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 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.

The test cases we are running on the parser can be found in Appendix A within this document. This gives an indication of the tests we have currently implemented in the system. More tests are being added periodically, as different types of HTML5 errors are added to the parser. Each HTML5 error will have its own associated test. Other parser related tests are also contained in Appendix A, including the JSON-RPC server tests. These test for concurrency (can the parser handle 5 concurrent requests?) as well as correctness of JSON-RPC input and output.

Implications of Functional Testing

Our functional testing highlighted some issues with all aspects of our application. Primarily, there are several aspects of our functionality which are unimplemented at the moment but we hope to implement before the end of the project. However, we knew that these were issues. Instead, the interesting implications of the testing showed that we were mostly assured that we were programming our code correctly. We thought we had implemented an appropriate fix for the case of empty files (Website and Server Tests, 11) but our testing revealed this was an inadequate fix. Since we are still unsure of the implementation methods of each individual error check, many of the tests relating to errors are unimplemented. They will be periodically added as we introduce more error checks to the parser. From here, the next version of our prototype will aim to implement all the remaining functionality that our tests cover, and include tests on the remaining error parsing.

Test Case Transcript

Python Parser Tests

Test Number	Test Description	Inputs	Expected Output / Resulting Action	Pass / Fail + How to Fix
1	Testing that a specific error is being reported correctly, given a particular fragment of HTML as the input. Since not all of the error checks have been or can be defined in advance, the implementation of this test case is reactionary and will be continually updated to include new error checks as they are added. The existing tests will have to be run each time a new error check is implemented to ensure that all existing functionality still works as intended.	A tailored fragment of HTML that should cause a specific error to be reported. eg. ``<head> <head> </head> </head>" =>Testing for the duplicate set of head tags.	Confirmation that the expected error and associated error code is being returned for the given HTML fragment, in the expected character position of the input fragment.	Pass / Ongoing. The usage of this test case is currently being implemented after each error type is implemented. These tests are passing for the error checks we have implemented so far.
2	Test that the JSON-RPC server is running and can respond to a remote function call.	A single client making a function call to the server.	The function call should be processed without an error being raised, implying the server is currently running.	Pass
3	Testing that the JSON-RPC server is able to handle up to a maximum of 5 concurrent remote function calls.	Five concurrent function calls are made to the server.	The test case records the time that each response is received by each of the client instances. The function being called has an internal sleep delay of 2 seconds, so the recorded time for each client should be slightly over 2 seconds, implying all 5 calls were made and processed at the same time.	Pass
4	Testing that a 6th concurrent connection (1 connection over the maximum of 5 concurrent connections) to the JSON-RPC server results in a delayed response.	Six concurrent function calls are made to the server.	As above, the time the response is received is compared to the time the call was made. The first 5 connections should behave as above, receiving a response in just over 2 seconds. The 6th call will receive a response after 4 seconds, 2 seconds after the server is able to respond after the first 5 connections have been responded to.	Pass

5	Testing the parser's response to a case where input of an empty string is supplied.	An empty string.	The parser should respond with a general error stating that the input is empty, preventing other general errors such a missing closing HTML tags or page structure sections (head, body, footer).	Failed. This check will have to be performed at the start of the main parsing loop, preventing the general structural errors from being reported.
6	Testing the parser's response when the input string doesn't contain any valid HTML.	A garbage string which doesn't contain any HTML tags or tag like elements eg. ¡blah¿	The parser should respond with a general error stating that the input doesn't contain any valid HTML.	Failed. The parser will have to check the tokenizer for the existence of any valid tags. If none are found and the input string isn't empty, an error will be thrown stating that no valid HTML was found in the input.
7	Testing that a correctly formed JSON-RPC 2.0 request is handled by the server, which should respond with the correct response.	A JSON-RPC 2.0 request containing a small HTML code fragment to be parsed.	A JSON-RPC 2.0 response containing an array of errors related to the given request. The response should also contain the same ID value as the one passed to it with the request.	Pass
8	Testing that a malformed JSON-RPC 2.0 request containing incorrect parameters for a particular function call causes the server to return an error.	A JSON-RPC 2.0 request containing an invalid parameters array for the function call `parse_html`.	A JSON-RPC 2.0 error response with a message of "Invalid parameters".	Pass
9	Testing that a JSON-RPC 2.0 request calling a function that isn't registered on the server causes the server to send an error response.	A JSON-RPC 2.0 request containing a function name that hasn't been registered on the server.	A JSON-RPC 2.0 error response with a message of "".	Pass
10	Testing the parser response when a tag with a URL attribute is supplied with an valid relative file path.	Html fragment: ``<img src=``../image.jpg"><img src=``directory2/ image2.jpg">". File list: [``image.jpg", ``directory/", ``directory/current.html", ``directory/directory2/ image2.jpg"]. Current file: ``directory/current.html"	The parser response should NOT contain an error indicating an invalid file path.	Pass
11	Testing the parser response when a tag with a URL attribute is supplied with an non-existent relative file path.	Html fragment: ``<img src=../../image.jpg><img src=directory2/ image2.jpg>". File list: [``directory/", ``directory/current.html"]. Current file: ``directory/current.html"	The parser response should contain two errors indicating invalid file paths, associated with the src attributes of the img tags.	Pass

12	Testing the parser response when a tag with a URL attribute is supplied with an non-existent files in existing relative filepaths.	Html fragment: ‘‘<img src=../image.jpg><img src=directory2/ image2.jpg>". File list: [‘‘directory/", ‘‘directory/current.html", ‘‘directory/directory2/"]. Current file: ‘‘directory/current.html"	The parser response should contain two errors indicating invalid file paths, associated with the src attributes of the img tags.	Pass
13	Testing the parser response when a tag with a URL attribute is supplied with an valid absolute file path.	Html fragment: ‘‘<img src=/image.jpg><img src=/directory/directory2/ image2.jpg>". File list: [‘‘directory/", ‘‘/image.jpg", ‘‘directory/current.html", ‘‘directory/directory2/", ‘‘directory/directory2/ image2.jpg"]. Current file: ‘‘directory/current.html".	The parser response should NOT contain an error indicating an invalid file path.	Pass
14	Testing the parser response when a tag with a URL attribute is supplied with an non-existent absolute file path.	Html fragment: ‘‘<img src=/directory3/ image.jpg>". File list: [‘‘directory/", ‘‘directory/current.html"]. Current file: ‘‘directory/current.html"	The parser response should contain an error indicating invalid file paths, associated with the src attributes of the img tag.	Pass
15	Testing the parser response when a tag with a URL attribute is supplied with non-existent files in existing relative file paths.	Html fragment: ‘‘<img src=/directory/ image2.jpg>". File list: [‘‘directory/", ‘‘directory/current.html"]. Current file: ‘‘directory/current.html".	The parser response should contain one errors indicating invalid file paths, associated with the src attributes of the img tags.	Pass

Website and Server Tests

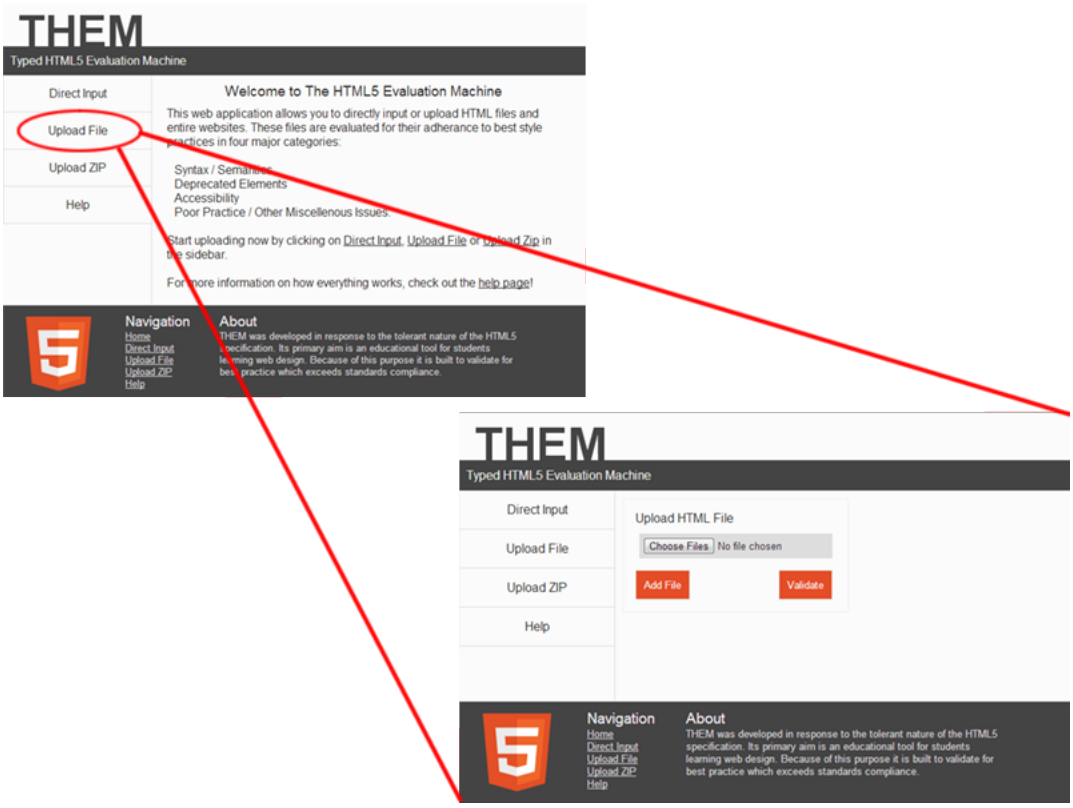
Test Number	Test Description	Inputs	Expected Output / Resulting Action	Pass / Fail + How to Fix
1	View Home page	Go to URL, or click link from any page	The home page is displayed.	Pass
2	View Help page	Click link from any page	The user is sent to the help page.	Pass
3	View Direct Input page	Click link from any page	The user is sent to the direct input page.	Pass
4	Validate direct input	The user types their input into the text field on the Direct Input page and clicks Validate.	The input text is saved in a new set with a single file in it. The user is redirected to the Show File page.	Pass
5	View Upload File(s) page	Click link from any page	The user is sent to the Upload File page.	Pass
6	Upload single HTML file	The user selects a file and then clicks Validate.	The file is saved in a new set with a single file in it. The user is redirected to the Show File page.	Pass
7	Upload multiple HTML files individually	The user clicks the Add File button the required number of times, then selects a file for each field. They then click Validate.	Files are saved in a new set, user is redirected to uploaded set page	Pass
8	Upload multiple HTML files together from one dialogue	The user selects multiple files in the dialogue box, then clicks Validate.	Files are saved in a new set, user is redirected to uploaded set page	Pass
9	Upload multiple HTML files, some individually and some from one dialogue box	The user performs a combination of multiple Add Files and selecting multiple files in the dialogue boxes. They then click validate.	Files are saved in a new set, user is redirected to uploaded set page	Pass
10	Upload non-HTML file	The user attempts to upload a file which is not HTML.	The user is redirected to the same page and shown a information box informing them that the file chosen is not a HTML file.	Failed. Currently it parses all uploaded files for HTML. Fix by checking mime type before sending to parser
11	No file selected on upload	The user attempts to upload a file when no file is selected.	Redirect to upload file page with a helpful error message	Failed. Currently an empty file is shown
12	View Upload Zip page	Click link from any page	The user is sent to the Upload Zip page.	Pass
13	Upload zip file	Zip file selected on previous page, user clicked validate	Zip archive is unpacked, files are saved in a new set, user is redirected to uploaded set page	Pass
14	View Uploaded Set page	User either uploads multiple files, or uploads a zip archive	The user is shown the list of files uploaded in this set, with corresponding error bars.	Pass, except in the case of a single file in a set or zip, in which case the user is redirected directly to the show file page

15	View Uploaded File page	User either selects a file on the Uploaded Set page, or uploads a single file, or validates by direct input	The user is shown their uploaded file, with corresponding error bar, general error information, and uploaded text with error highlighting.	Pass
16	Remove file after certain period of being untouched in the server	A file should be removed from the server after a period of inactivity.	The files are removed from the database after a time. (3 hours)	Failed. Currently, files stay in database. The cron job, which cleans up a file after it is left untouched for a time, is written and tested, but not installed on the development server
17	User attempts to access files they have not uploaded	The user manually attempts to alter the URL to view a file they have not uploaded.	The user is redirected to the home page.	Failed. Currently no security checking exists. The field exists in the data structure, but is not populated and validated against a php session id

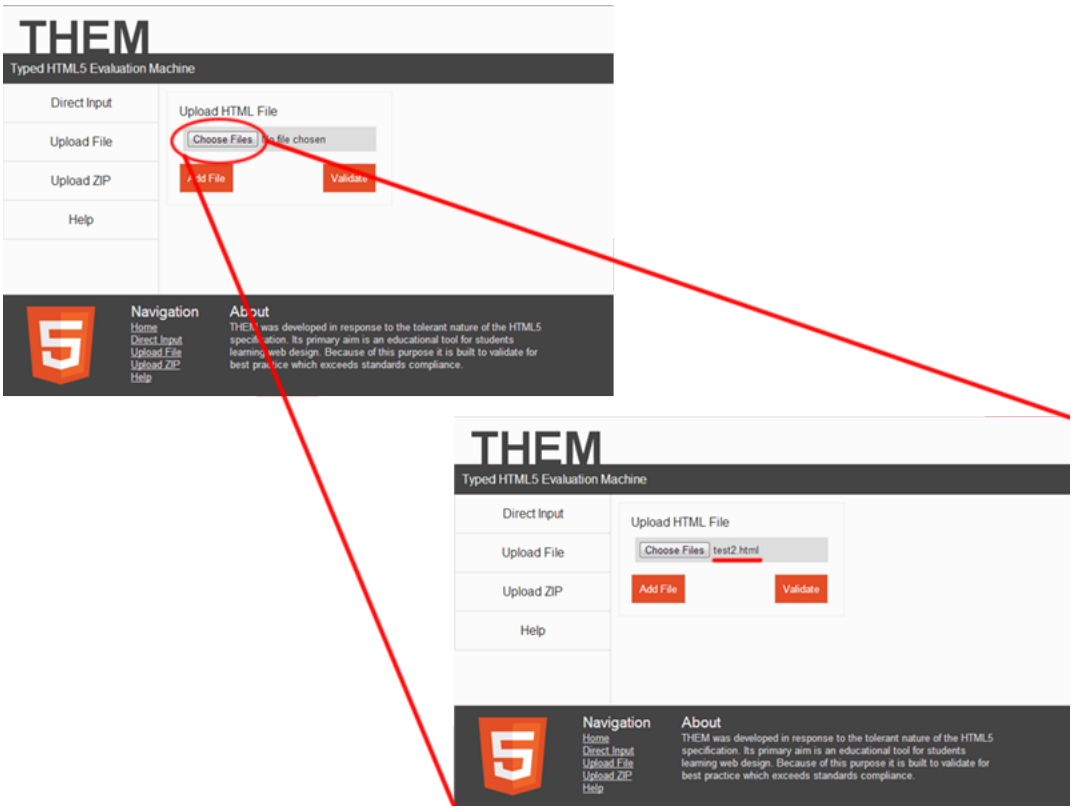
User Experience Goals

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, and passive experience. The tool should give users almost 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, and be held on the website for long periods at a time. 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 instead 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.

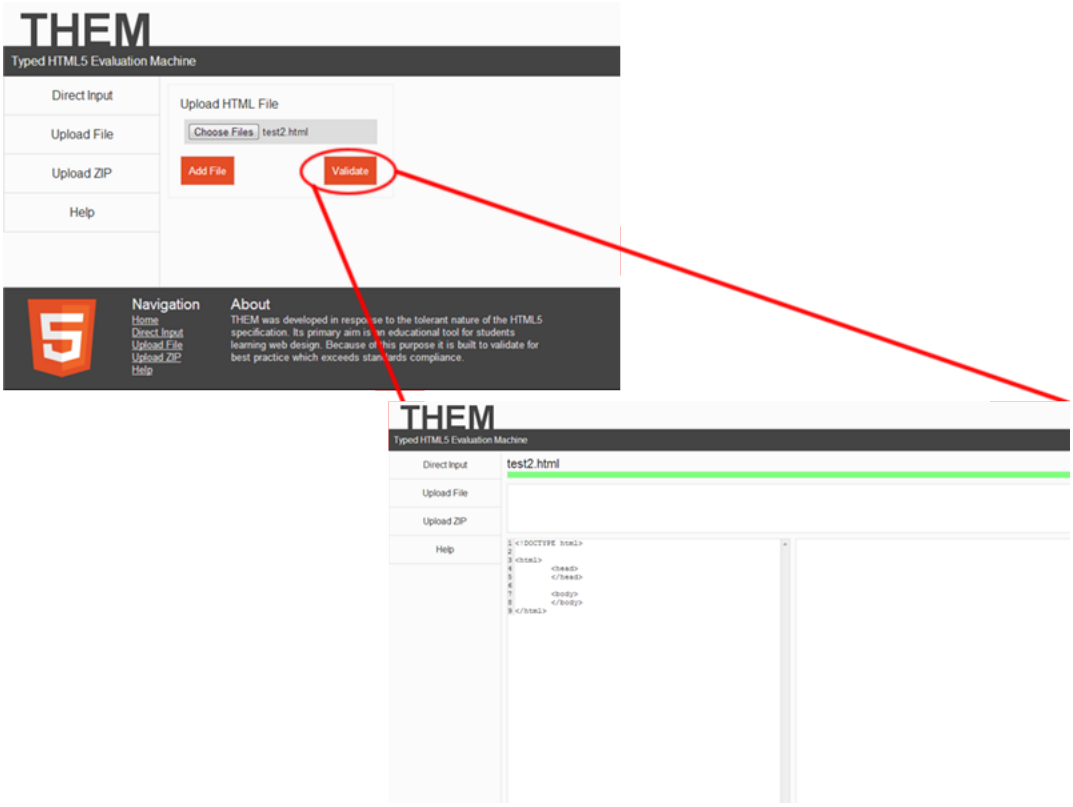
Our priorities are on quick and easy use, which is why all web pages are instantly accessible from all other web pages and users only need a few clicks to navigate. We have designed the website to require as few clicks as possible to access the primary functionality of the system. For example, the following images represent an average user's attempt to verify a file, after brief knowledge of the system's workings.



The first click takes the user to the webpage.



The second click chooses a file to verify. Other clicks may be employed here as the user navigates their file system to find the file they want to upload.



The third click validates the selected file. In this way, the user quickly and easily travels from the opening page to viewing their validated HTML. All throughout the web application, we have aimed to create similar experiences where few clicks are required by the user, and they reach their end goal in minimal time.

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; and
- 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 had not done DECO1400, as these users represented students "new" to DECO1400. Focusing also on past DECO1400 students would have allowed us to understand the needs of users who had previously completed 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 nine users - four were undergraduates who had not done DECO1400, one was an undergraduate who had done DECO1400 and four were masters students who had done DECO7140.

We formulated six key scenarios for our users to undertake. Each scenario was performed on the live prototype at http://underwaterfall.com. Storyboards for each of them can be found in the separately submitted Appendix B.

- Getting started by reading the help page (First Encounter Scenario)

Actor:	New User
Goal:	To understand how the website works, and understand the feedback it provides.
Necessity of Scenario:	This scenario is required for first time users to understand how to use and interact with the program.
Preconditions:	User has not previously visited the webpage.

- Validating HTML via Direct Input

Actor:	User
Goal:	To check the validity of a piece of copied or typed HTML.
Necessity of Scenario:	This scenario represents one of the key ways users can get insight into how to program using HTML5.
Preconditions:	User has a clear understanding of the validation the website provides from the help page.

- Validating HTML via uploading a file

Actor:	User
Goal:	To check the validity of a HTML file.
Necessity of Scenario:	This scenario represents one of the key ways users can get insight into how to program using HTML5.
Preconditions:	User has a clear understanding of the validation the website provides from the help page.

- Validating websites or multiple HTML files via uploading a zip
  - Actor: User
  - Goal: To check the validity of a zip file of either website files or HTML.
  - Necessity of Scenario: This scenario represents one of the key ways users can get insight into how to program using HTML5.
  - Preconditions: User has a clear understanding of the validation the website provides from the help page.
- Fixing a file based on the error suggestions, resubmitting and getting a valid file
  - Actor: User
  - Goal: To check the validity of a piece of copied or typed HTML.
  - Necessity of Scenario: This scenario is the primary point of the application - users learning to correct their HTML5 pages.
  - Preconditions: User has already uploaded a file and determined the errors relating to their webpage.
- Attempting to upload a non-HTML file (Fringe Case Scenario)
  - Actor: User
  - Goal: To check the validity of a non-HTML file.
  - Necessity of Scenario: Users are fallible and can upload incorrect files. They may also believe the website is capable of evaluating other types of files, like Javascript or CSS.
  - Preconditions: N/A

We focused on the metrics of time taken to complete each scenario, and, in keeping with our surgical user experience, number of clicks required to complete each scenario. As we also wanted to create an enjoyable environment for the users, we also made note of any particular emotions and reactions of the users as they undertook the scenarios. 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 the timer. Be prepared to count the number of clicks they required to complete the task. Take hand notes as the participant works, according to the arrangements you have 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?
6. When they complete the first scenario, move them onto the next one, and so on.
7. After completing all six scenarios, ask the participant to fill in the questionnaire. Clarify as necessary.

8. 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.

9. Tell the participant that the session is at an end. Thank the participant for their time.

You can see the results of testing below, after the "Implications of User Testing" section, as well as the Questionnaire we used for testing.

Implications of User Testing

In general, users had no trouble navigating the system. The average rating for how easy the system was to navigate was 4.78 / 5 on a Likert scale. We also clearly met our goal of the small number of clicks required for each operation. No user (from those who had click data registered) needed to click more than ten times. However, our system requires improvement in several key areas. On recommendation from the users, here is a key list of changes we plan to implement before the completion of this project:

- Invite the users to click on the error tags. Users often did not realise that they could click on the highlighted text to find out more about the error. We have already gone part way to completing this by changing the mouse from a text highlight symbol to a pointer finger commonly used with hyperlinks and buttons.

- Errors with non-HTML files. As part of our user testing, we asked users to upload a non-HTML file, a common action which could be performed by a user. Many were surprised that the file was actually parsed. We plan to check the mimetype of the uploaded file, and prevent the parser from being passed non-HTML code. The user will be presented with an error on the Upload File screen when they attempt to upload a non-HTML file. The good news is that for the most part, the website did not break, except when

- Add multiple files via Add File button. Users noted that an attempt to add a second file via the Add File button after selecting the first file caused their previously selected file to be forgotten. This needs to be fixed. One user had trouble due to the similarity of the Validate and Add File buttons, and we may look into making these two differ. However, as we have already implemented selecting multiple files from the file explorer itself, we may simply regress to this option, and discussion with our client may be necessary in this regard.

- Blank file uploading. Users choosing no file in a file field of the form, along with some file fields being filled, were parsed as if they were files with no name. As such, they were sent to a set page showing bars with blank file names and error bars. These files should not be parsed, and if a user clicks Validate with no file selected, they should be redirected back to the user page and asked to upload a file.

We do not plan on redefining any test plans, but for future testing, we will likely work closer together to complete it. The differences between our methods of testing was obvious, and it lead us to getting less data than we could have gotten.

All in all, we believe that our web application was well received, with many users who had experience with HTML5 stating they would find our tool beneficial to their studies. We receiving a rating 4.6 / 5 on the Likert scale for "How likely would you use this tool to assist you in your study?" among those students who had previous experience with HTML5. Although the tool still requires a lot of work to reach completion, it is well on the way to being a fantastic tool for users to evaluate their HTML5.

User Test Results

Metric Results

	Get Informed	Direct Input	Upload a File	Fixing a File	Upload Zip	Upload Non-HTML File
Tester 1 - DECO1400						
Time taken (secs)		49.03	39.9	32.7	13.3	6.03
Clicks						
Reaction		Confused about the highlighting word, unsure it is clickable	Feel confused when add file buton cancel previous upload entry	Learn the error quickly, getting used to the system presentation	Feeling comfortable	Feeling surprised when the result is as expected
Tester 2 - DECO7140						
Time taken (secs)		56	22.3	27	11.2	6.5
Clicks						
Reaction		Unsure about what the error bar representing	Frustrated as keep mispressing add file instead of validate due to same colour button	Learn the error quickly	Feeling comfortable	Feeling surprised when the result is as expected
Tester 3 - DECO7140						
Time taken (secs)		43.5	19.2	23.3	9.9	6.9
Clicks						
Reaction		Feeling unimpressed as the error is not validated correctly	Feeling satisfied with the simple way to upload file	Learn the error quickly and fixed it	Feeling comfortable	Feeling surprised when the result is as expected
Tester 4 - DECO7140						
Time taken (secs)		51.2	23.1	31.78	11.67	8.3
Clicks						
Reaction		Feeling that the presentation of errors is good	Frustrating when trying to upload multiple file, the add file button cancel previous entry	Learn the error quickly, feeling good	Feeling comfortable	Feeling surprised when the result is as expected
Tester 5 - DECO7140						
Time taken (secs)		48.12	17	20.1	10.8	5.87
Clicks						
Reaction		Unsure about the highlighting words are clickable	Feeling good as it is easy to upload single file	Learn the error quickly	Feeling comfortable	Feeling surprised when the result is as expected

Tester 6 - Undergraduate						
Time taken (secs)	7	11	24.8	10	8	N/A
Clicks:	1	2	6	5	4	5
Reaction		Didn't realise you could click on errors		Multiple files added but no files given - still shows the bars		Powerpoint file uploaded - "max_allowed_packets" error given
Tester 7 - Undergraduate						
Time taken (secs)	2	45	29	20	18	20
Clicks:	1	5	5	3	5	5
Reaction						Note, inf file still was parsed.
Tester 8 - Undergraduate						
Time taken (secs)	2	41	14.8	63	22	13
Clicks:	1	3	6	8	7	5
Reaction				Backtracked to copy files, didn't get to put in entire tag		
Tester 9 - Undergraduate						
Time taken (secs)	2	60	28	211	9	15
Clicks:	1	5	6	9	4	3
Reaction			Clicked add file accidentally	Not intuitive to click highlighted text		

Questionnaire

Tester	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Tester 1	Yes (DECO1400, Programming)	Yes (Java, Javascript, PHP, Python)	4	5	4	3	5	No	Yes	If you upload a non-zip file to the upload zip no error message is shown. Selecting file then clicking add file clears previous additions. Upload invalid file to upload file (specifically provided zip) shows the file. Adding ‘‘<!DOCTYPE html>." (note full stop) causes errors to appear (raw output). Clicking add file then only uploading 1 file takes you to collections screen instead of the single file screen. Leaving uploads blank causes blank error bars to appear.
Tester 2	Yes (DECO7140)	Yes (Python, Java, Actionscript)	3	2	4	5	4	No	Yes	
Tester 3	Yes (DECO7140)	No	4	4	4	4	4	Yes	Yes	
Tester 4	Yes (B InfTech)	Yes (Java, PHP, C♯, HTML5, CSS, Javascript, MYSQL, etc)	5	4	4	3	5	No	No	Confused about the “upload” buttons. Color theme is nice. Single file ->upload file, multiple files ->zip it and upload. Highlight the code in the correction section will be more user friendly (Upload File). General instruction on the UI can be added. Maybe do more market research about existing validation tools. Line mumber is good to be placed in direct input.
Tester 5	Yes (DECO7140)	Yes (Actionscript, Python)	3	5	4	5	5	No	No	Simple & clean layout, I would like if I could copy the text and paste the text again to modify it.
Tester 6	No	Yes (Python)	2	4	4	2	5	Yes	Yes	
Tester 7	No	Yes (Python, Java, Matlab)	2	4	4	3	5	Yes	No	No way to gauge the effects of the error based on the error message. Didn’t initially realise you can click on highlighted text to see error notes nor which colours associated to errors (thought colour was gauging error intensity.)
Tester 8	No	Yes (Python, Matlab)	3	5	4	4	5	Yes	No	All good :)
Tester 9	No (though did make website in primary school)	Yes (Python, Matlab)	1	4	4	5	5	Yes	Yes	It was fun.

User Test Document

DECO3801 User Testing Document

Name: _____

Program or Degree: _____

Please wait for your instructions from supervisors before completing these scenarios. Time will be taken between each scenario to write down key information.

Get Informed

Navigate to the help page. Since this is your first time using the software, read up on what the error bars mean here.

Direct Input

Copy some basic HTML text into the Direct Input page, and validate it for errors. This can be from a file you have locally, or you can use the following provided code:

```
<html>
<head></head>
<body></body>
</html>
```

Upload File

Upload a file to the Upload File page, and validate it for errors. If you do not have a file of your own, we can provide you with one.

Fixing a File

Based on the error messages provided, fix your uploaded file and resubmit it.

Upload Zip

Upload a zipped website to the Upload Zip page, and validate it for errors. If you do not have a website of your own, we can provide you with one.

Upload Non-HTML File

Try uploading any file you like. Does the application behave well?

Questionnaire

1. Have you taken DECO1400/DECO7140? If not, do you have any past learning experience in web design (such as HTML4, Javascript, etc).
   ☐ Yes ☐ No
   Past learning experience: _____

2. Do you have any programming background, if so what languages you have been using?
   ☐ Yes ☐ No
   Past learning experience: _____

3. How likely would you use this tool to assist you in your study?

Very likely				Very unlikely
5	4	3	2	1

4. How visually appealing is the website to you?

Very appealing				Not very appealing
5	4	3	2	1

5. How well did this tool meet your expectations?

Met my expectations very well				Did not meet my expectations
5	4	3	2	1

6. Do you feel comfortable with the presentation of the error message(s)?

Very comfortable				Not very comfortable
5	4	3	2	1

7. How easy did you find the tool to navigate?

Very easy				Very difficult
5	4	3	2	1

8. Is this your first time using HTML5 in developing a website?
   ☐ Yes ☐ No

9. Would you prefer the option to select multiple files at once (in the explorer window) when uploading?
   ☐ Yes ☐ No

10. If you have any general feedback/suggestions, feel free to use the space below to tell us.

Appendix A - 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
```

```
105     def test_max_concurrent_connections(self):
106         """
107         Tests that the server processes excess function calls after the
108         initial batch of calls.
109 
110         Input: 6 concurrent function calls to the server.
111 
112         Expected Result: The remote function, test_concurrency, contains
                a 2
113         second sleep call. The server has a maximum number of concurrent
114         connections of 5, so the 6th call will take slightly over 4
                seconds
115         to complete. The sum of the times for all 6 calls should be
                within
116         the range 15 > totalTime >= 14.
117         """
118 
119         self.resetCurrentTime()
120 
121         totalTime = 0
122 
123         for time in self.getExecutionTimes(6):
124             totalTime += time
125 
126         self.assertTrue(totalTime >= 14 and totalTime < 15, "Failed to "
                +
127             "execute all 6 concurrent function calls within the expected
                    " +
128             "time frame.")
129 
130     def test_json_rpc_correct_response(self):
131         """
132         Tests that the server responds as expected to a correctly
133         formed JSON-RPC 2.0 request.
134 
135         Input: A correctly formed JSON-RPC 2.0 request containing
136         an empty array of file names, an empty file name (direct
137         input method) and a HTML fragment consisting of '<html></html>'.
138 
139         Expected Result: The returned JSON-RPC 2.0 response string
140         should match the string expectedResponse, which contains
141         the expected array of errors.
142         """
143         conn = httplib.HTTPConnection("127.0.0.1:8080")
144         fragment = base64.b64encode(b'<html></html>')
145         params = [{"files": [], "document": fragment, "filename": ""}]
146         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "parse_html",
147             "params": params, "id": "A3s23"})
148         header = {"Content-type": "application/json"}
149 
150         conn.request("POST", "", request, header)
151         response = conn.getresponse()
152         conn.close()
153 
154         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     def test_invalid_filepath(self):
216         """
217         Tests that the server responds with an error when
218         a client attempts to make a function call for a function
219         which hasn't been registered to the server.
220
221         Input: A JSON-RPC 2.0 request containing a function name
222         which hasn't been registered on the server.
223
224         Expected Result: The returned JSON-RPC 2.0 response
225         string should match the string expectedResponse, which
226         contains a response representing an unsupported method
227         error.
228         """
229         conn = httplib.HTTPConnection("127.0.0.1:8080")
230         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
                directory2/image2.jpg>')
231         params = [{"files": ["image.jpg", "directory/", "directory/
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
232         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "not_registered",
233              "params": params, "id": "A3s23"})
234         header = {"Content-type": "application/json"}
235
236         conn.request("POST", "", request, header)
237         response = conn.getresponse()
238         conn.close()
239
240         expectedResponse = '{"error": {"message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
241
242         self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
243
244     def test_invalid_filepath(self):
245         """
246         Tests that the server responds with an error when
247         a client attempts to make a function call for a function
248         which hasn't been registered to the server.
249
250         Input: A JSON-RPC 2.0 request containing a function name
251         which hasn't been registered on the server.
252
```

```
253         Expected Result: The returned JSON-RPC 2.0 response
254         string should match the string expectedResponse, which
255         contains a response representing an unsupported method
256         error.
257         """
258         conn = httplib.HTTPConnection("127.0.0.1:8080")
259         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
                directory2/image2.jpg>')
260         params = [{"files": ["image.jpg", "directory/", "directory/
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
261         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "not_registered",
262          "params": params, "id": "A3s23"})
263         header = {"Content-type": "application/json"}
264 
265         conn.request("POST", "", request, header)
266         response = conn.getresponse()
267         conn.close()
268 
269         expectedResponse = '{"error": {"message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
270 
271         self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
272 
273     def test_invalid_filepath(self):
274         """
275         Tests that the server responds with an error when
276         a client attempts to make a function call for a function
277         which hasn't been registered to the server.
278 
279         Input: A JSON-RPC 2.0 request containing a function name
280         which hasn't been registered on the server.
281 
282         Expected Result: The returned JSON-RPC 2.0 response
283         string should match the string expectedResponse, which
284         contains a response representing an unsupported method
285         error.
286         """
287         conn = httplib.HTTPConnection("127.0.0.1:8080")
288         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
                directory2/image2.jpg>')
289         params = [{"files": ["image.jpg", "directory/", "directory/
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
290         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "not_registered",
291          "params": params, "id": "A3s23"})
292         header = {"Content-type": "application/json"}
293 
294         conn.request("POST", "", request, header)
295         response = conn.getresponse()
296         conn.close()
297 
```

```
298         expectedResponse = '{"error": {"message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
299
300         self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
301
302     def test_invalid_filepath(self):
303         """
304         Tests that the server responds with an error when
305         a client attempts to make a function call for a function
306         which hasn't been registered to the server.
307
308         Input: A JSON-RPC 2.0 request containing a function name
309         which hasn't been registered on the server.
310
311         Expected Result: The returned JSON-RPC 2.0 response
312         string should match the string expectedResponse, which
313         contains a response representing an unsupported method
314         error.
315         """
316         conn = httplib.HTTPConnection("127.0.0.1:8080")
317         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
                directory2/image2.jpg>')
318         params = [{"files": ["image.jpg", "directory/", "directory/
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
319         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "not_registered",
320             "params": params, "id": "A3s23"})
321         header = {"Content-type": "application/json"}
322
323         conn.request("POST", "", request, header)
324         response = conn.getresponse()
325         conn.close()
326
327         expectedResponse = '{"error": {"message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
328
329         self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
330
331     def test_invalid_filepath(self):
332         """
333         Tests that the server responds with an error when
334         a client attempts to make a function call for a function
335         which hasn't been registered to the server.
336
337         Input: A JSON-RPC 2.0 request containing a function name
338         which hasn't been registered on the server.
339
340         Expected Result: The returned JSON-RPC 2.0 response
341         string should match the string expectedResponse, which
342         contains a response representing an unsupported method
343         error.
344         """
```

```
345         conn = httplib.HTTPConnection("127.0.0.1:8080")
346         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
                directory2/image2.jpg>')
347         params = [{"files": ["image.jpg", "directory/", "directory/
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
348         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "not_registered",
349             "params": params, "id": "A3s23"})
350         header = {"Content-type": "application/json"}
351 
352         conn.request("POST", "", request, header)
353         response = conn.getresponse()
354         conn.close()
355 
356         expectedResponse = '{"error": {"message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
357 
358         self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
359 
360     def test_invalid_filepath(self):
361         """
362         Tests that the server responds with an error when
363         a client attempts to make a function call for a function
364         which hasn't been registered to the server.
365 
366         Input: A JSON-RPC 2.0 request containing a function name
367         which hasn't been registered on the server.
368 
369         Expected Result: The returned JSON-RPC 2.0 response
370         string should match the string expectedResponse, which
371         contains a response representing an unsupported method
372         error.
373         """
374         conn = httplib.HTTPConnection("127.0.0.1:8080")
375         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
                directory2/image2.jpg>')
376         params = [{"files": ["image.jpg", "directory/", "directory/
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
377         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "not_registered",
378             "params": params, "id": "A3s23"})
379         header = {"Content-type": "application/json"}
380 
381         conn.request("POST", "", request, header)
382         response = conn.getresponse()
383         conn.close()
384 
385         expectedResponse = '{"error": {"message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
386 
387         self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
```

```
388
389     def test_invalid_filepath(self):
390         """
391         Tests that the server responds with an error when
392         a client attempts to make a function call for a function
393         which hasn't been registered to the server.
394
395         Input: A JSON-RPC 2.0 request containing a function name
396         which hasn't been registered on the server.
397
398         Expected Result: The returned JSON-RPC 2.0 response
399         string should match the string expectedResponse, which
400         contains a response representing an unsupported method
401         error.
402         """
403         conn = httplib.HTTPConnection("127.0.0.1:8080")
404         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
                directory2/image2.jpg>')
405         params = [{"files": ["image.jpg", "directory/", "directory/
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
406         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "not_registered",
407             "params": params, "id": "A3s23"})
408         header = {"Content-type": "application/json"}
409
410         conn.request("POST", "", request, header)
411         response = conn.getresponse()
412         conn.close()
413
414         expectedResponse = '{"error": {"message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
415
416         self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
417
418 if __name__ == '__main__':
419     unittest.main()
```