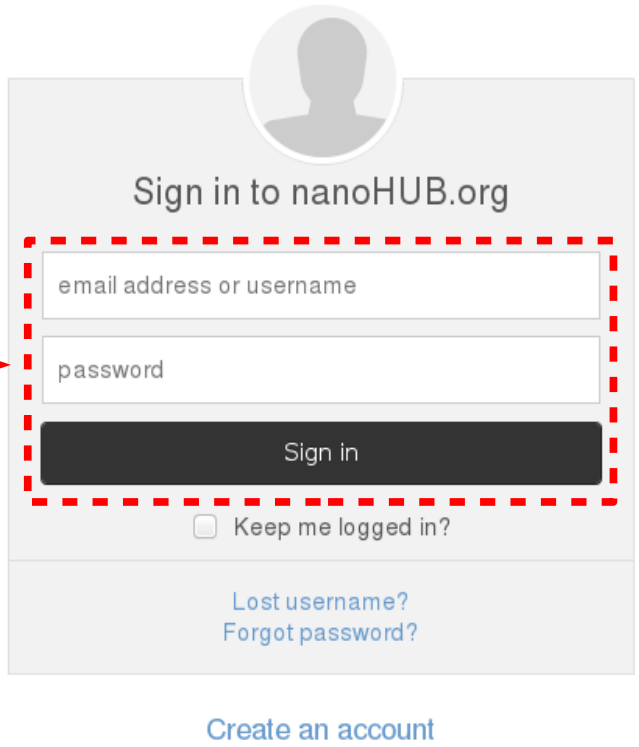


Web Automation

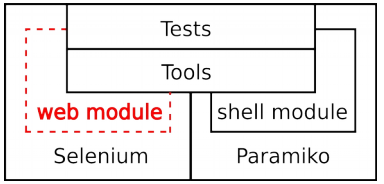


HUBcheck Page Object

```
class LoginPage(BasePageWidget):
    def __init__(self, locator):
        ...
        self.username = Text(self,...)
        self.password  = Text(self,...)
        self.submit      = Button(self,...)
        ...

    def login_as(self,username,password):
        self.username.value = username
        self.password.value = password
        self.submit.click()
```

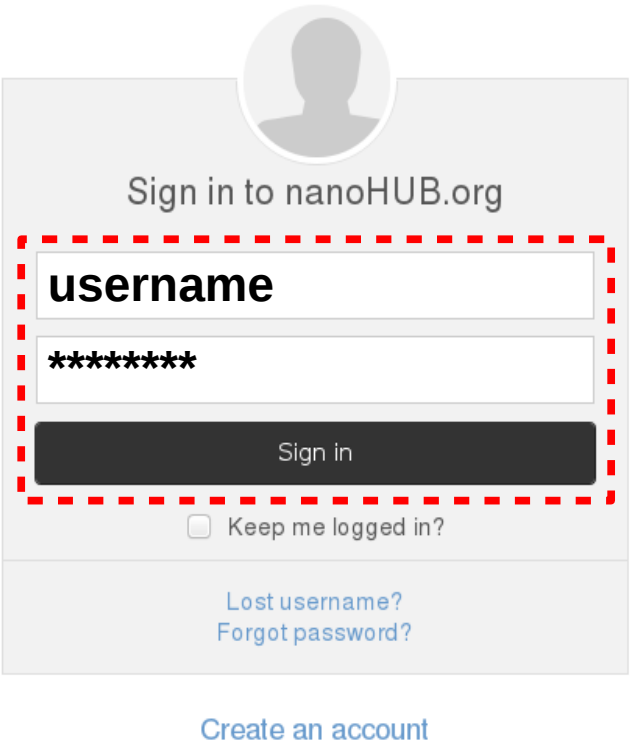


Web Automation



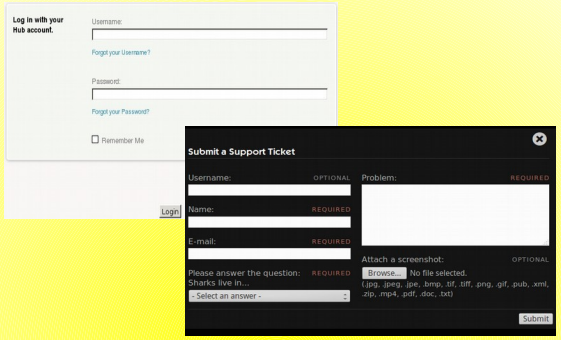
Python Script

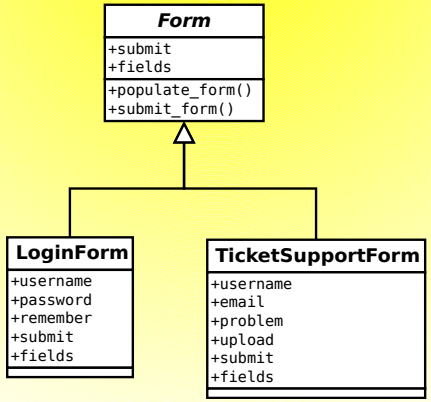
```
import hubcheck

username = 'username'
password = 'password'

hc = hubcheck.Hubcheck(hostname='nanohub.org',
                       locators='nanohub',
                       browsertype='Firefox')

# setup a web browser
hc.browser.get('https://nanohub.org')

# login to the website
po = hc.catalog.load_pageobject('LoginPage')
po.goto_page()
po.login_as(username,password)
```



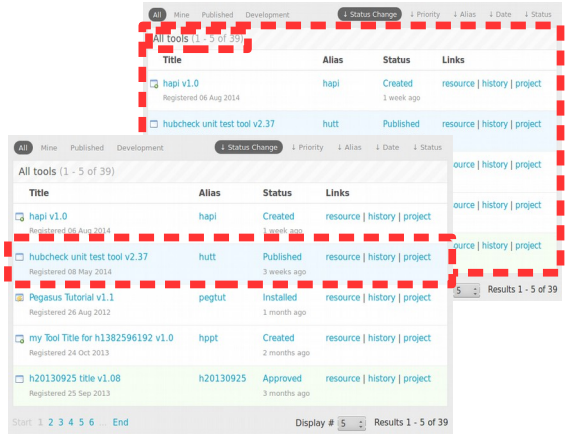
Page Object Based Design Patterns

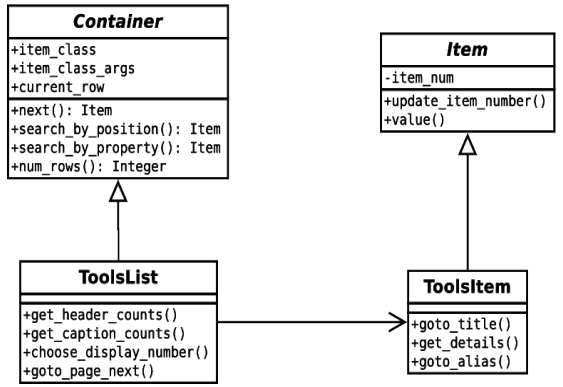
WebForm Pattern



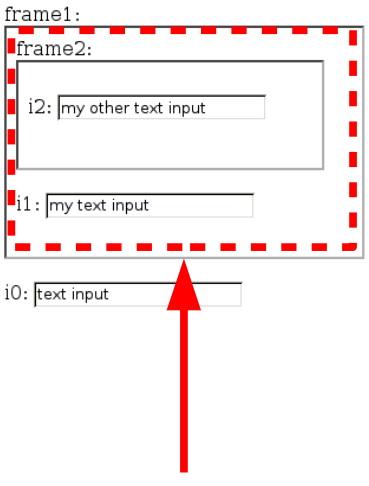


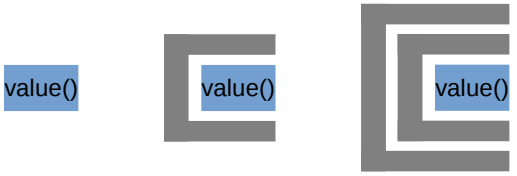
ItemList Pattern





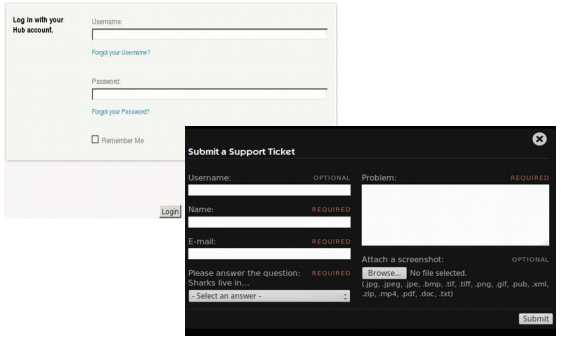
IframeWrap Pattern

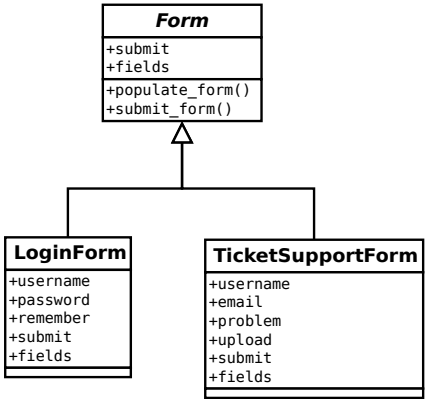




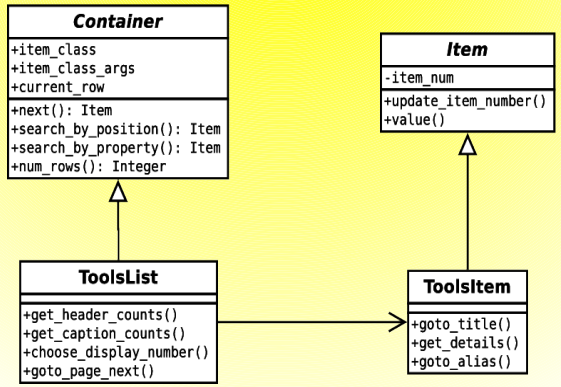
Page Object Based Design Patterns

WebForm Pattern

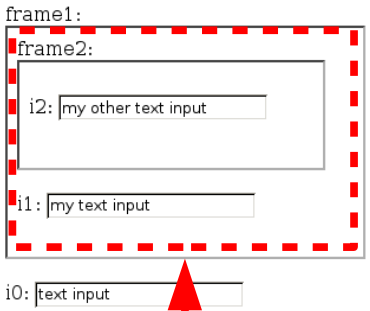




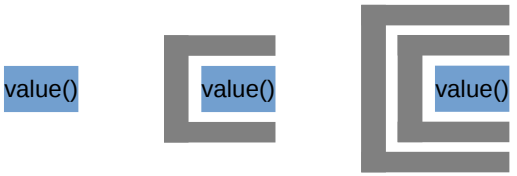
ItemList Pattern



IframeWrap Pattern

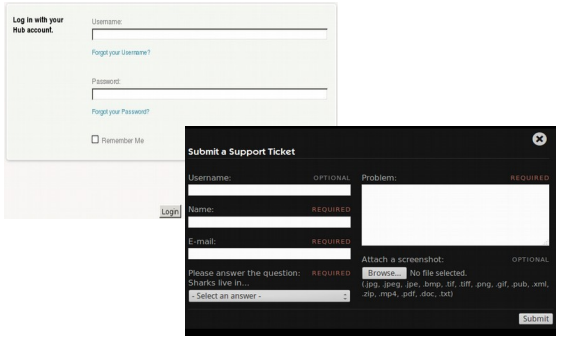


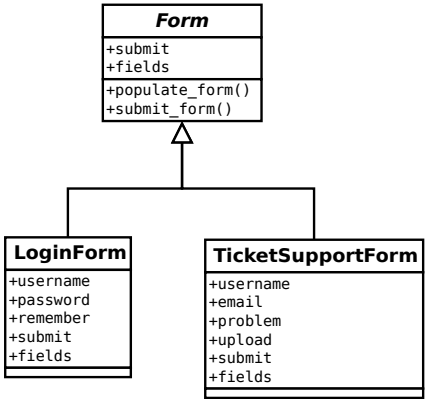
Frame1 Context



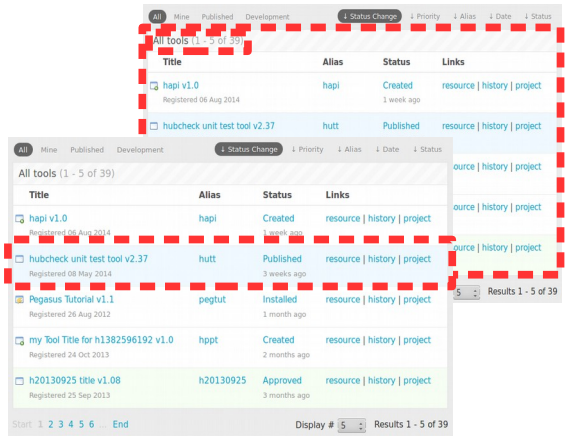
Page Object Based Design Patterns

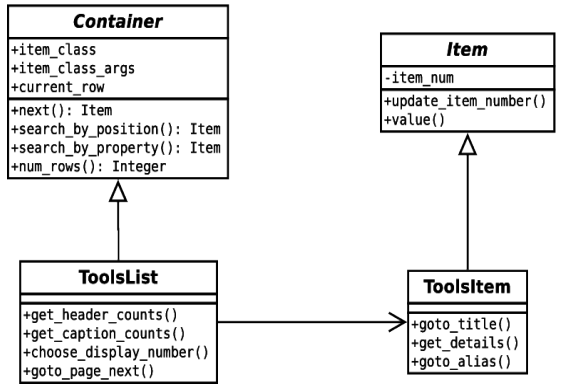
WebForm Pattern



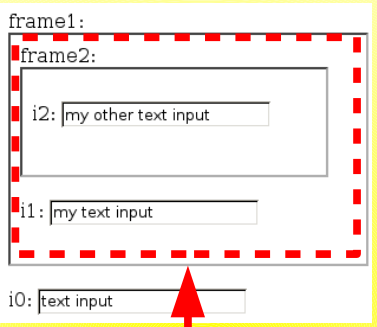


ItemList Pattern

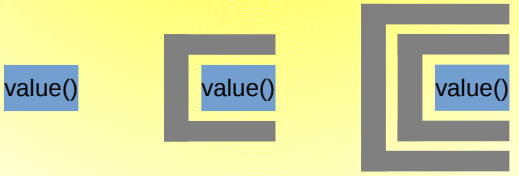




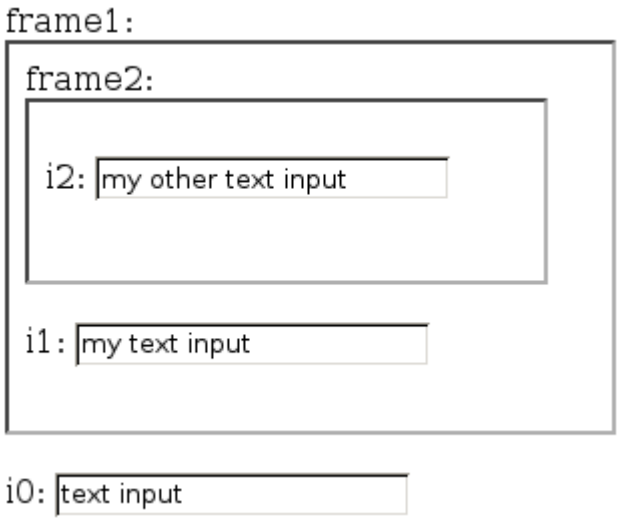
IframeWrap Pattern



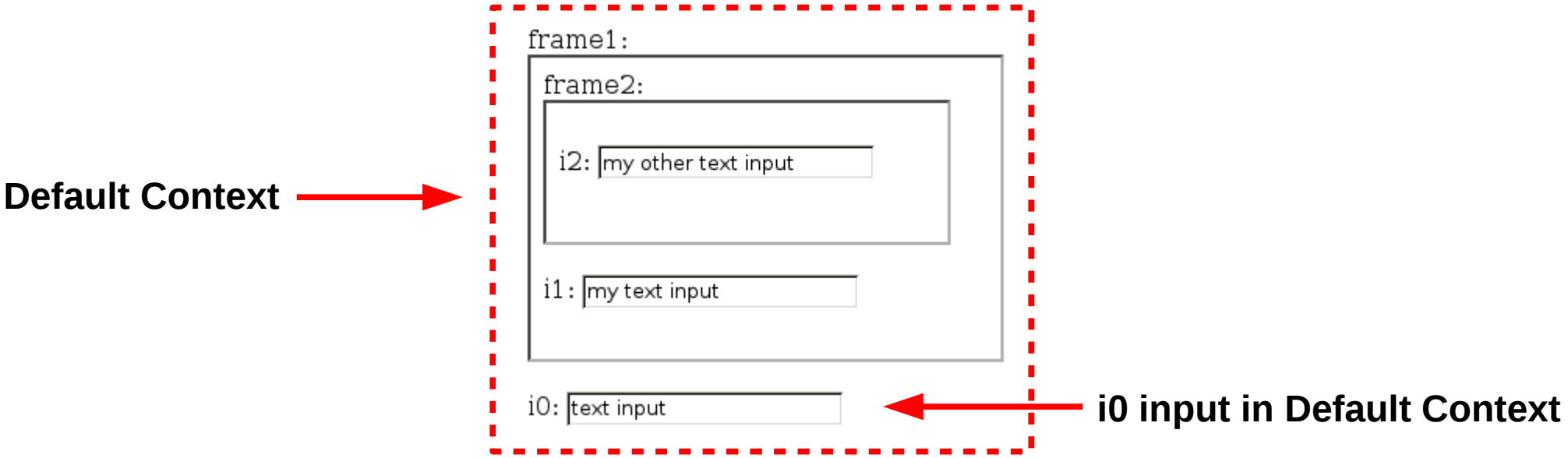
Frame1 Context



How do Iframes work

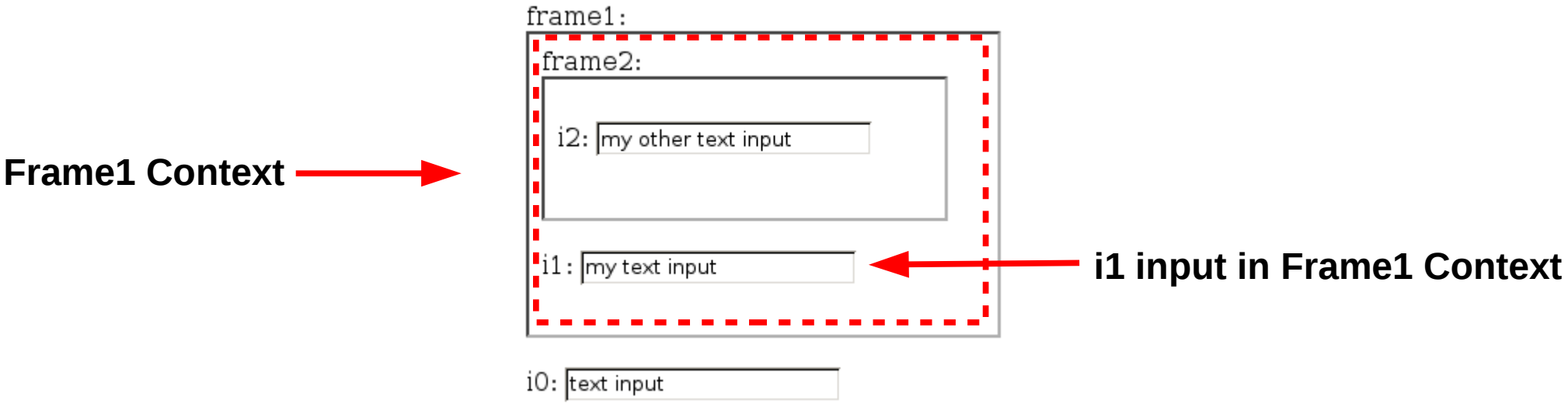


How do Iframes work



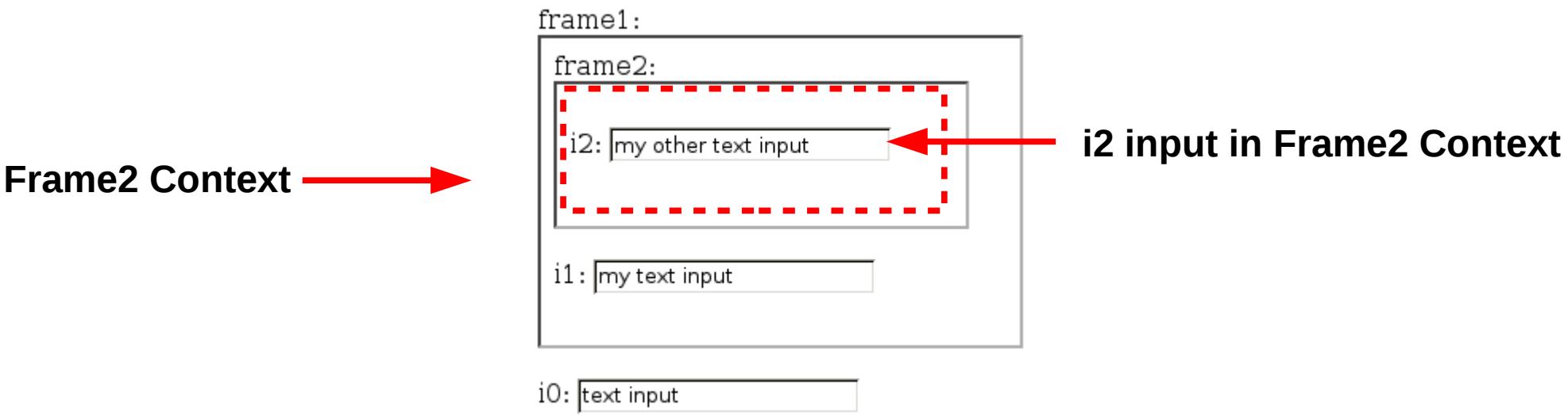

```
<html>
   <body>
      <label for="frame1">frame1: </label>
      <iframe id="frame1" src="inner_page.html"></iframe>
      <br/><br/>
      <label for="i0">i0: </label>
      <input type="text" id="i0" value="text input"></input>
   </body>
</html>
```

How do Iframes work




```
<html>
   <body>
      <label for="frame2">frame2: </label>
      <iframe id="frame2" src="another_page.html"></iframe>
      <br/><br/>
      <label for="i1">i1: </label>
      <input type="text" id="i1" value="my text input"></input>
   </body>
</html>
```

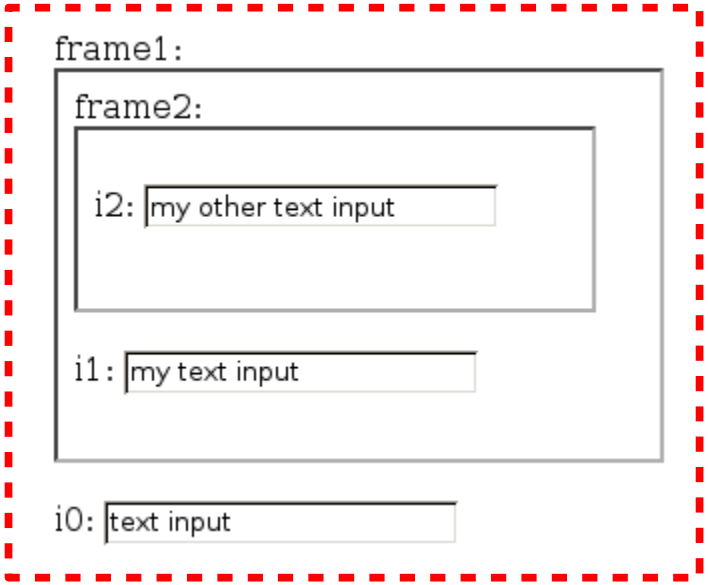
How do Iframes work




```
<html>
   <body>
      <label for="i2">i2: </label>
      <input type="text" id="i2" value="my other text input"></input>
   </body>
</html>
```

Page Object for i0

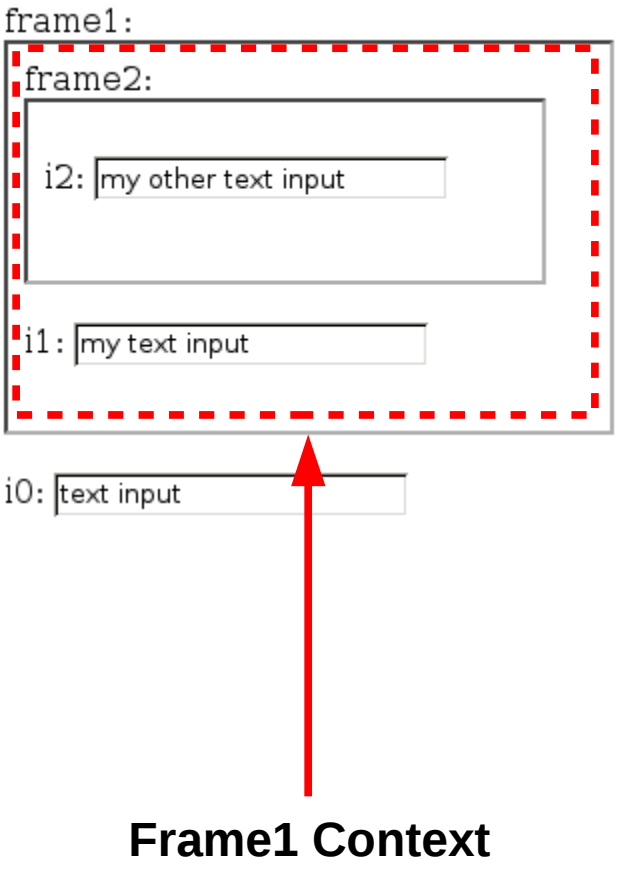
```
class Text(BasePageWidget):
    def __init__(self, locator):
        ...

    # getter
    def value(self):
        e = self.find_element(self.locator)
        return e.get_attribute('value')

    # setter
    def value(self, text):
        e = self.find_element(self.locator)
        e.clear()
        e.send_keys(text)

    def append(self, text):
        e = self.find_element(self.locator)
        e.send_keys(text)
```



Default Context

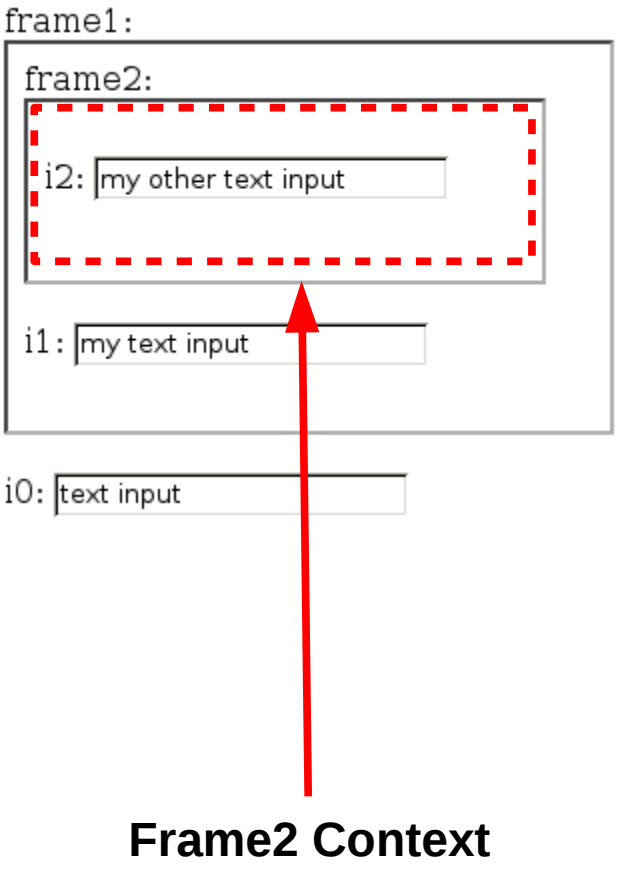
Page Object for i1

```
class Text1Frame(BasePageWidget):
    def __init__(self, locator):
        ...

    # getter
    def value(self):
        ...

    # setter
    def value(self, text):
        frame = self.find_element('#frame1')
        self._browser.switch_to_frame(frame)
        e = self.find_element(self.locator)
        e.clear()
        e.send_keys(text)
        self._browser.switch_to_default_content()

    def append(self, text):
        ...
```



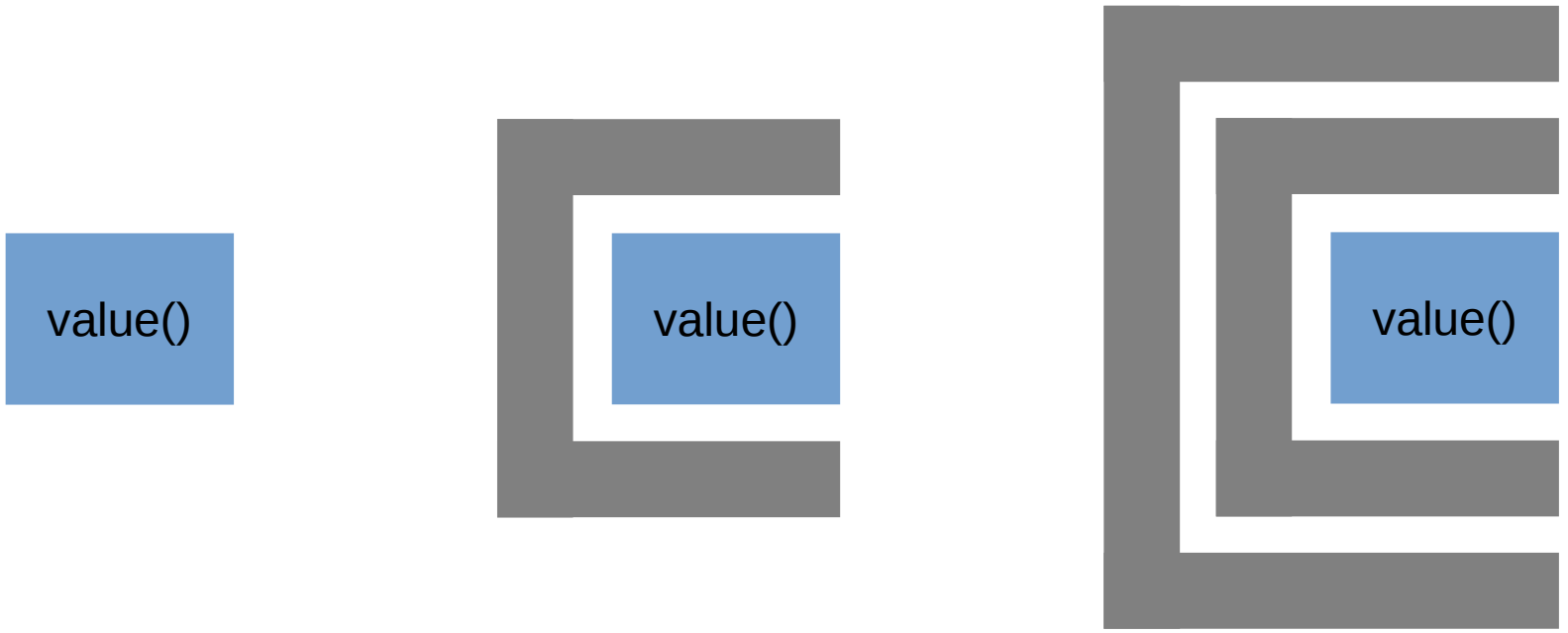
Page Object for i2

```
class Text2Frame(BasePageWidget):
    def __init__(self, locator):
        ...

    # getter
    def value(self):
        ...

    # setter
    def value(self, text):
        frame1 = self.find_element('#frame1')
        self._browser.switch_to_frame(frame1)
        frame2 = self.find_element('#frame2')
        self._browser.switch_to_frame(frame2)
        e = self.find_element(self.locator)
        e.clear()
        e.send_keys(text)
        self._browser.switch_to_default_content()

    def append(self, text):
        ...
```



IframeWrap Design Pattern

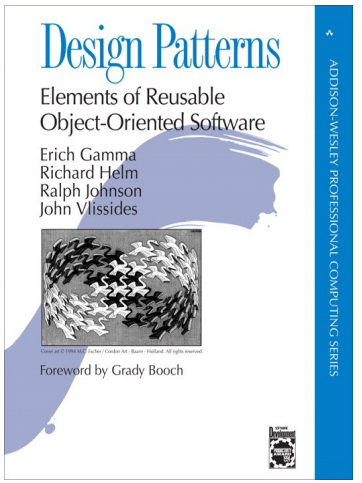
Use Decorator Pattern to wrap attributes of a page object with Enter / Exit iframe calls.



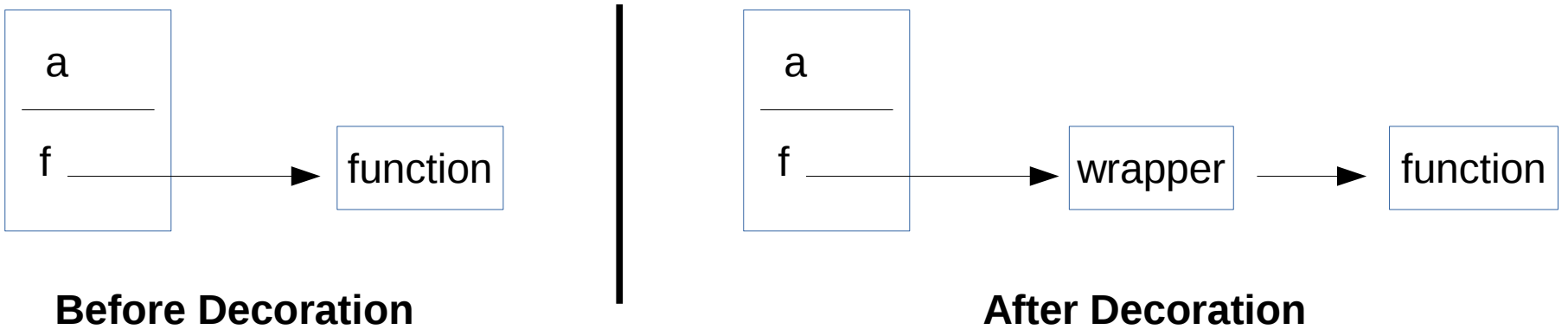
0 Iframes **1 Iframe** **2 Iframes**

Decorator Pattern

Attach additional responsibilities to an object dynamically and transparently.




```
class A (object):

    def f (self):
        do_some_stuff()

a = A()
```



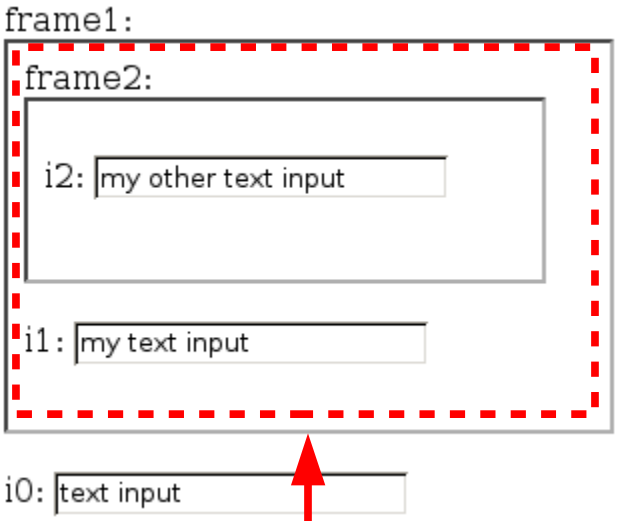
Before Decoration

After Decoration

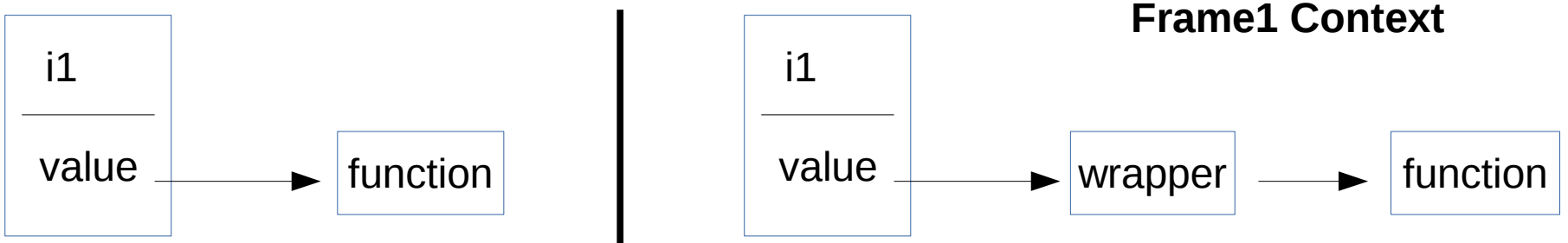
Applying the Decorator Pattern

```
class Text(BasePageWidget):

    # setter
    def value(self, text):
        e = self.find_element(self.locator)
        e.clear()
        e.send_keys(text)
```

```
i1 = Text('#i1')
i1.value('new i1 text')
```



Frame1 Context

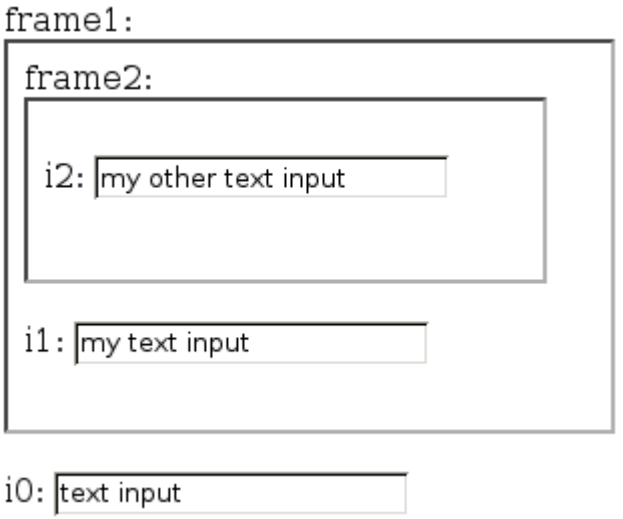


Before Decoration

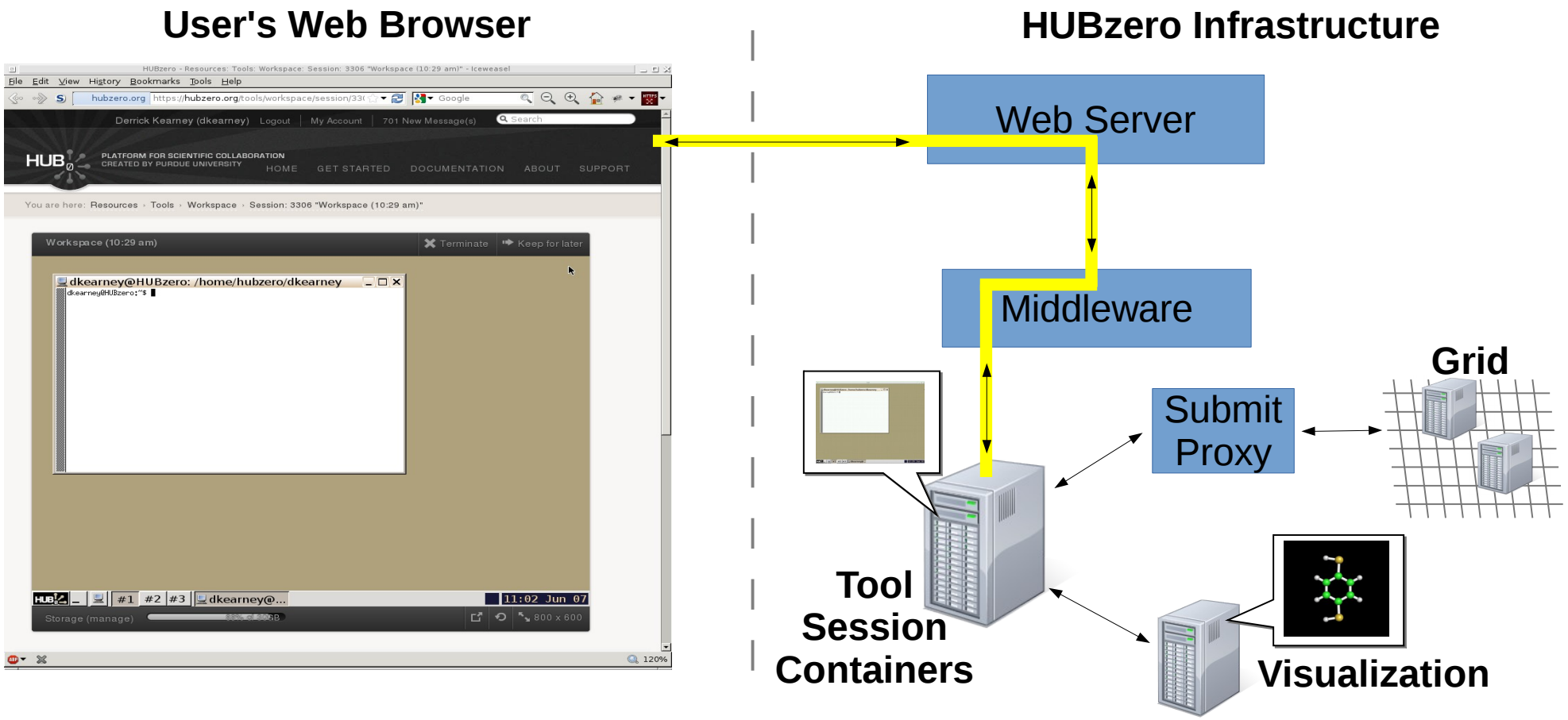
After Decoration

Using an IframeWrap'd page object

```
class FramedInputs(BasePageObject):
    def __init__(self):
        self.i0 = Text('#i0')
        self.i1 = IframeWrap( Text('#i1'), ['#frame1'] )
        self.i2 = IframeWrap( Text('#i2'), ['#frame2', '#frame1'] )


po = FramedInputs()

# print out the current text in the widgets
print "i0.value = %s" % (po.i0.value)
print "i1.value = %s" % (po.i1.value)
print "i2.value = %s" % (po.i2.value)

# update the text in the widgets
po.i0.value = 'i0 text'
po.i1.value = 'new i1 text'
po.i2.value = 'new i2 text too'
```



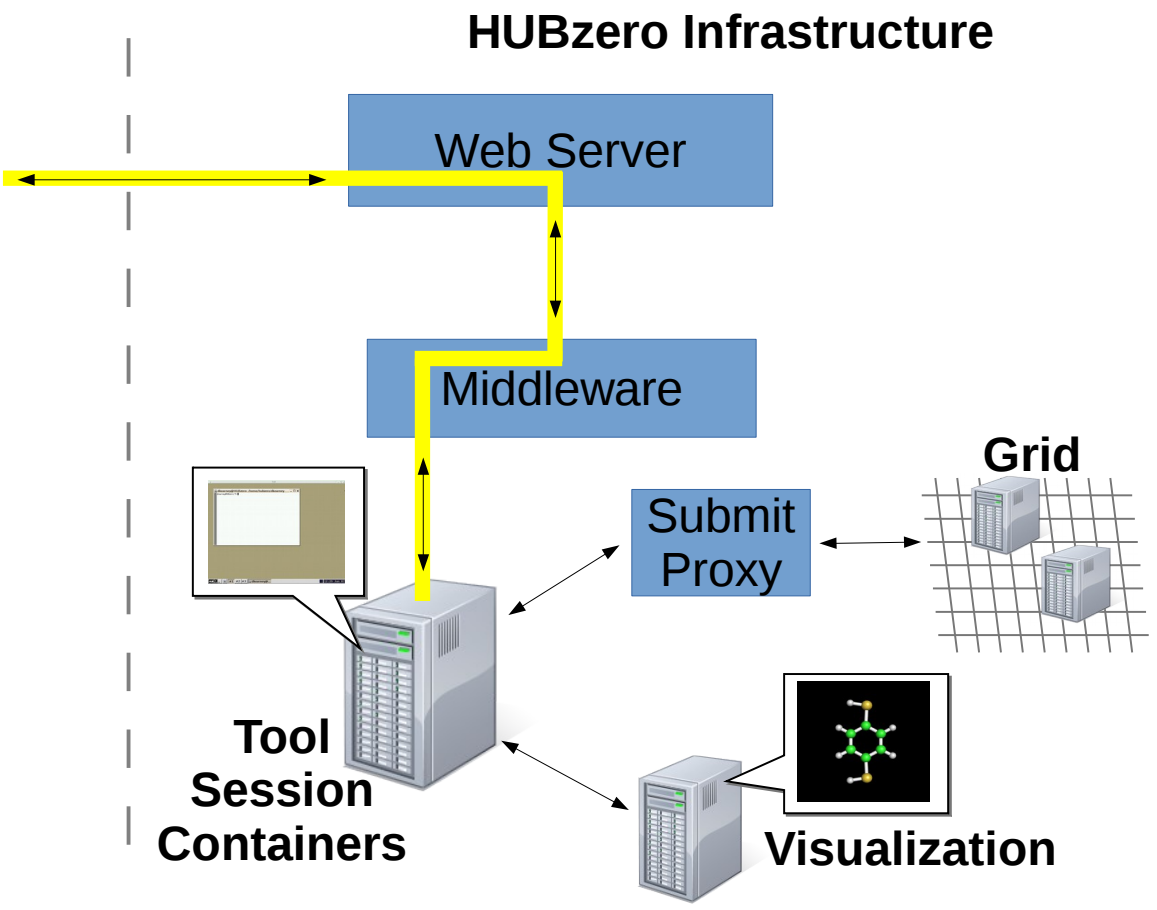
Accessing the hub shell



Accessing the hub shell

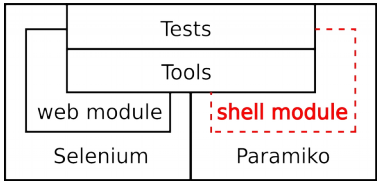
User's Terminal

```
$ ssh username@hub session

Entering session 1234

hub $
```



SSH + paramiko

Shell Automation

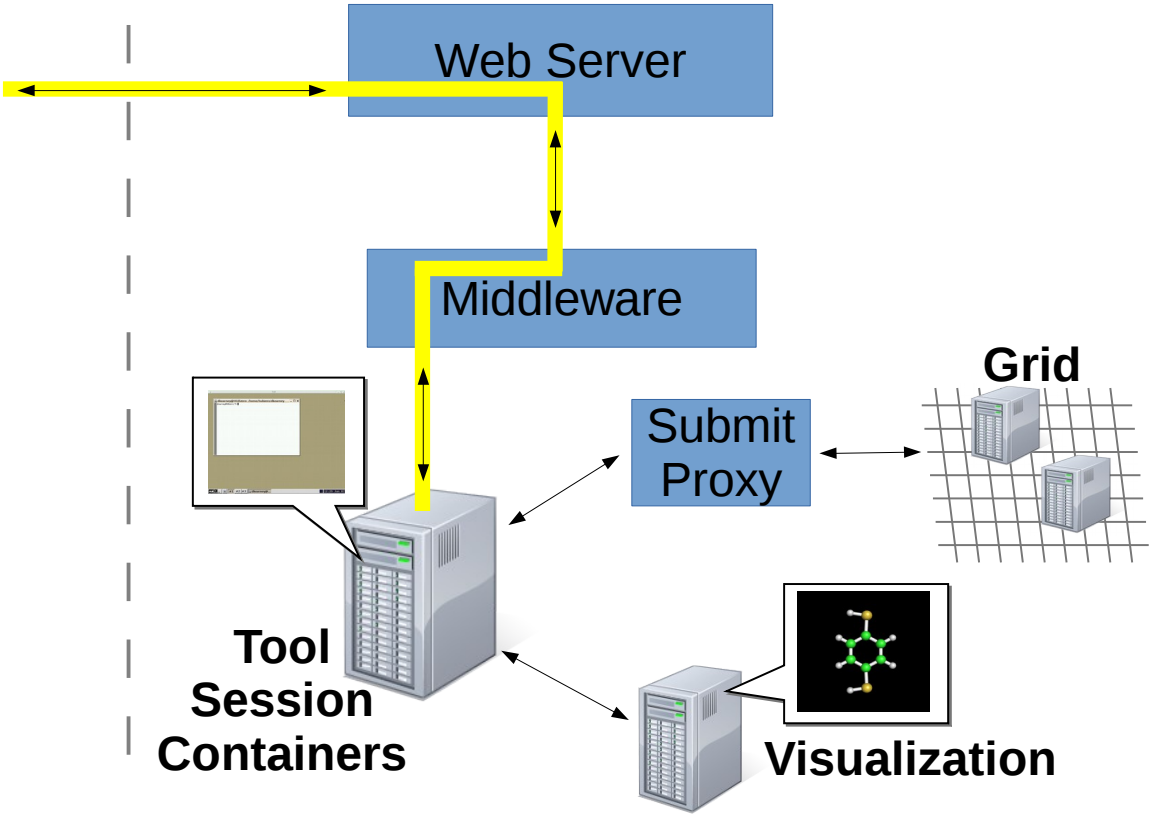


Python Script

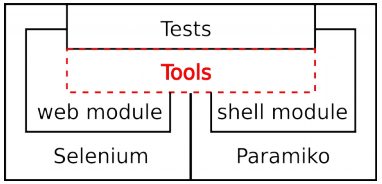
```
import hubcheck

cm = hubcheck.ContainerManager()

shell = cm.access(host,user,passwd)

out,status = shell.execute('echo hi')
```

Expect-like interface for running commands

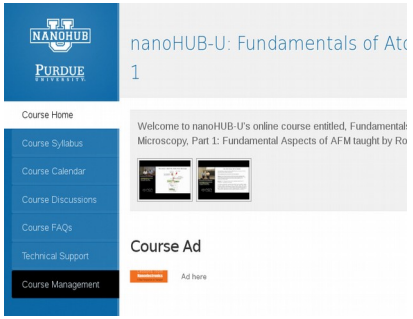
HUBzero Infrastructure



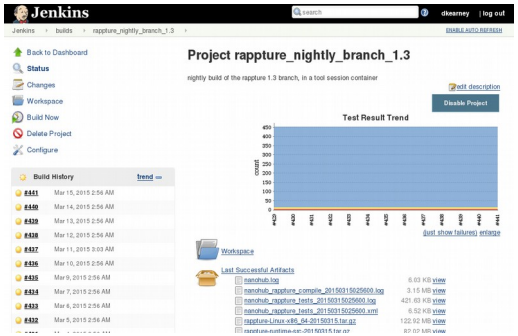
HUBcheck Tools



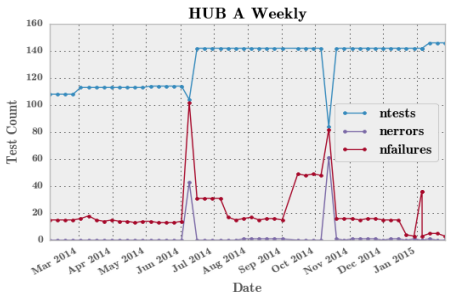
NANOHUB-U Course Registration



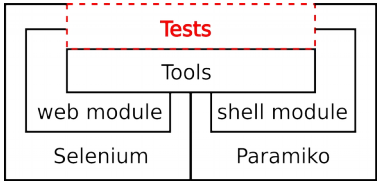
Nightly Rappture Builds



Nightly/Weekly Test Suites



Test Runner

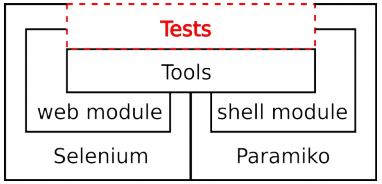


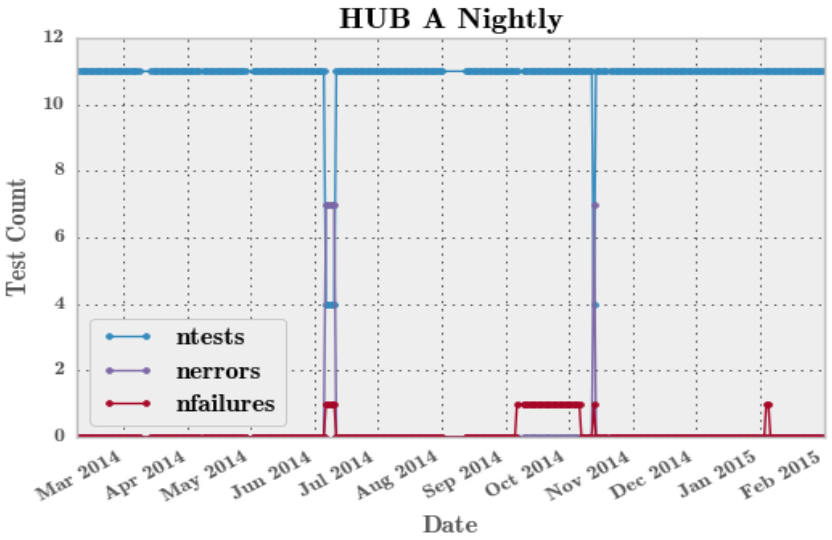
Nightly Test Suite
* covers access to workspace
* short jobs to make sure submit and webdav are working

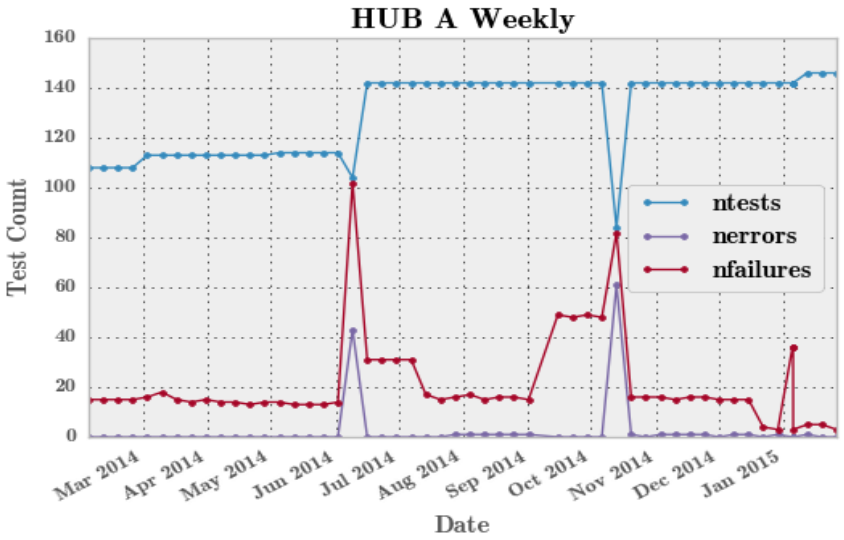
Weekly Test Suite
* more in depth testing of workspace and website
* check workspace setup
* help track hub configuration problems.

Promoting Healthier Hubs

Turbulent Hubs



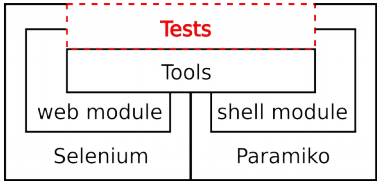


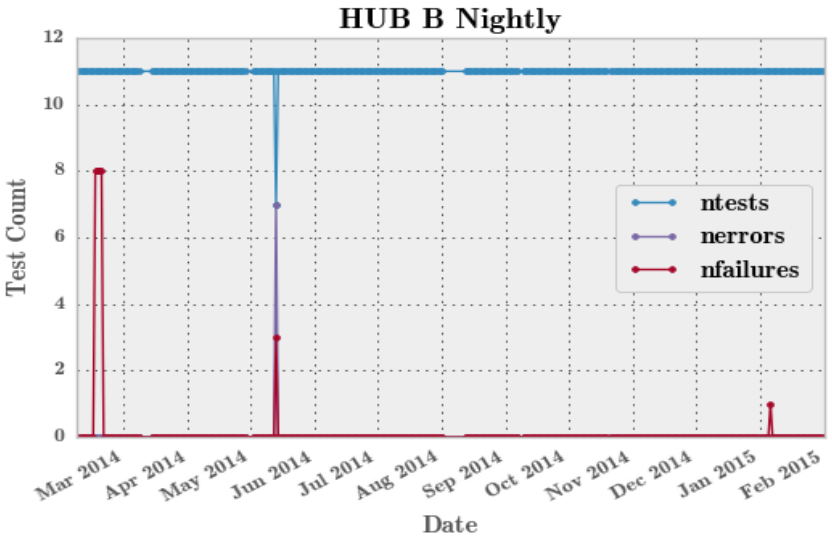


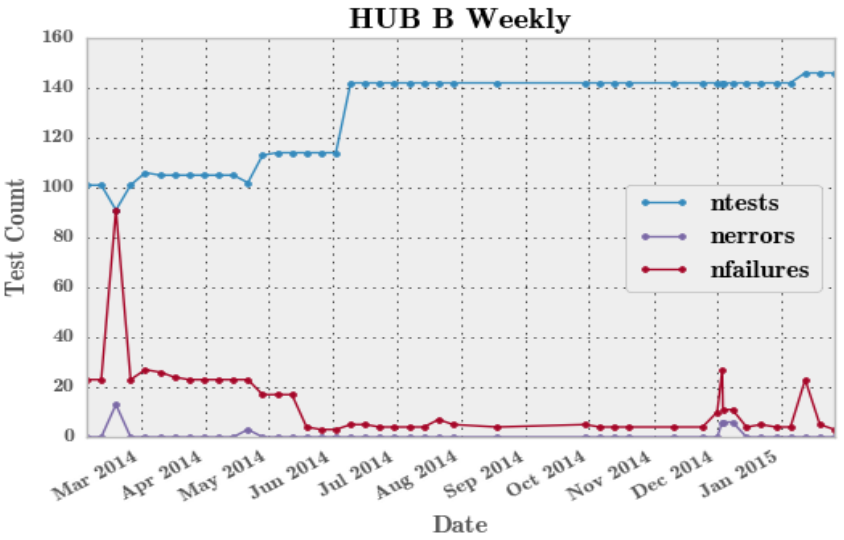
Characterized by:

- High test failure count
- Long spikes of elevated test failures

Upgraded Hubs



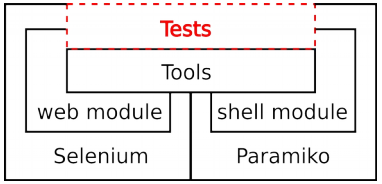


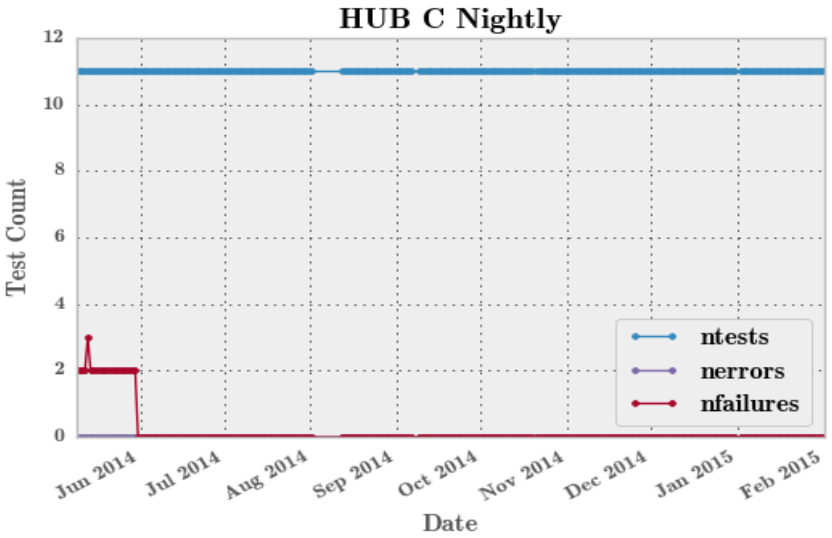


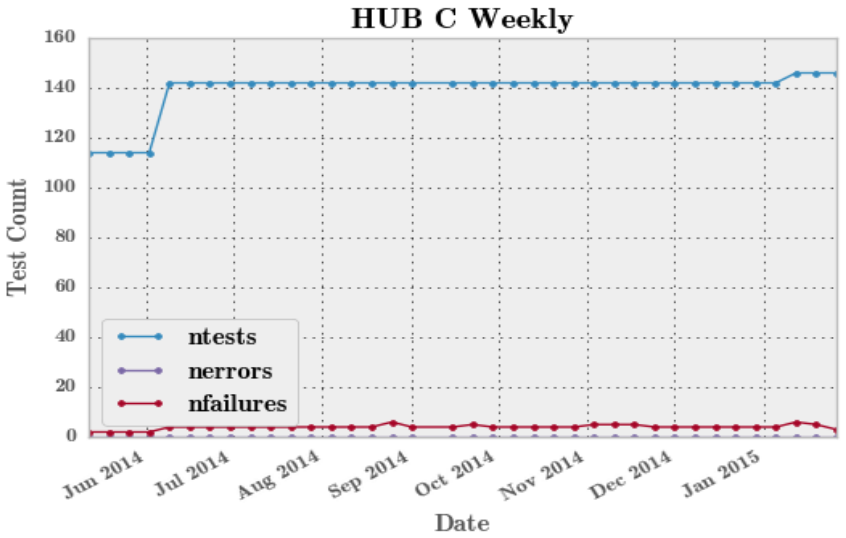
Characterized by:

- Low test failure rate
- Small spikes in failures that are quickly resolved

New Hubs







Characterized by:

- Few test failures
- Generally ~~boring~~ uneventful

Future Work

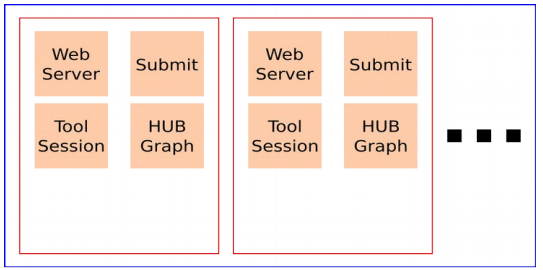
Library Improvements

Page Objects

Debian Packaging

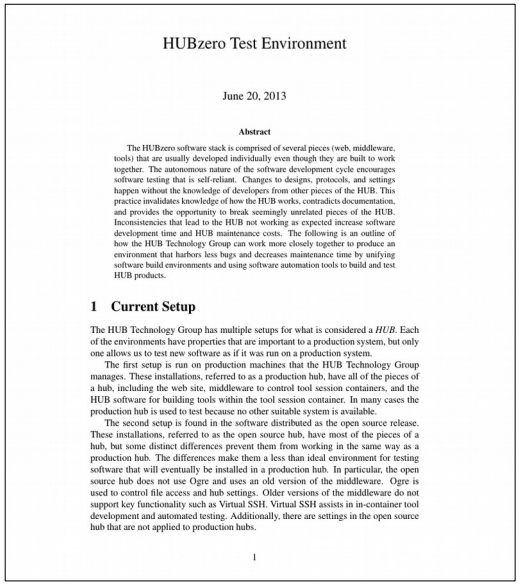
GitHub + hub zero

Test Environments



Proposed Testing Environment Using VMWare, VirtualBox, or KVM Virtual Machines.

Physical Machine
Virtual Machine
OpenVZ Container



HUBzero Test Environment

June 20, 2013

Abstract

The HUBzero software stack is comprised of several pieces (web, middleware, tools) that are usually developed individually even though they are built to work together. The autonomous nature of the software development cycle encourages software testing that is self-reliant. Changes to designs, protocols, and settings happen without the knowledge of developers from other pieces of the HUB. This practice invalidates knowledge of how the HUB works, contradicts documentation, and provides the opportunity to break seemingly unrelated pieces of the HUB. Inconsistencies that lead to the HUB not working as expected increase software development time and HUB maintenance costs. The following is an outline of how the HUB Technology Group can work more closely together to produce an environment that harbors less bugs and decreases maintenance time by unifying software build environments and using software automation tools to build and test HUB products.

1 Current Setup

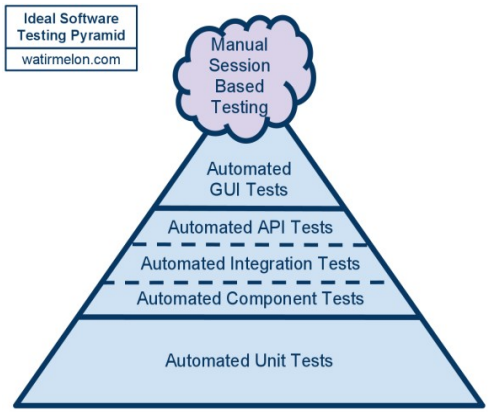
The HUB Technology Group has multiple setups for what is considered a *HUB*. Each of the environments have properties that are important to a production system, but only one allows us to test new software as if it was run on a production system.

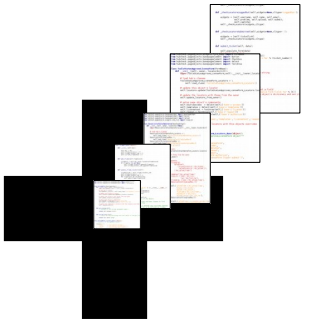
The first setup is run on production machines that the HUB Technology Group manages. These installations, referred to as a production hub, have all of the pieces of a hub, including the web site, middleware to control tool session containers, and the HUB software for building tools within the tool session container. In many cases the production hub is used to test because no other suitable system is available.

The second setup is found in the software distributed as the open source release. These installations, referred to as the open source hub, have most of the pieces of a hub, but some distinct differences prevent them from working in the same way as a production hub. The differences make them a less than ideal environment for testing software that will eventually be installed in a production hub. In particular, the open source hub does not use Ogre and uses an old version of the middleware. Ogre is used to control file access and hub settings. Older versions of the middleware do not support key functionality such as Virtual SSH. Virtual SSH assists in in-container tool development and automated testing. Additionally, there are settings in the open source hub that are not applied to production hubs.

1

Evangelization





Future Work

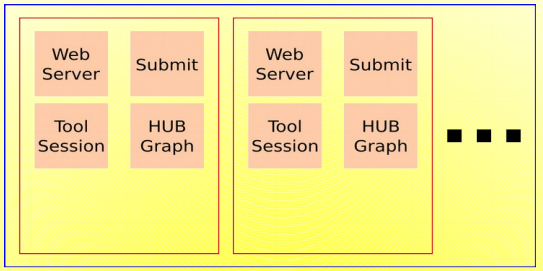
Library Improvements

Page Objects

Debian Packaging

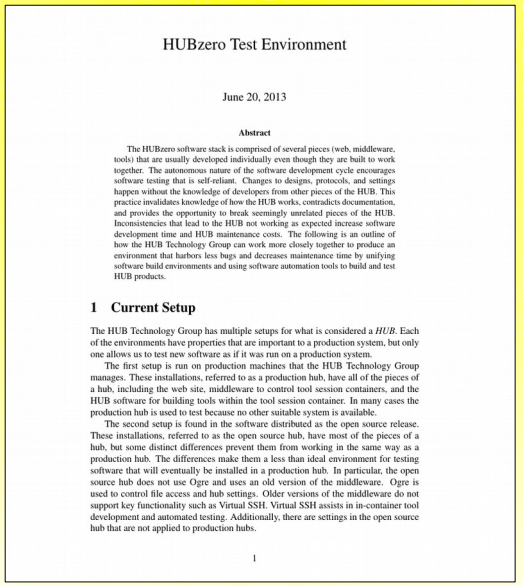
GitHub + hub zero

Test Environments



Proposed Testing Environment Using VMWare, VirtualBox, or KVM Virtual Machines.

Physical Machine
Virtual Machine
OpenVZ Container



HUBzero Test Environment

June 20, 2013

Abstract

The HUBzero software stack is comprised of several pieces (web, middleware, tools) that are usually developed individually even though they are built to work together. The autonomous nature of the software development cycle encourages software testing that is self-reliant. Changes to designs, protocols, and settings happen without the knowledge of developers from other pieces of the HUB. This practice invalidates knowledge of how the HUB works, contradicts documentation, and provides the opportunity to break seemingly unrelated pieces of the HUB. Inconsistencies that lead to the HUB not working as expected increase software development time and HUB maintenance costs. The following is an outline of how the HUB Technology Group can work more closely together to produce an environment that harbors less bugs and decreases maintenance time by unifying software build environments and using software automation tools to build and test HUB products.

1 Current Setup

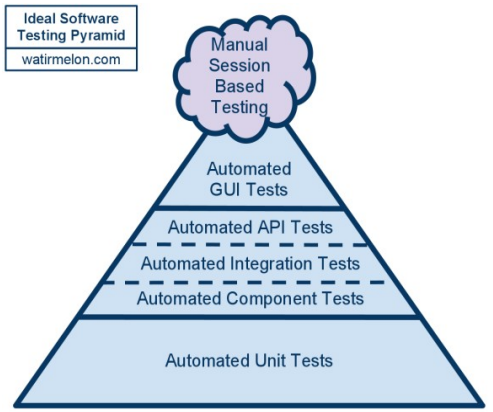
The HUB Technology Group has multiple setups for what is considered a *HUB*. Each of the environments have properties that are important to a production system, but only one allows us to test new software as if it was run on a production system.

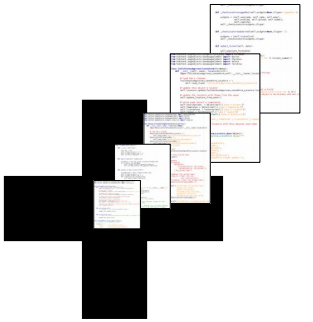
The first setup is run on production machines that the HUB Technology Group manages. These installations, referred to as a production hub, have all of the pieces of a hub, including the web site, middleware to control tool session containers, and the HUB software for building tools within the tool session container. In many cases the production hub is used to test because no other suitable system is available.

The second setup is found in the software distributed as the open source release. These installations, referred to as the open source hub, have most of the pieces of a hub, but some distinct differences prevent them from working in the same way as a production hub. The differences make them a less than ideal environment for testing software that will eventually be installed in a production hub. In particular, the open source hub does not use Ogre and uses an old version of the middleware. Ogre is used to control file access and hub settings. Older versions of the middleware do not support key functionality such as Virtual SSH. Virtual SSH assists in in-container tool development and automated testing. Additionally, there are settings in the open source hub that are not applied to production hubs.

1

Evangelization





Future Work

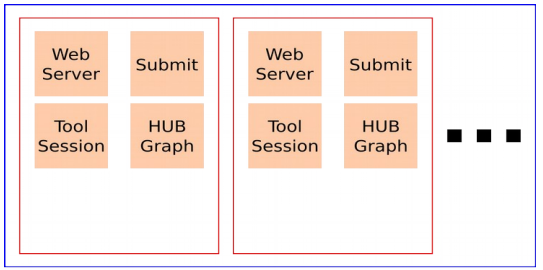
Library Improvements

Page Objects

Debian Packaging

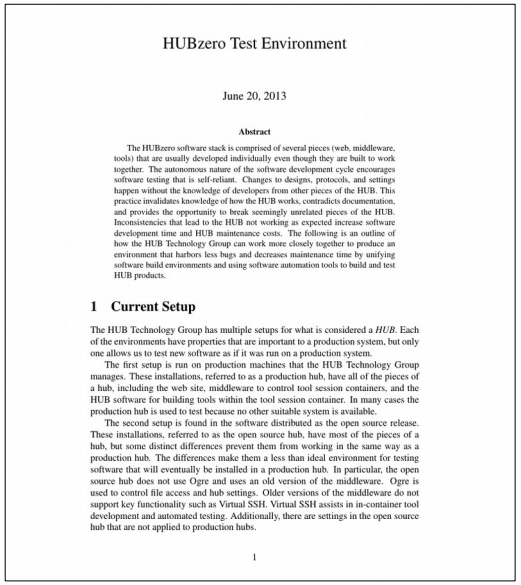
GitHub + hub zero

Test Environments



Proposed Testing Environment Using VMWare, VirtualBox, or KVM Virtual Machines.

Physical Machine
Virtual Machine
OpenVZ Container



HUBzero Test Environment

June 20, 2013

Abstract

The HUBzero software stack is comprised of several pieces (web, middleware, tools) that are usually developed individually even though they are built to work together. The autonomous nature of the software development cycle encourages software testing that is self-reliant. Changes to designs, protocols, and settings happen without the knowledge of developers from other pieces of the HUB. This practice invalidates knowledge of how the HUB works, contradicts documentation, and provides the opportunity to break seemingly unrelated pieces of the HUB. Inconsistencies that lead to the HUB not working as expected increase software development time and HUB maintenance costs. The following is an outline of how the HUB Technology Group can work more closely together to produce an environment that harbors less bugs and decreases maintenance time by unifying software build environments and using software automation tools to build and test HUB products.

1 Current Setup

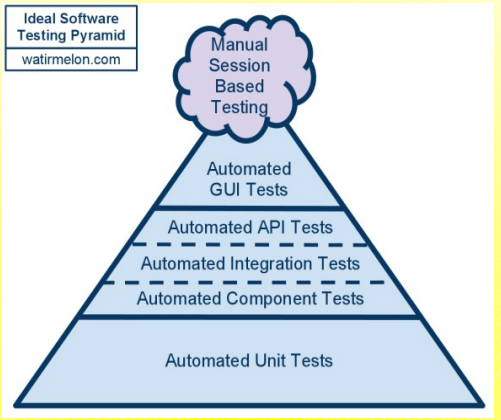
The HUB Technology Group has multiple setups for what is considered a *HUB*. Each of the environments have properties that are important to a production system, but only one allows us to test new software as if it was run on a production system.

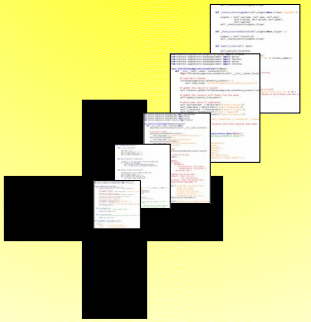
The first setup is run on production machines that the HUB Technology Group manages. These installations, referred to as a production hub, have all of the pieces of a hub, including the web site, middleware to control tool session containers, and the HUB software for building tools within the tool session container. In many cases the production hub is used to test because no other suitable system is available.

The second setup is found in the software distributed as the open source release. These installations, referred to as the open source hub, have most of the pieces of a hub, but some distinct differences prevent them from working in the same way as a production hub. The differences make them a less than ideal environment for testing software that will eventually be installed in a production hub. In particular, the open source hub does not use Ogre and uses an old version of the middleware. Ogre is used to control file access and hub settings. Older versions of the middleware do not support key functionality such as Virtual SSH. Virtual SSH assists in in-container tool development and automated testing. Additionally, there are settings in the open source hub that are not applied to production hubs.

1

Evangelization

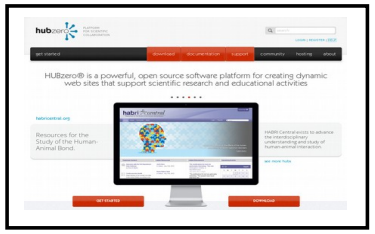


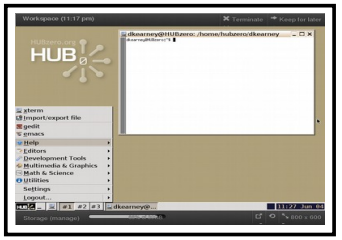


Special Thanks

Image Credits

1. Debian - https://twitter.com/debian
2. HUBzero github image - https://github.com/hubzero
3. GitHub logo – https://github.com/logos
4. Ideal Automated Testing Pyramid - http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/
5. Software Testing Icecream Antipattern - http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/
6. Firefox logo - https://www.mozilla.org/en-US/firefox/desktop/
7. Selenium logo - http://docs.seleniumhq.org/projects/webdriver/
8. Paramiko logo - http://www.lag.net/paramiko/legacy.html





mozilla Firefox

> SSH v2

Se

paramiko / **paramiko**

Native Python SSHv2 protocol library

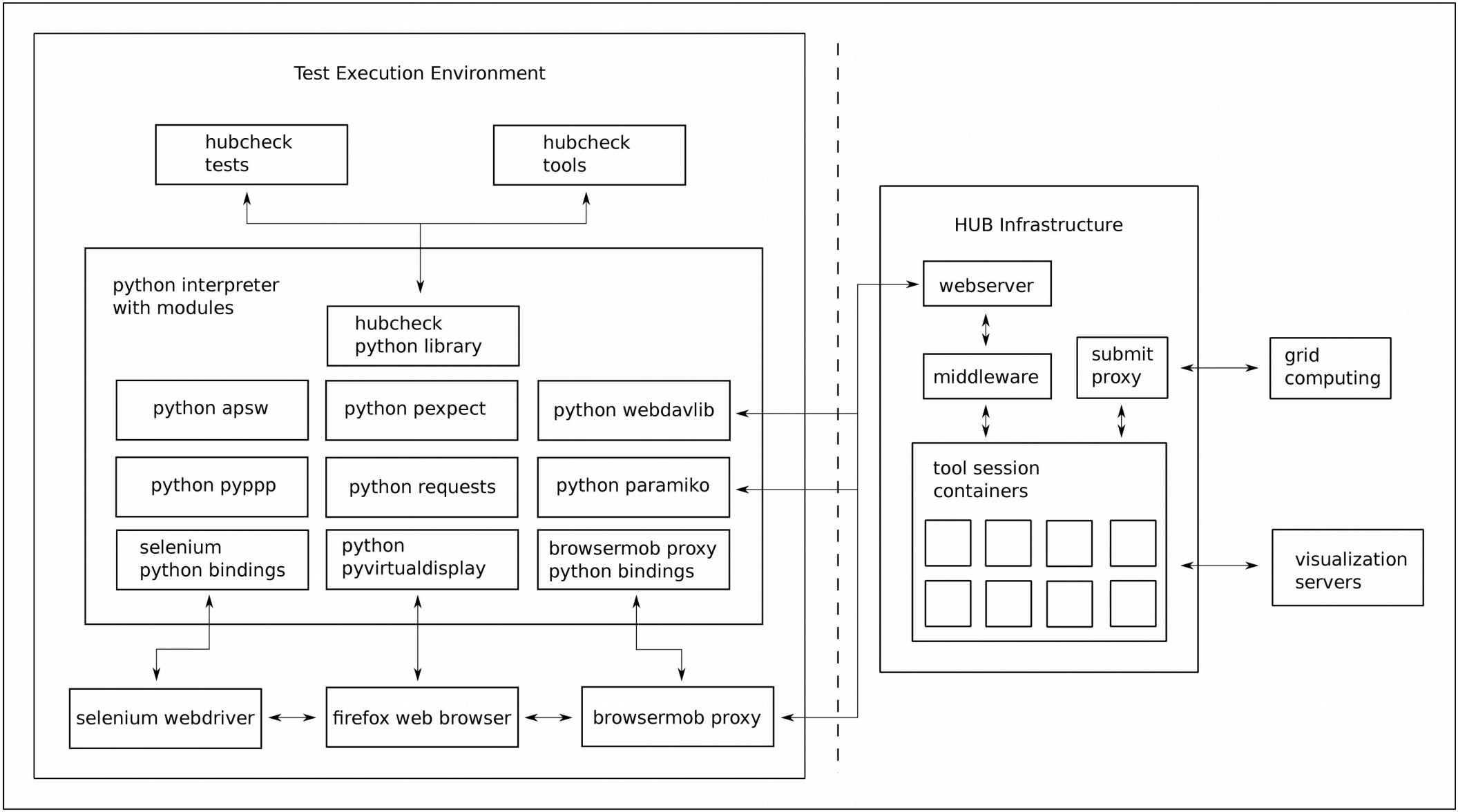
hub check

web tools
* widgets
* pageobjects
* modules

shell tools
* ToolSession
* ToolSessionShell
* SSHShell
* SSH / SFTP

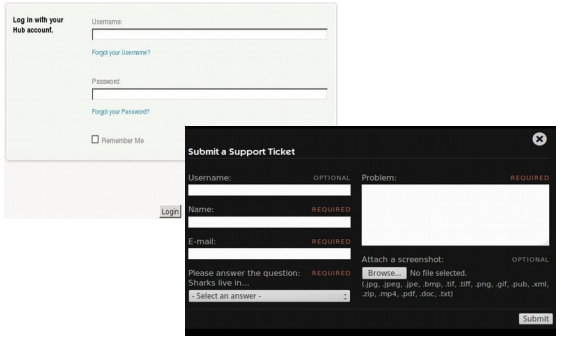
tests
* container
* website
* mixture

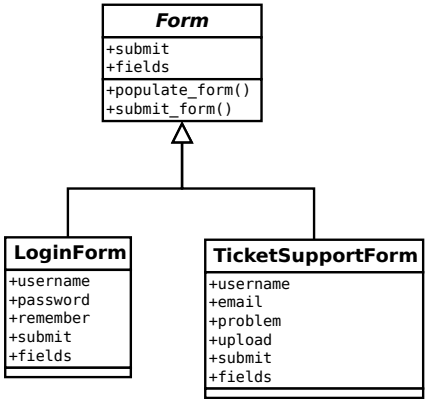
tools
* hctestrunner
* change_password
* nanohubu_enroll
* inspector
* registerAccount
* registerTool
* renderdiff


Test Execution Environment
hubcheck tests
hubcheck tools
python interpreter with modules
hubcheck python library
python apsw
python pexpect
python webdavlib
python pyppp
python requests
python paramiko
selenium python bindings
python pyvirtualdisplay
browsermob proxy python bindings
selenium webdriver
firefox web browser
browsermob proxy
HUB Infrastructure
webserver
middleware
submit proxy
grid computing
tool session containers
visualization servers

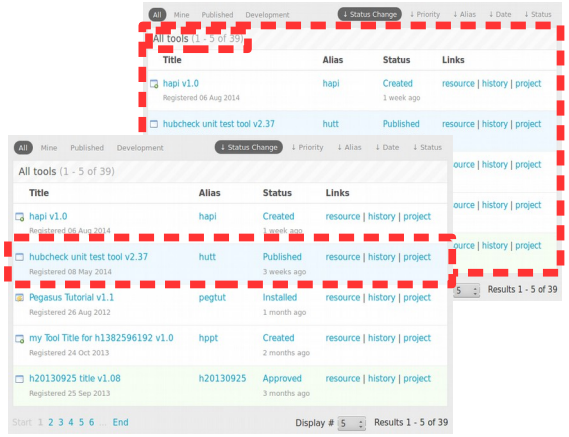
Page Object Based Design Patterns

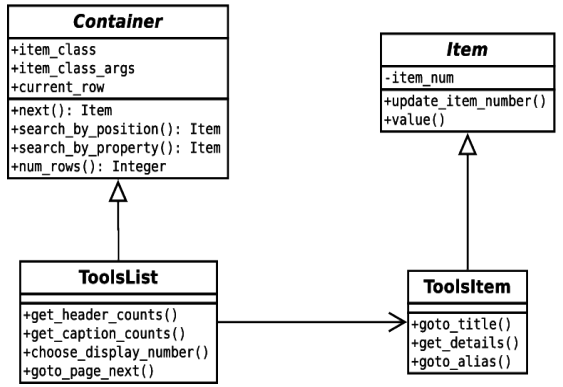
WebForm Pattern



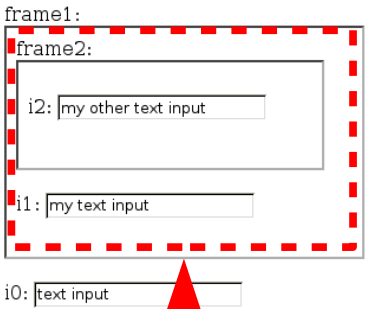


ItemList Pattern

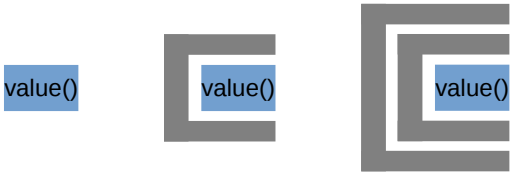




IframeWrap Pattern

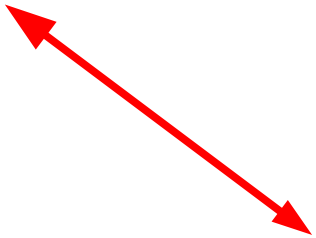


Frame1 Context



Wrap attributes with iframe calls

```
class IframeTracker(object):
    ...
    def wrap_callable_attributes(self,o):
        for attr, item in o.__class__.__dict__.items():
            if callable(item):
                item = getattr(o,attr)
                setattr(o,attr,self.wrap_attribute(item))

    def wrap_attribute(self,item):
        def wrapper(*args, **kwargs):
            ...
            switched = self._switch_to_iframe_context(final_framelevel)
            result = item(*args, **kwargs)
            self._switch_to_iframe_context(initial_framelevel)

            return result
        return wrapper
```



```
frame = self.find_element('#frame1')
self._browser.switch_to_frame(frame)
e = self.find_element(self.locator)
e.clear()
e.send_keys(text)
self._browser.switch_to_default_content()
```