SoftServe
Ability. Agility. Advantage.

Selenium RC и Python:
История одного проекта

Константин
Прищенко

Содержание

- Введение
- Selenium IDE
- Использование mapping-файла
- PageObject model
- Modified PageObject model
- Введение в py.test
- Заключение

Введение

Введение

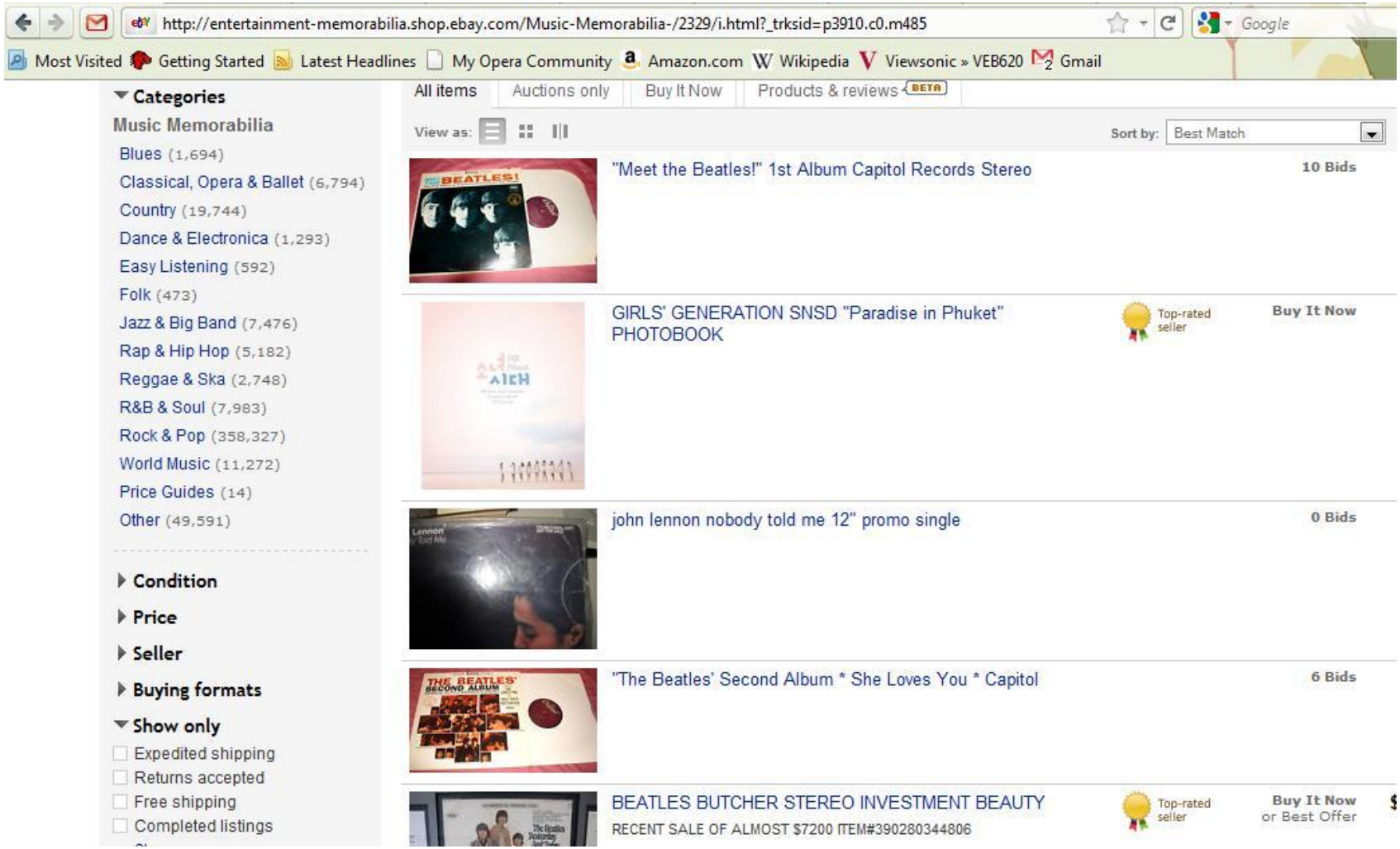
Workstation:
OS Windows XP/7

Workstation:
Mac OS X

Server:
OS Ubuntu Linux

Selenium IDE

Selenium IDE



SoftServe
Ability. Agility. Advantage.

Selenium IDE

```
def test_untitled(self):
    sel = self.selenium
    sel.open("/")
    sel.click("link=Entertainment Memorabilia")
    sel.wait_for_page_to_load("30000")
    sel.click("//tr[@id='v4-5']/td[2]/div/a")
    sel.wait_for_page_to_load("30000")
    sel.click("//a[@id='e50']/b")
    sel.wait_for_page_to_load("30000")
    sel.click("//a[@id='e49']/b")
    sel.wait_for_page_to_load("30000")
    sel.click("e31")
    sel.wait_for_page_to_load("30000")
    sel.select("v4-38", "label=Price + Shipping: lowest first")
    sel.wait_for_page_to_load("30000")
```

Selenium IDE

Преимущества:

- Простота написания тестов с использованием такого метода

Недостатки:

- Очень много повторяющегося кода.
- Локаторы объектов хранятся в тесте.

Внешний файл для хранения локаторов

Использование mapping-файла

Для обеспечения лучшего контроля вынесем объявление всех локаторов в отдельное хранилище.

Это может быть xml-файл:

```
<root>
        <obj name="Username" locator="//input[@name='userID2']"/>
        <obj name="Password" locator="//input[@name='userPsw2']"/>
        <obj name="Logout" locator="//input[@name='userOut2']"/>
</root>
```

Использование mapping-файла

Это может быть cfg/ini-файл:

```
1 [Mapping]
2 Username: //input[@name='userID2']
3 Password: //input[@name='userPsw2']
4 Logout: //input[@name='userOut2']
```

Был выбран для использования cfg-файл из-за простоты реализации и поддержки.

Использование mapping-файла

Пример кода для работы с cfg-файлом:

```
1  import ConfigParser
2
3  class Mapping(object):
4      def __init__(self):
5          _cfg = ConfigParser.SafeConfigParser()
6          _cfg.read("mapping.cfg")
7          for option in _cfg.options("Mapping"):
8              setattr(self, option, _cfg.get("Mapping", option))
```

В итоге мы получаем возможность получать локатор:

```
1  mapping = Mapping()
2  print Mapping.Username # > //input[@name='userID2']
```

Использование mapping-файла

```
1  def test_untitled(self):
2      mapping = Mapping()
3      sel = self.selenium
4      sel.open("/")
5      sel.click(mapping.ent_catalog)
6      sel.wait_for_page_to_load("30000")
7      sel.click(mapping.browse_by_music)
8      sel.wait_for_page_to_load("30000")
9      sel.click(mapping.show_free_shipping)
10     sel.wait_for_page_to_load("30000")
11     sel.click(mapping.show_expediting_shipping)
12     sel.wait_for_page_to_load("30000")
13     sel.click(mapping.style_dance_electro)
14     sel.wait_for_page_to_load("30000")
15     sel.select(mapping.sort_by,
16                "label=Price + Shipping: lowest first")
17     sel.wait_for_page_to_load("30000")
```

Использование mapping-файла

Преимущества данного подхода:

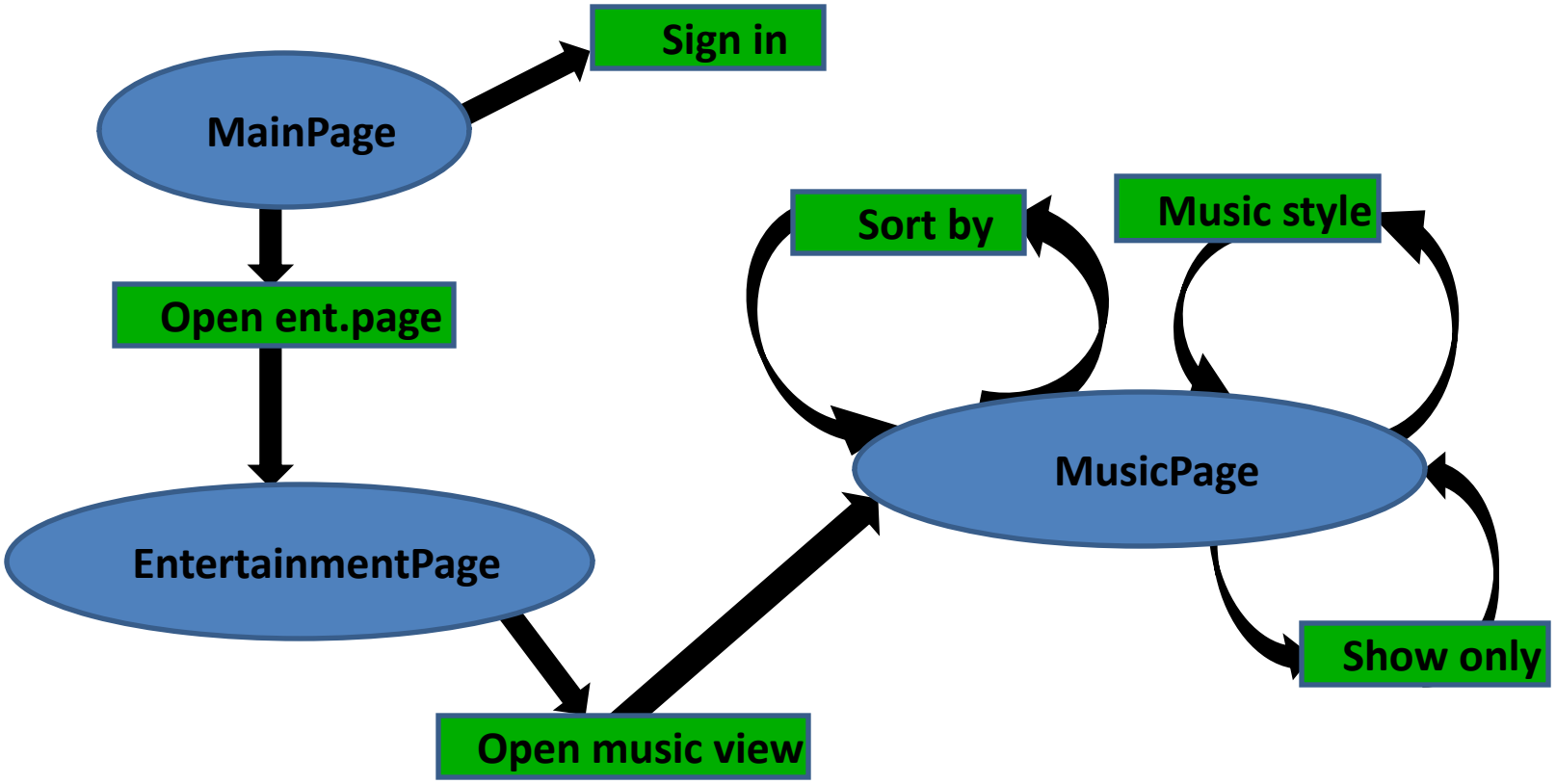
- Если локатор меняется – его необходимо обновить только в одном месте;
- Повышается читабельность кода – имена локаторов позволяют понять логику теста;

Использование Page Object модели

SoftServe
Ability. Agility. Advantage.

PageObject модель

Модель сайта:



PageObject модель

Для данной модели необходимо использовать mapping-файл с отдельной секцией для каждой страницы.

```
1  [MainPage]
2  ent_catalog = link=Entertainment Memorabilia
3
4  [EntertainmentPage]
5  music_view = //a[text()='Music Memorabilia']
6
7  [MusicPage]
8  expedited_shipping = //a/*[text()='Expedited shipping']
9  expedited_shipping = //a/*[text()='Free shipping']
10 electro_style = a//[contains(@href,"Dance-Electronica")]
11 sort_by = //div[@class='sb']/select[@id='v4-38']
```

PageObject модель

Пример класса для базовой страницы:

Создание объекта:

```
1  class Page(object):
2      def __init__(self, browser, url, load = True):
3          self._browser = browser
4          self._url = url
5          if load == True:
6              self._browser.open(self._url)
7              self._browser.wait_for_page_to_load(60000)
```

PageObject модель

Пример класса для базовой страницы:

Нажатие на кнопку/ссылку:

```
1  def click_link(self, locator, timeout = "30000"):
2      res = True
3      self._browser.click(locator)
4      try:
5          self._browser.wait_for_page_to_load(timeout)
6      except Exception, exc:
7          res = False
8          print "Exception happened while loading"
9          print exc.message
10     return res
```

PageObject модель

Пример класса для страницы:

```
1  class EntCatalogPage(Page):
2      def __init__(self, browser, load = True):
3          self._mapping = Mapping(self.__class__.__name__)
4          self._browser = browser
5          Page.__init__(self, self._browser,
6                        self._mapping.url, load)
7
8      def open_music_view(self):
9          locator = self._mapping.browse_by_music
10         res = self.click_link(locator)
11         return MusicViewPage(self._browser, not res)
```

PageObject модель

```
1  def test_untitled(self):
2      views_list = ["free shipping", "expediting shipping"]
3      music_style = "dance_electro"
4      sort_type = "Price + Shipping: lowest first"
5
6      sel = self.selenium
7      main_page = MainPage(sel, True)
8      ent_page = main_page.open_ent_catalog()
9      music_page = ent_page.open_music_view()
10     music_page.select_views(views_list)
11     music_page.view_style(dance_style)
12     music_page.sort_by(sort_type)
```

PageObject модель

Данный подход позволяет:

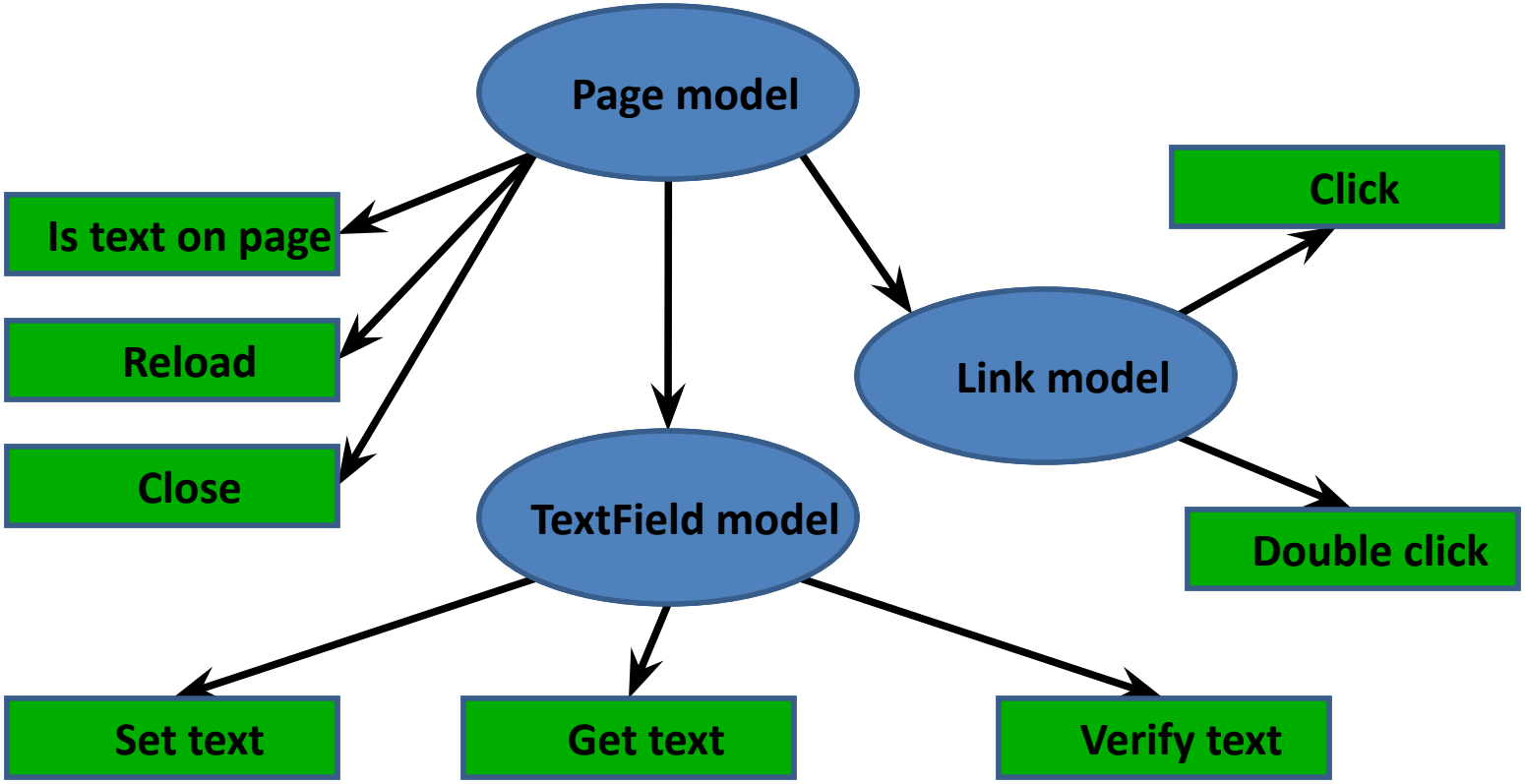
- Улучшить структуру тестов;
- Уменьшить повторяемость кода;
- Увеличить читабельность кода;
- Упрощается поддержка тестов в случае изменения тестируемого приложения.

Использование модифицированной Page Object модели

SoftServe
Ability. Agility. Advantage.

Modified PageObject модель

Модель сайта:



Modified PageObject модель

Выделим следующий элемент в логической структуре PageObject модели — объект элемента страницы, что обеспечит:

- Улучшенную структуру объектов страниц;
- Уменьшит повторяемость кода;
- Уменьшит время на ознакомление с общей моделью.

Modified PageObject модель

Данный подход позволит поместить необходимые проверки во внутрь методов элементов:

```
1  def set_text(self, text, verify = True):
2      res = False
3      if (self.is_exists()):
4          self._sel.type(self._locator, text)
5          if verify == True:
6              res = self.verify_text(text)
7          else:
8              res = True
9      else:
10         msg = "element does not exist '%s'" % self._locator
11         raise Exception(msg)
12     return res
```

Modified PageObject модель

Данный подход не изменяет внешнего вида теста.

```
1  def edit_details(self, title = '', visiability = ''):
2      if title != '':
3          locator = self._mapping.title
4          TextBox(self._browser, locator).set_text(title)
5
6      if visiability != '':
7          locator = self._mapping.item_visible
8          ComboBox(self._browser, locator).select_value(visiability)
9
10     locator = self.mapping.update
11     res = Link(self._browser, locator).click_and_wait()
12     items_page = ItemsList(self._browser, not res)
13     text = self.mapping.item_successfully_updated
14     assert items_page.is_text_on_page(text) == True
15     return items_page
```

Modified PageObject модель

С помощью данного подхода упрощается работа с о специфическими элементами различных библиотек.

Например JSON ComboBox-элемент: поле для ввода, кнопка и div-секции со всеми возможными элементами.

Modified PageObject модель

```
1  def set_value(self, value):
2      res = False
3      i = 1
4      locator = self._values_locator
5      elem = Link(self._browser, locator.replace("#", str(i)))
6      field = Link(self._browser, self._input_locator)
7      if (not elem.is_exists() or not elem.is_visible()):
8          field.clickAt()
9      while (elem.is_exists()):
10         if elem.get_text() == value:
11             elem.click()
12             res = (field.get_value() == value)
13             break
14         i += 1
15         elem.set_locator(locator.replace("#", str(i)))
16     return res
```

Введение в py.test

SoftServe
Ability. Agility. Advantage.

Введение в py.test

Система разработана Holger Krekel-ом.

Инсталляция:

```
easy_install pytest
pip install pytest
```

Запуск через командную строку:

```
py.test
```

Введение в py.test

Стандартные операции ввода/вывода показываются только в случае ошибки

Автоматическое распознование тестов:

- ищутся все файлы test_*.py
- внутри файлов ищутся все функции test_* и все Test классы.

Введение в py.test

Для верификации значений в тесте используется assert:

```
1  def test_assert_introspection():
2                      #with unittest.py
3      assert x        # assertTrue()
4      assert x == 1   # assertEqual(x,1)
5      assert x != 2   # assertNotEqual(x,2)
6      assert not x    # assertFalse(x)
```

Введение в py.test

Можно пропустить тест:

```
1 py.test.skip(expr)
```

Можно пропустить тест при условии:

```
2 py.test.skikif("sys.platform != 'win32'")
```

Можно пометить тест как негативный:

```
3 py.test.xfail(expr)
```

Можно пометить тест своей собственной меткой:

```
4 py.test.mark.YOURMARK
```

Введение в py.test

```
1  def pytest_funcarg__browser(request):
2      # all parameters extracted before:
3      browser = selenium(host, port, browser_type, url_under_test)
4
5      try:
6          browser.start()
7      except Exception, msg:
8          raise Exception("Unable to create and start" +
9                          "Selenium object. View" +
10                         " exception:\n\t%s" % msg)
11
12     # Make sure we stop the browser session
13     # after each test
14     request.addfinalizer(lambda: browser.stop())
15
16     return browser
```

Заключение

Заключение

```
C:\Eclipse\workspace_python\project_1\tests>py.test -v test_testcreation.py
============================= test session starts ==============================
platform win32 -- Python 2.6.6 -- pytest-1.3.4 -- C:\Python26\python26.exe
test path 1: test_testcreation.py

test_testcreation.py:36: test_TC1977_admin_testcreation PASS
...
test_testcreation.py:616: test_admin_ako_test_edit FAIL
test_testcreation.py:658: test_TC2006_admin_grade_wo_passing_student PASS
test_testcreation.py:704: test_TC2009_teacher_grade_wo_passing_student PASS
test_testcreation.py:750: test_TC2007_admin_grade_wo_passing_student_mc PASS
test_testcreation.py:796: test_TC2009_teacher_grade_wo_passing_student_mc PASS
test_testcreation.py:842: test_TC2008_admin_grade_wo_passing_stud_bool PASS
test_testcreation.py:886: test_TC2011_teacher_grade_wo_passing_stud_bool PASS

=================================== FAILURES ===================================
_________________ test_TC1920_copy_shared_test_and_assign_to_grade _____________

browser = <selenium.selenium.selenium.selenium instance at 0x013943C8>

....

..\tfunc\testbuilder\tests.py:426: AssertionError
====================== 5 failed, 19 passed in 1405.88 seconds ======================
```

SoftServe
Ability. Agility. Advantage.

Заключение

По-моему мнению, связка Selenium RC + py.test с использованием PageObject модели для автоматического тестирования является наиболее оптимальной.

Построенный фреймворк позволил использовать тесты для:

- проверки новой версии приложения (BVT);
- полной проверки приложения (full regression);
- выборочной проверки выбранных модулей;
- организации помощи ручному тестированию.

Контакты

Прищенко
Константин

kprish@softserveinc.com

Спасибо за внимание!

SoftServe
Ability. Agility. Advantage.