DECO3801 Test Plan Document

THEM - Typed HTML5 Evaluation Machine
Carl Hattenfels, Scott Heiner, Shen Yong Lau, Robert Meyer, Brendan Miller, David Uebergang

Functional Test Plan

Testing Strategy

There are three major testable components of the Typed HTML5 Evaluation Machine, our web application: the front-end website, back-end parser and database. While it was easy to write Python test cases for the back-end parser, it was more difficult to test our front-end website and database with a suite of computer-run tests. Instead, we wrote up a series of scenarios that we would undertake to ensure that the web application was running correctly and as expected. Clearly, all of these scenario tests can be "implemented" as they are merely actions performed by us. This means that a test fails when some functionality is not yet implemented, or when fixing one error creates another error.

In terms of the parser, - talk about error tests being added periodically - add code for error tests that are currently implemented into appendix.

Test Case Transcript

Implications of Functional Testing

Our functional testing highlighted some issues with all aspects of our application.

User Experience Goals

We had a clear user experience in mind while developing this website. Through its ease of use and minimal effort on the part of the user, we have aimed to create a very surgical, ambient, passive experience. The tool should give users immediate insight into the issues with their HTML and websites. This is where the user's experience with our tool ends, for this session. The user now can go and fix their file externally, return to our program and almost instantly receive another assessment of their code's validity. We do not aim to get the user invested in our system. However, we wish to create a reliable and worthwhile experience, brief as it is. The user should not be frustrated by the errors the program reveals, with the focus on helping the user learn and develop better web practices. It is meant to be a program that a user just "touches", that is, they upload their file they want to check, and then go back and fix it, and then come back to this to validate again, in a cyclic process.

Our priorities are on quick and easy use, which is why everything is instantly accessible and requires very few clicks to navigate. We have designed the website to require as few as five clicks to access the primary functionality of the system.

User Testing Plan

User Testing Strategy

Our web application will eventually be utilised by two user groups - students of DECO1400, and students of DECO7140. As such, we determined four major user testing groups:

- Undergraduate students who have already completed DECO1400
- Undergraduate students who have not completed DECO1400 but have worked with computers
- Masters students who have already completed DECO7140
- Masters students who have not already completed DECO7140

However, poor initial consideration due to this highly targeted user base caused us to primarily focus on students who hadn't done DECO1400, as these students represented students "new" to DECO1400. Focusing also on past DECO1400 students would have allowed us to get a feel for people who had previously done the course, and could determine whether the tool would have been worthwhile to them. Poor communication on our part lead to us getting very few in this category. Ultimately, we got information from ten users - five were undergraduates who had not done DECO1400, one was an undergradute who had done DECO1400 and four were masters students who had done DECO7140.

Our primary strategy for user testing was as follows.

1. Prepare/lay out materials for the participant so that everything is ready.
2. Introduce ourselves to the participant and give them a high-level idea of what they will be doing in their tasks today.
3. Ask participant to fill in and sign consent form. The test conductor will fill in their parts too.
4. Give the participant more detailed instructions about the task they are to do (i.e., access the website, upload file and validate). Ask them to think out loud or to make comments as they work. See if there are any questions from the participants before we get started, and answer these where appropriate.
5. When participant is ready, ask the participant to start on the task. Start timer/be prepared to taking clicks. Take hand notes as the participant works, according to the arrangements youve worked out amongst the non-participant group. If the participant goes a bit quiet, ask what are you thinking now? or what are you working on now? When they complete the task, move them onto the next task.
6. After completing all six scenarios, ask the participant to fill in the questionnaire. Clarify as necessary.
7. When the participant has finished filling in the questionnaire, check over the responses to make sure that all parts have been filled out, and that the answers are legible.
8. Tell the participant that the session is at an end. Thank the participant for their time.

User Test Results

Summarise the results of your tests. For each scenario-based test: tabulate your metric results against each task describe or present your users feedback during/after the test

Implications of User Testing

- in general, users had no trouble navigating the system - the result of the validation (i.e. highlighted tags) was not well understood by users

Close with a general discussion that: summarises the issues raised identifies areas for improvement and design suggestions outlines any redefinition of functional and user test plans for final prototype (We dont care whether your prototype passes the tests. What is important is that your prototype is sufficiently broad to validate your test plan for the final product.)

Appendix - Python Test Code

Syntax Tests

```
from __future__ import absolute_import, division, unicode_literals

#from . import support
import unittest, html5lib
from html5lib import treebuilders

class TestSyntax(unittest.TestCase):
    """
    Provides a number of test cases to test the syntax used
    in the document.
    """

    def setUp(self):
        self.parser = html5lib.HTMLParser(tree=treebuilders.
            getTreeBuilder("etree"))

    def test_malformed_tag_name(self):
        """
        Test that the tag name isn't an invalid symbol.

        Input:
        A HTML fragment containing a tag with an invalid tag name.

        Expected Results:
        An error should be thrown reporting an invalid tag name.
        """

        inputFragmentEmptyName = "<html>< ></body>"
        inputFragmentQuestionMark = "<html><?></body>"
        inputFragmentRightBracket = "<html><></html>"

        self.parser.parse(inputFragmentEmptyName)

        self.assertIn(((6, 6), u'expected-tag-name', {u'data': u' '}),
            self.parser.errors, "Failed to report invalid tag name. Get "
                )

        self.parser.reset()
        self.parser.parse(inputFragmentQuestionMark)

        self.assertIn(((6, 6), u'expected-tag-name-but-got-question-mark'
            , {}),
            self.parser.errors, "Failed to report valid tag name. Got
                question mark instead.")

        self.parser.reset()
        self.parser.parse(inputFragmentRightBracket)

        self.assertIn(((6, 7), u'expected-tag-name-but-got-right-bracket'
            , {}),
            self.parser.errors, "Failed to report valid tag name. Got
                question mark instead.")
```

```
47
48      def test_self_closing_end_tag(self):
49          """
50          Test that a closing tag with a misplaced forwardslash
51          raises an error.
52
53          Input:
54          A HTML fragment containing a closing tag with a misplaced
                forwardslash.
55
56          Expected Results:
57          An error should be thrown reporting an invalid tag name.
58          """
59
60          inputFragment = "<html><a></a /></html>"
61
62          self.parser.parse(inputFragment)
63
64          self.assertIn(((9, 14), u'self-closing-flag-on-end-tag', {}),
65              self.parser.errors, "Failed to report misplaced forwardslash
                    in closing tag.")
66
67      def test_invalid_self_closing_tag(self):
68          """
69          Test that the use of a self closing tag for a tag
70          which isn't considered a self closing tag returns
71          an error.
72
73          Input:
74          A HTML fragment containing a start tag with a trailing
                forwardslash
75          (self-closing) for a tag type which isn't a self closing tag.
76
77          Expected Results:
78          An error should be thrown reporting the given tag type isn't a
                self-closing
79          tag.
80          """
81
82          inputFragment = "<html><a /></html>"
83
84          self.parser.parse(inputFragment)
85
86          self.assertIn(((6, 10), u'non-void-element-with-trailing-solidus'
                , {u'name': u'a'}),
87              self.parser.errors, "Failed to report invalid self-closing
                    tag.")
88
89      def test_attributes_in_end_tag(self):
90          """
91          Test that attributes occuring in a closing tag are
92          reported as an error.
93
94          Input:
95          A HTML fragment containing a closing tag which contains
96          at least one attribute.
97
```

```
98          Expected Results:
99          An error should be thrown reporting that the closing tag shouldn't contain
100         attributes.
101         """
102
103         inputFragment = '<html><a></a src="blah"></html>'
104
105         self.parser.parse(inputFragment)
106
107         self.assertIn(((9, 23), u'attributes-in-end-tag', {}),
108             self.parser.errors, "Failed to report attributes in closing tag.")
109
110 if __name__ == '__main__':
111     unittest.main()
```

Page Structure Tests

```
from __future__ import absolute_import, division, unicode_literals

#from . import support
import unittest, html5lib
from html5lib import treebuilders

class TestPageStructure(unittest.TestCase):
    """
    Provides a number of test cases related to basic page structure
    for html5 documents.
    """

    def setUp(self):
        self.parser = html5lib.HTMLParser(tree=treebuilders.
            getTreeBuilder("etree"))

    def test_singular_tags(self):
        """
        Test that the multiple-instance-singular-tag error is thrown
        for cases where more than one instance of a singular tag block is
        present.

        Input:
        Nested blocks of singular tags (html, body, head).
        eg. <html><html></html></html>

        Ouput:
        All three test cases should report a multiple instance of both
        the start and closing tags for each of the three singular tags.
        """
        multipleHTMLInstances = "<html><html></html></html>"
        multipleHeadInstances = "<html><head><head></head></head><body></
            body></html>"
        multipleBodyInstances = "<html><head></head><body><body></body></
            body></html>"

        self.parser.parse(multipleHTMLInstances)

        self.assertIn(((6, 11), u'multiple-instance-singular-tag', {u'
            name': u'html'}),
            self.parser.errors, "Multiple instances of starting HTML tag
                not reported.")

        self.assertIn(((12, 18), u'incorrect-placement-html-end-tag', {u'
            name': u'html'}),
            self.parser.errors, "Multiple instances of closing HTML tag
                not reported.")

        self.parser.reset()
        self.parser.parse(multipleHeadInstances)

        self.assertIn(((12, 17), u'multiple-instance-singular-tag', {u'
            name': u'head'}),
            self.parser.errors, "Multiple instances of starting HTML tag
                not reported.")
```

```
47
48          self.assertIn(((25, 31), u'incorrect-placement-singular-end-tag',
                {u'name': u'head'}),
49              self.parser.errors, "Multiple instances of closing head tag
                    not reported.")
50
51          self.parser.reset()
52          self.parser.parse(multipleBodyInstances)
53
54          self.assertIn(((25, 30), u'multiple-instance-singular-tag', {u'
                name': u'body'}),
55              self.parser.errors, "Multiple instances of starting HTML tag
                    not reported.")
56
57          self.assertIn(((38, 44), u'unexpected-end-tag-after-body', {u'
                name': u'body'}),
58              self.parser.errors, "Multiple instances of closing body tag
                    not reported.")
59
60      def test_missing_doctype(self):
61          """
62          Test that the expected-doctype-but-got-start-tag error is thrown
63          for cases where no DOCTYPE is declared.
64
65          Input:
66          Nested blocks of singular tags (html, body, head), all of which
67          are missing the DOCTYPE declaration.
68          eg. <html><html></html></html>
69
70          Expected Results:
71          All test cases should report a missing DOCTYPE declaration.
72          """
73          startTagBeforeDoctype = "<html><html></html></html>"
74          endTagBeforeDoctype = "</head></head>"
75          eofBeforeDoctype = ""
76
77          self.parser.parse(startTagBeforeDoctype)
78
79          self.assertIn(((0, 5), u'expected-doctype-but-got-start-tag', {u'
                name': u'html'}),
80              self.parser.errors, "Failed to report missing DOCTYPE
                    declaration (start tag before doctype.")
81
82          self.parser.reset()
83          self.parser.parse(endTagBeforeDoctype)
84
85          self.assertIn(((0, 6), u'expected-doctype-but-got-end-tag', {u'
                name': u'head'}),
86              self.parser.errors, "Failed to report missing DOCTYPE
                    declaration (closing tag before doctype.")
87
88          self.parser.reset()
89          self.parser.parse(eofBeforeDoctype)
90
91          self.assertIn(((-1, -1), u'expected-doctype-but-got-eof', {}),
92              self.parser.errors, "Failed to report missing DOCTYPE
                    declaration (EOF before doctype.")
```

```
93
94
95      def test_closing_html(self):
96          """
97          Test that a missing HTML closing tag is reported when none
98          are present in the document.
99
100         Input:
101         Nested blocks of singular tags (head, body).
102
103         Expected Results:
104         Report whether the the closing HTML tag is present.
105         """
106         multipleHeadInstances = "<head><head></head></head>"
107         multipleBodyInstances = "<body><body></body></body>"
108
109         self.parser.parse(multipleHeadInstances);
110
111         self.assertIn(((-1, -1), u'no-closing-html-tag', {}),
112             self.parser.errors, "Failed to report missing closing HTML
                tag.")
113
114         self.parser.reset()
115         self.parser.parse(multipleBodyInstances)
116
117         self.assertIn(((-1, -1), u'no-closing-html-tag', {}),
118             self.parser.errors, "Failed to report missing closing HTML
                tag.")
119
120     def test_misplaced_tags_before_head(self):
121         """
122         Test that both start and closing tags occuring before the head
123         section are reported as being misplaced.
124
125         Input:
126         A number of instances of start and closing tags being placed
                before
127         the head section.
128
129         Expected Results:
130         Report whether or not the tags preceding the head section are
                reported
131         as being misplaced.
132         """
133         misplacedHeadTags = "<html><body></body><head></head></html>"
134         misplacedLinkTags = "<html><a></a><head></head><body></body></
                html>"
135
136         self.parser.parse(misplacedHeadTags)
137
138         self.assertIn(((6, 11), u'incorrect-start-tag-placement-before-
                head', {u'name': u'body'}),
139             self.parser.errors, "Failed to report start body tag before
                head section.")
140
141         self.assertIn(((12, 18), u'incorrect-end-tag-placement-before-
                head', {u'name': u'body'}),
```

```
142             self.parser.errors, "Failed to report closing body tag before
                    head section.")
143
144         self.parser.reset()
145         self.parser.parse(misplacedLinkTags)
146
147         self.assertIn(((6, 8), u'incorrect-start-tag-placement-before-
                head', {u'name': u'a'}),
148             self.parser.errors, "Failed to report start link (a) tag
                    before head section.")
149
150         self.assertIn(((9, 12), u'incorrect-end-tag-placement-before-head
                ', {u'name': u'a'}),
151             self.parser.errors, "Failed to report closing link (a) tag
                    before head section.")
152
153     def test_incorrect_tags_in_head(self):
154         """
155         Test that tags which don't belong in the head section
156         are reported as misplaced using the 'incorrect-start-tag-
                placement-in-head'
157         and 'incorrect-end-tag-placement-in-head' errors.
158
159         Input:
160         A HTML fragment with a pair of head tags enclosing a tag
161         pair which doesn't belong in the head phase.
162
163         Expected Results:
164         Inclusion of the 'incorrect-start-tag-placement-in-head'
165         and 'incorrect-end-tag-placement-in-head' errors being reported
166         as part of the returned array of error codes.
167         """
168         inputFragment = "<html><head><a></a></head></html>"
169
170         self.parser.parse(inputFragment)
171
172         self.assertIn(((12, 14), u'incorrect-start-tag-placement-in-head'
                , {u'name': u'a'}),
173             self.parser.errors, "Failed to report starting tag which
                    doesn't belong in the head section.")
174
175         self.assertIn(((15, 18), u'incorrect-end-tag-placement-in-head',
                {u'name': u'a'}),
176             self.parser.errors, "Failed to report closing tag which doesn
                    't belong in the head section.")
177
178     def test_tags_after_eof(self):
179         """
180         Tests that starting and closing tags occuring after the last
181         instace of a closing HTML tag are reported as an error.
182
183         Input:
184         A HTML fragment with a start and closing tag pair occuring
185         after the start and closing HTML pair.
186
187         Expected Results:
188         An error being thrown for both the start and closing tags
```

```
            occuring
189         after the HTML tags.
190         """
191
192         inputFragment = "<html></html><a></a>"
193
194         self.parser.parse(inputFragment)
195
196         self.assertIn(((13, 15), u'expected-eof-but-got-start-tag', {u'
                name': u'a'}),
197             self.parser.errors, "Failed to report start tag after closing
                    HTML tag.")
198
199         self.assertIn(((16, 19), u'expected-eof-but-got-end-tag', {u'name
                ': u'a'}),
200             self.parser.errors, "Failed to report closing tag after
                    closing HTML tag.")
201
202     def test_missing_start_tag(self):
203         """
204         Tests that a missing start tag is reported in the case
205         that a closing tag is found without a matching start tag.
206
207         Input:
208         A HTML fragment containing a closing tag without a matching
209         start tag.
210
211         Expected Results:
212         An error being thrown reporting that the matching start tag
213         is missing.
214         """
215
216         inputFragment = "<html><head></head><body></a></body></html>"
217
218         self.parser.parse(inputFragment)
219
220         self.assertIn(((25, 28), u'unexpected-end-tag', {u'name': u'a'}),
221             self.parser.errors, "Failed to report the lack of a matching
                    start tag.")
222
223     def test_misplaced_tags_after_body(self):
224         """
225         Tests that any tags occuring after the body phase
226         are reported as being incorrectly placed.
227
228         Input:
229         A HTML fragment with a pair of start and closing tags placed
230         after the closing body tag.
231
232         Expected Results:
233         An error should be thrown for both the start and closing
234         tags found after the closing body tag.
235         """
236
237         inputFragment = "<html><head></head><body></body><a></a></html>"
238
239         self.parser.parse(inputFragment)
```

```
240
241         self.assertIn(((32, 34), u'unexpected-start-tag-after-body', {u'
                name': u'a'}),
242             self.parser.errors, "Failed to report misplaced starting tag
                found after the closing body tag.")
243
244         self.assertIn(((35, 38), u'unexpected-end-tag-after-body', {u'
                name': u'a'}),
245             self.parser.errors, "Failed to report misplaced closing tag
                found after the closing body tag.")
246
247     def test_missing_closing_html_tag(self):
248         """
249         Test that a missing closing HTML tag is reported.
250
251         Input:
252         A HTML fragment missing a closing HTML tag.
253
254         Expected Results:
255         An error should be thrown stating that the closing HTML tag is
                missing.
256         """
257
258         inputFragment = "<html><head></head><body></body>"
259
260         self.parser.parse(inputFragment)
261
262         self.assertIn(((-1, -1), u'no-closing-html-tag', {}),
263             self.parser.errors, "Failed to report missing closing HTML
                tag.")
264
265     def test_early_termination_before_head(self):
266         """
267         Test that an early closing HTML tag before the head phase
268         is reported as an error.
269
270         Input:
271         A HTML fragment with the head and body sections placed after
272         a closed set of HTML tags.
273
274         Expected Results:
275         An error should be thrown stating that the closing HTML tag
276         has been found before the head phase.
277         """
278
279         inputFragment = "<html></html><head></head><body></body>"
280
281         self.parser.parse(inputFragment)
282
283         self.assertIn(((6, 12), u'early-termination-before-head', {u'name
                ': u'html'}),
284             self.parser.errors, "Failed to report early termination
                before head section.")
285
286     def test_early_termination_in_head(self):
287         """
288         Test that an early closing HTML tag in the head phase
```

```
289         is reported as an error.
290
291         Input:
292         A HTML fragment with the closing HTML tag placed within
293         the set of head tags.
294
295         Expected Results:
296         An error should be thrown stating that the closing HTML tag
297         has been found in the head phase.
298         """
299
300         inputFragment = "<html><head></html></head><body></body>"
301
302         self.parser.parse(inputFragment)
303
304         self.assertIn(((12, 18), u'early-termination-in-head', {u'name':
                u'html'}),
305             self.parser.errors, "Failed to report early termination
                    before head section.")
306
307     def test_early_termination_before_body(self):
308         """
309         Test that an early closing HTML tag before the body phase
310         is reported as an error.
311
312         Input:
313         A HTML fragment with the closing HTML tag placed before the body
314         section.
315
316         Expected Results:
317         An error should be thrown stating that the closing HTML tag
318         has been found before the body phase.
319         """
320
321         inputFragment = "<html><head></head></html><body></body>"
322
323         self.parser.parse(inputFragment)
324
325         self.assertIn(((19, 25), u'early-termination-before-body', {u'
                name': u'html'}),
326             self.parser.errors, "Failed to report early termination
                    before head section.")
327
328     def test_early_termination_in_body(self):
329         """
330         Test that an early closing HTML tag in the body phase
331         is reported as an error.
332
333         Input:
334         A HTML fragment with the closing HTML tag placed within
335         the set of body tags.
336
337         Expected Results:
338         An error should be thrown stating that the closing HTML tag
339         has been found in the head phase.
340         """
341
```

```
342         inputFragment = "<html><head></head><body></html></body>"
343 
344         self.parser.parse(inputFragment)
345 
346         self.assertIn(((25, 31), u'early-termination-in-body', {u'name':
                u'html'}),
347             self.parser.errors, "Failed to report early termination
                before head section.")
348 
349     def test_tags_between_head_body(self):
350         """
351         Test that a set of tags placed after the head section
352         but before the body section is reported as an error.
353 
354         Input:
355         A HTML fragment with a set of tags between the head
356         and body sections.
357 
358         Expected Results:
359         An error should be thrown stating that the set of tags
360         can't be placed between the head and body sections.
361         """
362 
363         inputFragment = "<html><head></head><a></a><body></body></html>"
364 
365         self.parser.parse(inputFragment)
366 
367         self.assertIn(((19, 21), u'start-tag-before-body-after-head', {u'
                name': u'a'}),
368             self.parser.errors, "Failed to report start tag after head
                phase but before body phase.")
369 
370         self.assertIn(((22, 25), u'end-tag-before-body-after-head', {u'
                name': u'a'}),
371             self.parser.errors, "Failed to report closing tag after head
                phase but before body phase.")
372 
373     def test_missing_starting_html_tag(self):
374         """
375         Test that a missing starting HTML tag is reported as an error.
376 
377         Input:
378         A HTML fragment missing a starting HTML tag.
379 
380         Expected Results:
381         An error should be thrown indicating that the fragment doesn't
382         contain a starting HTML tag.
383         """
384 
385         inputFragment = "<head></head><body></body></html>"
386 
387         self.parser.parse(inputFragment)
388 
389         self.assertIn(((-1, -1), u'no-starting-html-tag', {}),
390             self.parser.errors, "Failed to report missing starting HTML
                tag.")
391 
```

```
392     def test_unknown_doctype(self):
393         """
394         Test that a doctype with an invalid name is reported as being
395         an unknown doctype.
396 
397         Input:
398         A HTML fragment containing an invalid doctype name.
399 
400         Expected Results:
401         An error should be thrown reporting that the doctype name is
                invalid.
402         """
403 
404         inputFragment = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN
                " "http://www.w3.org/TR/html4/strict.dtd">'
405 
406         self.parser.parse(inputFragment)
407 
408         self.assertIn(((0, 89), u'unknown-doctype', {}),
409             self.parser.errors, "Failed to report unknown doctype.")
410 
411     def test_space_after_doctype(self):
412         """
413         Test that a doctype tag has a space between the doctype
                declaration
414         and the doctype name.
415 
416         Input:
417         A HTML fragment containing a doctype with no space between the
                doctype
418         declaration and the doctype name.
419 
420         Expected Results:
421         An error should be thrown reporting that there is no space
                between
422         the doctype declaration and the doctype name.
423         """
424 
425         inputFragment = '<!DOCTYPEhtml>'
426 
427         self.parser.parse(inputFragment)
428 
429         self.assertIn(((0, 8), u'need-space-after-doctype', {}),
430             self.parser.errors, "Failed to report missing space after the
                    doctype declaration.")
431 
432 if __name__ == '__main__':
433     unittest.main()
```

JSON-RPC Server Tests

```
from __future__ import absolute_import, division, unicode_literals

#from . import support
import unittest
import time
import jsonrpclib
from multiprocessing import Pool
from jsonrpclib import Server
import httplib
import simplejson as json
import base64

"""
Calls the test_concurrency method on the server. Required to be external from the
TestJsonServer class as it was causing a "pickling" error when used as a method.
"""
def getTime(time):
        return Server("http://localhost:8080").test_concurrency(time)

class TestJsonServer(unittest.TestCase):
    """
    Provides a number of test cases related to the functionality of the json
    rpc server.

    These tests require that the server is currently running. The first
    test checks that the server is running.
    """

    def setUp(self):
        self.startTime = time.time()

    def resetCurrentTime(self):
        self.startTime = time.time()

    def getExecutionTimes(self, numProcesses):
        """
        Attempts to call the getTime function with startTime as the
        only argument in a concurrent manner using numProcesses as the
        number of concurrent calls to make. The resulting times returned
        by the remote function, test_concurrency, are added to a list
        and returned.

        The timeout for each call attempt is currently set to 5 seconds.
        This will only allow for numProcesses to go up to 10. After that
        the processing times at the server side will trigger the timeout
        and result in an exception being thrown.
        """

        results = []
        times = []

        pool = Pool(processes=numProcesses)
```

```
53          for i in range(numProcesses):
54              results.append(pool.apply_async(getTime, (self.startTime,)))
55
56          for result in results:
57              times.append(result.get(timeout=5))
58
59          return times
60
61      def test_client(self):
62          """
63          Test that the server is currently running. Required for the
64          remaining server tests to run.
65
66          Input: Attempt to execute a known function on the server.
67
68          Expected Result: No exception being raised, implying that the
                server
69          is currently running.
70          """
71
72          exceptionRaised = False;
73          try:
74              getTime(self.startTime)
75          except:
76              exceptionRaised = True
77
78          self.assertFalse(exceptionRaised, "The server isn't running.")
79
80      def test_concurrent_connections(self):
81          """
82          Test that the server can handle the maxmimum number of concurrent
83          connections while receiving a response in a similar time frame
84          for all requests.
85
86          Input: 5 concurrent function calls to the server.
87
88          Expected Result: The remote function, test_concurrency, contains
                a 2
89          second sleep call. The sum of the times taken to complete each of
90          the function calls, relative to self.startTime should be between
                the
91          range 11 > totalTime >= 10.
92          """
93
94          self.resetCurrentTime()
95
96          totalTime = 0
97
98          for time in self.getExecutionTimes(5):
99              totalTime += time
100
101         self.assertTrue(totalTime >= 10 and totalTime < 11, "Failed to "
                +
102             "execute all 5 concurrent function calls within the expected
                    " +
103             "time frame.")
104
```

```
    def test_max_concurrent_connections(self):
        """
        Tests that the server processes excess function calls after the
        initial batch of calls.

        Input: 6 concurrent function calls to the server.

        Expected Result: The remote function, test_concurrency, contains a 2
        second sleep call. The server has a maximum number of concurrent
        connections of 5, so the 6th call will take slightly over 4 seconds
        to complete. The sum of the times for all 6 calls should be within
        the range 15 > totalTime >= 14.
        """

        self.resetCurrentTime()

        totalTime = 0

        for time in self.getExecutionTimes(6):
            totalTime += time

        self.assertTrue(totalTime >= 14 and totalTime < 15, "Failed to " +
            "execute all 6 concurrent function calls within the expected " +
            "time frame.")

    def test_json_rpc_correct_response(self):
        """
        Tests that the server responds as expected to a correctly
        formed JSON-RPC 2.0 request.

        Input: A correctly formed JSON-RPC 2.0 request containing
        an empty array of file names, an empty file name (direct
        input method) and a HTML fragment consisting of '<html></html>'.

        Expected Result: The returned JSON-RPC 2.0 response string
        should match the string expectedResponse, which contains
        the expected array of errors.
        """
        conn = httplib.HTTPConnection("127.0.0.1:8080")
        fragment = base64.b64encode(b'<html></html>')
        params = [{"files": [], "document": fragment, "filename": ""}]
        request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method": "parse_html",
            "params": params, "id": "A3s23"})
        header = {"Content-type": "application/json"}

        conn.request("POST", "", request, header)
        response = conn.getresponse()
        conn.close()

        expectedResponse = '{"jsonrpc": "2.0", "result": [[1, 0, 5, {"name": "html"}], [25, 6, 12, {"name": "html"}]], "id": "A3s23
```

```
            "}'
155
156         self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
157
158     def test_json_rpc_malformed_parameters(self):
159         """
160         Tests that the server responds with an error when
161         a request contains incorrect parameters.
162
163         Input: A JSON-RPC 2.0 request containing incorrectly
164         formatted parameters to be passed on to the requested
165         function.
166
167         Expected Result: The returned JSON-RPC 2.0 response string
168         should match the string expectedResponse, which contains
169         a response representing an invalid parameters error.
170         """
171         conn = httplib.HTTPConnection("127.0.0.1:8080")
172         fragment = base64.b64encode(b'<html></html>')
173         params = []
174         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "parse_html",
175              "params": params, "id": "A3s23"})
176         header = {"Content-type": "application/json"}
177
178         conn.request("POST", "", request, header)
179         response = conn.getresponse()
180         conn.close()
181
182         expectedResponse = '{"error": {"message": "Invalid parameters.",
                "code": -32602}, "jsonrpc": "2.0", "id": "A3s23"}'
183
184         self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
185
186     def test_json_rpc_unsupported_method(self):
187         """
188         Tests that the server responds with an error when
189         a client attempts to make a function call for a function
190         which hasn't been registered to the server.
191
192         Input: A JSON-RPC 2.0 request containing a function name
193         which hasn't been registered on the server.
194
195         Expected Result: The returned JSON-RPC 2.0 response
196         string should match the string expectedResponse, which
197         contains a response representing an unsupported method
198         error.
199         """
200         conn = httplib.HTTPConnection("127.0.0.1:8080")
201         fragment = base64.b64encode(b'<html></html>')
202         params = [{"files": [], "document": fragment, "filename": ""}]
203         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "not_registered",
204              "params": params, "id": "A3s23"})
205         header = {"Content-type": "application/json"}
```

```
206
207         conn.request("POST", "", request, header)
208         response = conn.getresponse()
209         conn.close()
210
211         expectedResponse = '{"error": {"message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
212
213         self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
214
215 if __name__ == '__main__':
216     unittest.main()
```