DECO3801 Test Plan Document

THEM - Typed HTML5 Evaluation Machine

Carl Hattenfels, Scott Heiner, Shen Yong Lau, Robert Meyer, Brendan Miller, David Uebergang

Contents

1 Functional Test Plan

1.1 Testing Strategy

There are three major testable components of the 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 the front-end website and database with a suite of computer-run tests. Instead, a series of scenarios were drafted that would 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 that are being run on the parser can be found in Appendix A within this document. This gives an indication of the tests that are 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.

1.2 Implications of Functional Testing

The functional testing highlighted some issues with all aspects of the application during initial development. There were several sections of the functionality which were unimplemented initially that are now implemented. The interesting implications of the testing showed that the code was implemented correctly. An appropriate fix for the case of empty files (Website and Server Tests, 11) was implemented post-testing. Many of the tests relating to errors were unimplemented initially, but all are present in this final solution. The test suite that is now being used for this program is complete and thorough.

1.3 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
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).	Pass
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.	Pass
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

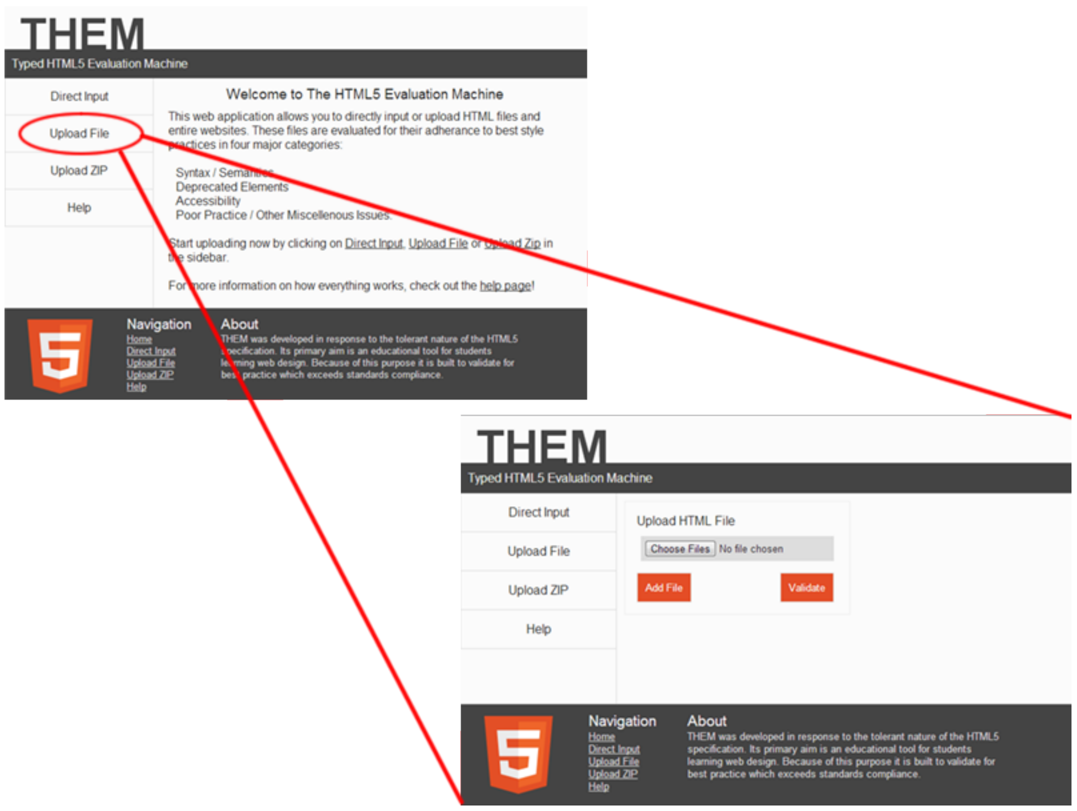
Test Number	Test Description	Inputs	Expected Output / Resulting Action	Pass / Fail + How to Fix
1	View Home page	Go to URL, or click link from any page	The home page is displayed.	Pass
2	View Help page	Click link from any page	The user is sent to the help page.	Pass
3	View Direct Input page	Click link 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.	Pass
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	Pass
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, 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.	Pass

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)	Pass

2 User Experience Goals

A clear user experience was in mind while developing this website. Through its ease of use and minimal effort on the part of the user, the application aims 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 the tool ends, for this session. The user now can go and fix their file externally, return to the program and almost instantly receive another assessment of their code's validity. There is no aim to get the user invested in the system, and be held on the website for long periods at a time. However, creating a reliable and worthwhile experience is a large focus of this project, brief as this experience with the user 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. The website has been designed 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.



Welcome to The HTML5 Evaluation Machine

This web application allows you to directly input or upload HTML files and entire websites. These files are evaluated for their adherance to best style practices in four major categories:

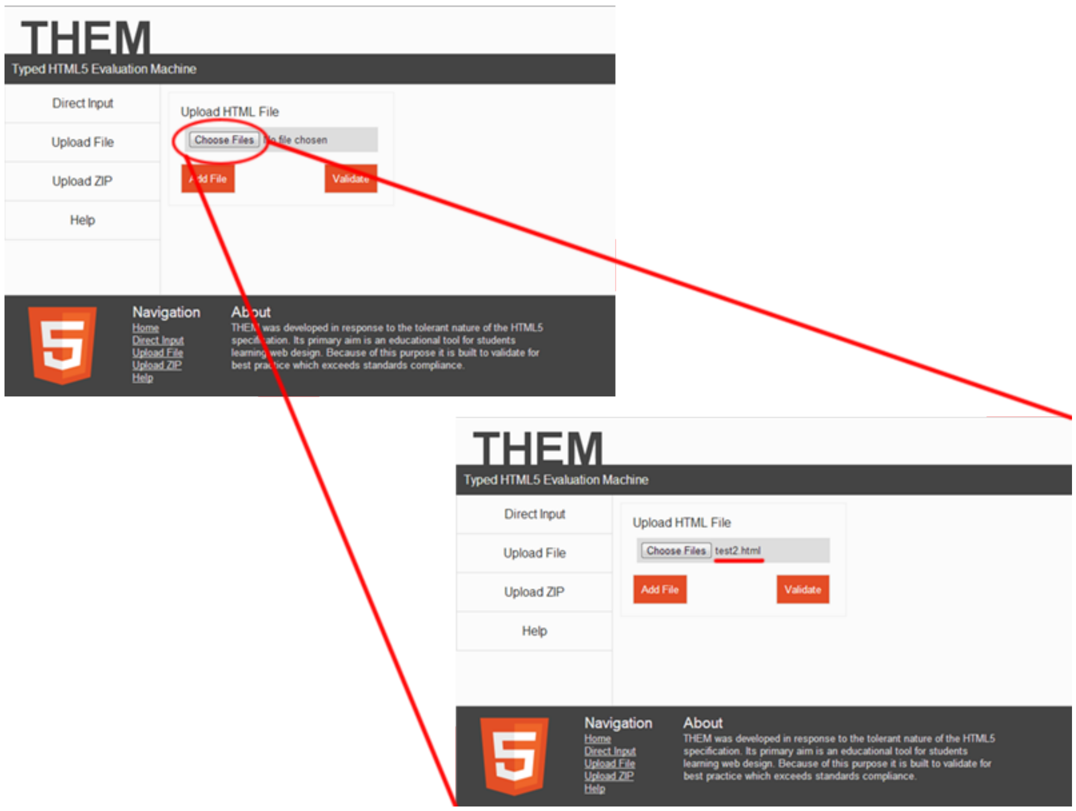
Syntax / Semantics
Deprecated Elements
Accessibility
Poor Practice / Other Miscellenous Issues

Start uploading now by clicking on Direct Input, Upload File or Upload Zip in the sidebar.

For more information on how everything works, check out the help page!

THEM was developed in response to the tolerant nature of the HTML5 specification. Its primary aim is an educational tool for students learning web design. Because of this purpose it is built to validate for best practice which exceeds standards compliance.

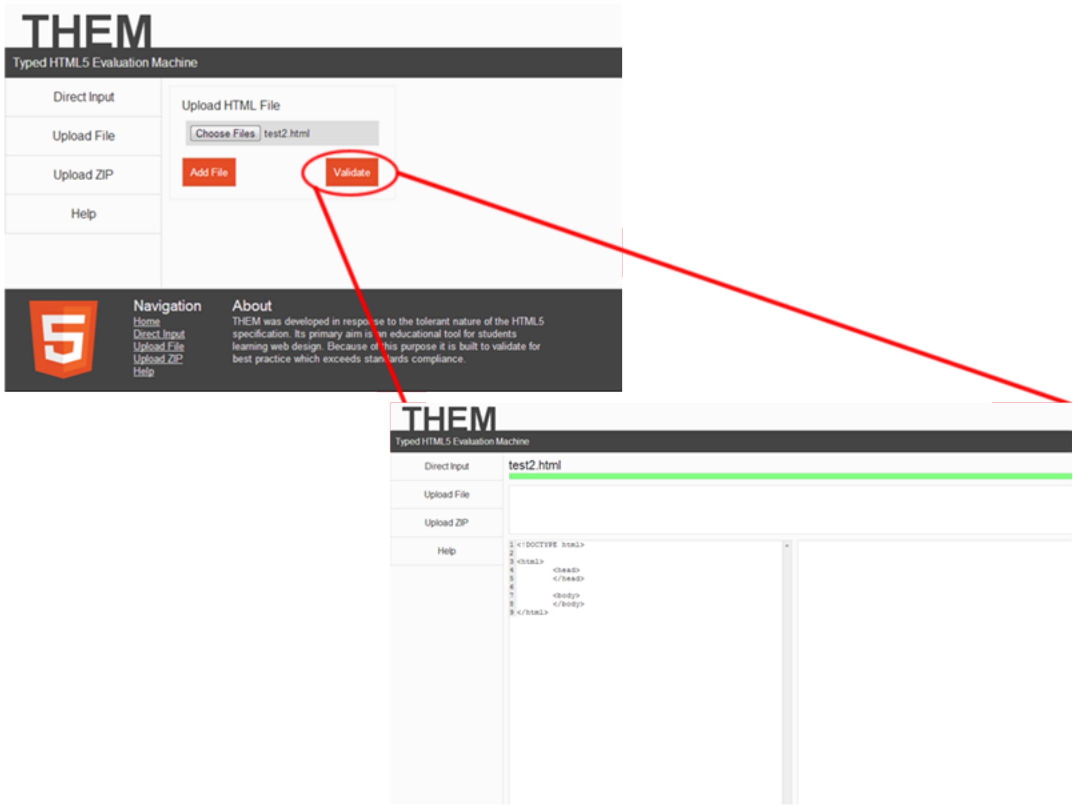
The first click takes the user to the webpage.



THEM was developed in response to the tolerant nature of the HTML5 specification. Its primary aim is an educational tool for students learning web design. Because of this purpose it is built to validate for best practice which exceeds standards compliance.

THEM was developed in response to the tolerant nature of the HTML5 specification. Its primary aim is an educational tool for students learning web design. Because of this purpose it is built to validate for best practice which exceeds standards compliance.

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.



THEM was developed in response to the tolerant nature of the HTML5 specification. Its primary aim is an educational tool for students learning web design. Because of this purpose it is built to validate for best practice which exceeds standards compliance.

```
<!DOCTYPE html>

<html>
        <head>
        </head>

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

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, the aim has been to create similar experiences where few clicks are required by the user, and they reach their end goal in minimal time.

Users benefit greatly from this experience. The surgical nature means that they develop a relationship with the website where it is used as an intermediate and reliable tool, much like their text editor or their browser. It is easily inserted as part of their workflow. It is hoped that by using this application, users will experience a great rush of joy when their program finally passes, and that long error bar on the web page turns completely green. Validation, after all, is its own reward.

3 User Testing Plan

3.1 User Testing Strategy

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

- 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 the part led to us getting very few in this category. Ultimately, information was gathered 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.

Six key scenarios were formulated for the 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

There was a focus in testing of two metrics: time taken to complete each scenario, and, in keeping with the surgical user experience, number of clicks required to complete each scenario. As there was a need to create an enjoyable environment for the users, any particular emotions and reactions of the users as they undertook the scenarios was also noted. the 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 starting, 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.

The results of testing are below, after the "Implications of User Testing" section, as well as the Questionnaire used for testing.

3.2 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. The user experience goal of the small number of clicks required for each operation was met. No user (from those who had click data registered) needed to click more than ten times. However, the system requires improvement in several key areas. On recommendation from the users, here is a key list of changes that were implemented in 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. Users are now provided with information on the help page about this, as well as told on the file page to "Click on highlighted text for more information."
- Errors with non-HTML files. As part of the user testing, users were asked 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. The mimetype of the uploaded file is now checked, and the parser is not passed non-HTML code, except in the case of code which is plain-text and could possibly be HTML code that lacks structure. The user is 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 a user attempted to upload a Powerpoint file. This is prevented by the previously mentioned fix.
- 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, so these now differ in colour to highlight the difference functions.
- 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 are no longer parsed, and if a user clicks Validate with no file selected, they are redirected back to the upload file page and asked to upload a file.

There was no plan to redefine any test plans, but for future testing, working closer together to complete it would be a priority. The differences between the methods of testing used by members of the group was obvious, and it lead to less data than expected. All in all, the web application was well received, with many users who had experience with HTML5 stating they would find the tool beneficial to their studies. It received a rating of 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. The Typed HTML5 Evaluation tool is a fantastic tool for users to evaluate their HTML5.

3.3 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.

3.4 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.

4 Appendix A - Python Test Code

4.1 Syntax Tests

```python
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.")

    def test_self_closing_end_tag(self):
        """
        Test that a closing tag with a misplaced forwardslash
        raises an error.

        Input:
        A HTML fragment containing a closing tag with a misplaced
            forwardslash.

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

        inputFragment = "<html><a></a /></html>"

        self.parser.parse(inputFragment)

        self.assertIn(((9, 14), u'self-closing-flag-on-end-tag', {}),
            self.parser.errors, "Failed to report misplaced forwardslash
                in closing tag.")

    def test_invalid_self_closing_tag(self):
        """
        Test that the use of a self closing tag for a tag
        which isn't considered a self closing tag returns
        an error.

        Input:
        A HTML fragment containing a start tag with a trailing
            forwardslash
        (self-closing) for a tag type which isn't a self closing tag.

        Expected Results:
        An error should be thrown reporting the given tag type isn't a
            self-closing
        tag.
        """

        inputFragment = "<html><a /></html>"

        self.parser.parse(inputFragment)

        self.assertIn(((6, 10), u'non-void-element-with-trailing-solidus'
            , {u'name': u'a'}),
            self.parser.errors, "Failed to report invalid self-closing
                tag.")

    def test_attributes_in_end_tag(self):
        """
        Test that attributes occuring in a closing tag are
        reported as an error.

        Input:
        A HTML fragment containing a closing tag which contains
        at least one attribute.

        Expected Results:
        An error should be thrown reporting that the closing tag shouldn't contain
        attributes.
        """

        inputFragment = '<html><a></a src="blah"></html>'

        self.parser.parse(inputFragment)

        self.assertIn(((9, 23), u'attributes-in-end-tag', {}),
            self.parser.errors, "Failed to report attributes in closing tag.")

    def test_duplicate_h1_tags(self):
        """
        Test that any duplicate h1 tags are reported as errors.

        Input:
        A HTML fragment containing more than one set of h1 tags.

        Expected Results:
        An error should be thrown reporting that duplicate h1 tags were found in the document.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1></h1>
<h1></h1>
<footer>
</footer>
</body>
</html>
        """

        self.parser.parse(inputFragment)

        self.assertIn(((56, 59), u'duplicate-h1-element', {u'name': u'h1'}),
            self.parser.errors, "Failed to report duplicate h1 tags.")

    def test_heading_order(self):
        """
        Test that heading elements maintain correct order within the document. The
        order follows from h1-h6 and they must appear in that order during the document.

        Input:
        A HTML fragment containing a set of h2 tags appearing before a set of h1 tags.

        Expected Results:
        An error should be thrown reporting that the h2 tags are out of order.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h2></h2>
<h1></h1>
<footer>
</footer>
</body>
</html>
        """

        self.parser.parse(inputFragment)

        self.assertIn(((46, 49), u'invalid-heading-order', {u'name': u'h2', u'missing': u'h1'}),
            self.parser.errors, "Failed to report heading tags out of order.")

if __name__ == '__main__':
    unittest.main()
```

4.2 Page Structure Tests

```python
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.")

        self.assertIn(((25, 31), u'incorrect-placement-singular-end-tag',
            {u'name': u'head'}),
            self.parser.errors, "Multiple instances of closing head tag
                not reported.")

        self.parser.reset()
        self.parser.parse(multipleBodyInstances)

        self.assertIn(((25, 30), u'multiple-instance-singular-tag', {u'
            name': u'body'}),
            self.parser.errors, "Multiple instances of starting HTML tag
                not reported.")

        self.assertIn(((38, 44), u'unexpected-end-tag-after-body', {u'
            name': u'body'}),
            self.parser.errors, "Multiple instances of closing body tag
                not reported.")

    def test_missing_doctype(self):
        """
        Test that the expected-doctype-but-got-start-tag error is thrown
        for cases where no DOCTYPE is declared.

        Input:
        Nested blocks of singular tags (html, body, head), all of which
        are missing the DOCTYPE declaration.
        eg. <html><html></html></html>

        Expected Results:
        All test cases should report a missing DOCTYPE declaration.
        """
        startTagBeforeDoctype = "<html><html></html></html>"
        endTagBeforeDoctype = "</head></head>"
        eofBeforeDoctype = ""

        self.parser.parse(startTagBeforeDoctype)

        self.assertIn(((0, 5), u'expected-doctype-but-got-start-tag', {u'
            name': u'html'}),
            self.parser.errors, "Failed to report missing DOCTYPE
                declaration (start tag before doctype.")

        self.parser.reset()
        self.parser.parse(endTagBeforeDoctype)

        self.assertIn(((0, 6), u'expected-doctype-but-got-end-tag', {u'
            name': u'head'}),
            self.parser.errors, "Failed to report missing DOCTYPE
                declaration (closing tag before doctype.")

        self.parser.reset()
        self.parser.parse(eofBeforeDoctype)

        self.assertIn(((-1, -1), u'expected-doctype-but-got-eof', {}),
            self.parser.errors, "Failed to report missing DOCTYPE
                declaration (EOF before doctype.")


    def test_closing_html(self):
        """
        Test that a missing HTML closing tag is reported when none
        are present in the document.

        Input:
        Nested blocks of singular tags (head, body).

        Expected Results:
        Report whether the the closing HTML tag is present.
        """
        multipleHeadInstances = "<head><head></head></head>"
        multipleBodyInstances = "<body><body></body></body>"

        self.parser.parse(multipleHeadInstances);

        self.assertIn(((-1, -1), u'no-closing-html-tag', {}),
            self.parser.errors, "Failed to report missing closing HTML tag.")

        self.parser.reset()
        self.parser.parse(multipleBodyInstances)

        self.assertIn(((-1, -1), u'no-closing-html-tag', {}),
            self.parser.errors, "Failed to report missing closing HTML tag.")

    def test_misplaced_tags_before_head(self):
        """
        Test that both start and closing tags occurring before the head
        section are reported as being misplaced.

        Input:
        A number of instances of start and closing tags being placed before
        the head section.

        Expected Results:
        Report whether or not the tags preceding the head section are reported
        as being misplaced.
        """
        misplacedHeadTags = "<html><body></body><head></head></html>"
        misplacedLinkTags = "<html><a></a><head></head><body></body></html>"

        self.parser.parse(misplacedHeadTags)

        self.assertIn(((6, 11), u'incorrect-start-tag-placement-before-head', {u'name': u'body'}),
            self.parser.errors, "Failed to report start body tag before head section.")

        self.assertIn(((12, 18), u'incorrect-end-tag-placement-before-head', {u'name': u'body'}),
            self.parser.errors, "Failed to report closing body tag before
                head section.")

        self.parser.reset()
        self.parser.parse(misplacedLinkTags)

        self.assertIn(((6, 8), u'incorrect-start-tag-placement-before-
            head', {u'name': u'a'}),
            self.parser.errors, "Failed to report start link (a) tag
                before head section.")

        self.assertIn(((9, 12), u'incorrect-end-tag-placement-before-head
            ', {u'name': u'a'}),
            self.parser.errors, "Failed to report closing link (a) tag
                before head section.")

    def test_incorrect_tags_in_head(self):
        """
        Test that tags which don't belong in the head section
        are reported as misplaced using the 'incorrect-start-tag-
            placement-in-head'
        and 'incorrect-end-tag-placement-in-head' errors.

        Input:
        A HTML fragment with a pair of head tags enclosing a tag
        pair which doesn't belong in the head phase.

        Expected Results:
        Inclusion of the 'incorrect-start-tag-placement-in-head'
        and 'incorrect-end-tag-placement-in-head' errors being reported
        as part of the returned array of error codes.
        """
        inputFragment = "<html><head><a></a></head></html>"

        self.parser.parse(inputFragment)

        self.assertIn(((12, 14), u'incorrect-start-tag-placement-in-head'
            , {u'name': u'a'}),
            self.parser.errors, "Failed to report starting tag which
                doesn't belong in the head section.")

    def test_tags_after_eof(self):
        """
        Tests that starting and closing tags occurring after the last
        instace of a closing HTML tag are reported as an error.

        Input:
        A HTML fragment with a start and closing tag pair occurring
        after the start and closing HTML pair.

        Expected Results:
        An error being thrown for both the start and closing tags
            occurring
        after the HTML tags.
        """

        inputFragment = "<html></html><a></a>"
```

```python

        self.parser.parse(inputFragment)

        self.assertIn(((13, 15), u'expected-eof-but-got-start-tag', {u'name': u'a'}),
            self.parser.errors, "Failed to report start tag after closing HTML tag.")

        self.assertIn(((16, 19), u'expected-eof-but-got-end-tag', {u'name': u'a'}),
            self.parser.errors, "Failed to report closing tag after closing HTML tag.")

    def test_missing_start_tag(self):
        """
        Tests that a missing start tag is reported in the case
        that a closing tag is found without a matching start tag.

        Input:
        A HTML fragment containing a closing tag without a matching
        start tag.

        Expected Results:
        An error being thrown reporting that the matching start tag
        is missing.
        """

        inputFragment = "<html><head></head><body></a></body></html>"

        self.parser.parse(inputFragment)

        self.assertIn(((25, 28), u'unexpected-end-tag', {u'name': u'a'}),
            self.parser.errors, "Failed to report the lack of a matching start tag.")

    def test_misplaced_tags_after_body(self):
        """
        Tests that any tags occurring after the body phase
        are reported as being incorrectly placed.

        Input:
        A HTML fragment with a pair of start and closing tags placed
        after the closing body tag.

        Expected Results:
        An error should be thrown for both the start and closing
        tags found after the closing body tag.
        """

        inputFragment = "<html><head></head><body></body><a></a></html>"

        self.parser.parse(inputFragment)

        self.assertIn(((32, 34), u'unexpected-start-tag-after-body', {u'name': u'a'}),
            self.parser.errors, "Failed to report misplaced starting tag found after the closing body tag.")

        self.assertIn(((35, 38), u'unexpected-end-tag-after-body', {u'name': u'a'}),
            self.parser.errors, "Failed to report misplaced closing tag found after the closing body tag.")

    def test_missing_closing_html_tag(self):
        """
        Test that a missing closing HTML tag is reported.

        Input:
        A HTML fragment missing a closing HTML tag.

        Expected Results:
        An error should be thrown stating that the closing HTML tag is missing.
        """

        inputFragment = "<html><head></head><body></body>"

        self.parser.parse(inputFragment)

        self.assertIn(((-1, -1), u'no-closing-html-tag', {}),
            self.parser.errors, "Failed to report missing closing HTML tag.")

    def test_early_termination_before_head(self):
        """
        Test that an early closing HTML tag before the head phase
        is reported as an error.

        Input:
        A HTML fragment with the head and body sections placed after
        a closed set of HTML tags.

        Expected Results:
        An error should be thrown stating that the closing HTML tag
        has been found before the head phase.
        """

        inputFragment = "<html></html><head></head><body></body>"

        self.parser.parse(inputFragment)

        self.assertIn(((6, 12), u'early-termination-before-head', {u'name': u'html'}),
            self.parser.errors, "Failed to report early termination before head section.")

    def test_early_termination_in_head(self):
        """
        Test that an early closing HTML tag in the head phase
        is reported as an error.

        Input:
        A HTML fragment with the closing HTML tag placed within
        the set of head tags.

        Expected Results:
        An error should be thrown stating that the closing HTML tag
        has been found in the head phase.
        """

        inputFragment = "<html><head></html></head><body></body>"

        self.parser.parse(inputFragment)

        self.assertIn(((12, 18), u'early-termination-in-head', {u'name':
            u'html'}),
             self.parser.errors, "Failed to report early termination
                 before head section.")

    def test_early_termination_before_body(self):
        """
        Test that an early closing HTML tag before the body phase
        is reported as an error.

        Input:
        A HTML fragment with the closing HTML tag placed before the body
        section.

        Expected Results:
        An error should be thrown stating that the closing HTML tag
        has been found before the body phase.
        """

        inputFragment = "<html><head></head></html><body></body>"

        self.parser.parse(inputFragment)

        self.assertIn(((19, 25), u'early-termination-before-body', {u'
            name': u'html'}),
             self.parser.errors, "Failed to report early termination
                 before head section.")

    def test_early_termination_in_body(self):
        """
        Test that an early closing HTML tag in the body phase
        is reported as an error.

        Input:
        A HTML fragment with the closing HTML tag placed within
        the set of body tags.

        Expected Results:
        An error should be thrown stating that the closing HTML tag
        has been found in the head phase.
        """

        inputFragment = "<html><head></head><body></html></body>"

        self.parser.parse(inputFragment)

        self.assertIn(((25, 31), u'early-termination-in-body', {u'name':
            u'html'}),
            self.parser.errors, "Failed to report early termination
                before head section.")

    def test_tags_between_head_body(self):
        """
        Test that a set of tags placed after the head section
        but before the body section is reported as an error.

        Input:
        A HTML fragment with a set of tags between the head
        and body sections.

        Expected Results:
        An error should be thrown stating that the set of tags
        can't be placed between the head and body sections.
        """

        inputFragment = "<html><head></head><a></a><body></body></html>"

        self.parser.parse(inputFragment)

        self.assertIn(((19, 21), u'start-tag-before-body-after-head', {u'
            name': u'a'}),
            self.parser.errors, "Failed to report start tag after head
                phase but before body phase.")

        self.assertIn(((22, 25), u'end-tag-before-body-after-head', {u'
            name': u'a'}),
            self.parser.errors, "Failed to report closing tag after head
                phase but before body phase.")

    def test_missing_starting_html_tag(self):
        """
        Test that a missing starting HTML tag is reported as an error.

        Input:
        A HTML fragment missing a starting HTML tag.

        Expected Results:
        An error should be thrown indicating that the fragment doesn't
        contain a starting HTML tag.
        """

        inputFragment = "<head></head><body></body></html>"

        self.parser.parse(inputFragment)

        self.assertIn(((-1, -1), u'no-starting-html-tag', {}),
            self.parser.errors, "Failed to report missing starting HTML
                tag.")

    def test_unknown_doctype(self):
        """
        Test that a doctype with an invalid name is reported as being
        an unknown doctype.

        Input:
        A HTML fragment containing an invalid doctype name.

        Expected Results:
        An error should be thrown reporting that the doctype name is invalid.
        """

        inputFragment = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">'

        self.parser.parse(inputFragment)

        self.assertIn(((0, 89), u'unknown-doctype', {}),
            self.parser.errors, "Failed to report unknown doctype.")

    def test_space_after_doctype(self):
        """
        Test that a doctype tag has a space between the doctype declaration
        and the doctype name.

        Input:
        A HTML fragment containing a doctype with no space between the doctype
        declaration and the doctype name.

        Expected Results:
        An error should be thrown reporting that there is no space between
        the doctype declaration and the doctype name.
        """

        inputFragment = '<!DOCTYPEhtml>'

        self.parser.parse(inputFragment)

        self.assertIn(((0, 8), u'need-space-after-doctype', {}),
            self.parser.errors, "Failed to report missing space after the doctype declaration.")

    def test_end_tag_before_doctype(self):
        """
        Test that a closing tag isn't placed before the doctype declaration.

        Input:
        A HTML fragment containing a single closing tag.

        Expected Results:
        An error should be thrown reporting that a closing tag has been
        placed before the doctype declaration.
        """

        inputFragment = '</html>'

```

```python
        self.parser.parse(inputFragment)

        self.assertIn(((0, 6), u'expected-doctype-but-got-end-tag', {u'name': u'html'}),
            self.parser.errors, "Failed to report closing tag before doctype declaration.")

    def test_EOF_before_doctype(self):
        """
        Test that an error is reported if the document is blank.

        Input:
        A blank document containing no characters of any kind.

        Expected Results:
        An error should be thrown reporting that the EOF was reached before
        a doctype was declared.
        """

        inputFragment = """

        """

        self.parser.parse(inputFragment)

        self.assertIn(((-1, -1), u'expected-doctype-but-got-eof', {}),
            self.parser.errors, "Failed to report the EOF occurring before the doctype declaration.")

    def test_form_element_not_in_form(self):
        """
        Test that form elements must be contained in wrapping form tags.

        Input:
        A HTML fragment containing form elements which aren't wrapped in
        form tags.

        Expected Results:
        An error should be thrown reporting that the form elements aren't contained
        in wrapping form tags.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<datalist></datalist>
<fieldset></fieldset>
<input></input>
<label></label>
<output></output>
</body>
</html>
```

```
        """

        self.parser.parse(inputFragment)

        self.assertIn(((46, 55), u'form-element-not-in-form', {u'name': u
            'datalist'}),
            self.parser.errors, "Failed to report datalist tag outside of
                 wrapping form tags.")

        self.assertIn(((68, 77), u'form-element-not-in-form', {u'name': u
            'fieldset'}),
            self.parser.errors, "Failed to report fieldset tag outside of
                 wrapping form tags.")

        self.assertIn(((90, 96), u'form-element-not-in-form', {u'name': u
            'input'}),
            self.parser.errors, "Failed to report input tag outside of
               wrapping form tags.")

        self.assertIn(((106, 112), u'form-element-not-in-form', {u'name':
             u'label'}),
            self.parser.errors, "Failed to report label tag outside of
               wrapping form tags.")

        self.assertIn(((122, 129), u'form-element-not-in-form', {u'name':
             u'output'}),
            self.parser.errors, "Failed to report output tag outside of
               wrapping form tag.")

    def test_duplicate_id_value(self):
        """
        Test that the occurrence of duplicate id values is reported as an
             error.

        Input:
        A HTML fragment containing 2 elements with the same value for
            their id attribute.

        Expected Results:
        An error should be thrown reporting that an id attribute has the
            same value of a previously declared id
        attribute.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p id="blah"></p>
<p id="blah"></p>
</body>
</html>
        """

        self.parser.parse(inputFragment)

        self.assertIn(((64, 76), u'duplicate-id-attribute', {u'name': u'p', u'original': u'p'}),
            self.parser.errors, "Failed to report duplicate id value usage.")

    def test_duplicate_page_title(self):
        """
        Test that a duplicate title element is reported as an error.

        Input:
        A HTML fragment containing duplicate title elements.

        Expected Results:
        An error should be thrown reporting that a duplicate instance of the page
        title has been found.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
<title>blah</title>
<title>blah</title>
</head>
<body>
</body>
</html>
        """

        self.parser.parse(inputFragment)

        self.assertIn(((51, 57), u'duplicate-title-in-head', {u'name': u'title'}),
            self.parser.errors, "Failed to report duplicate title element.")

    def test_missing_title_element(self):
        """
        Test that a missing title element as part of the head section
        is reported as missing.

        Input:
        A HTML fragment containing a basic page structure but missing the required
        title element in the head section.

        Expected Results:
        An error should be thrown reporting that the title element is missing from
        the head sectionself.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
```

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

        self.parser.parse(inputFragment)

        self.assertIn(((-1, -1), u'title-element-missing-from-head', {}),
            self.parser.errors, "Failed to report missing title element.")

    def test_img_missing_alt_attribute(self):
        """
        Test that an img tag missing the required alt attribute is reported
        as an error.

        Input:
        A HTML fragment containing an img tag missing the required alt attribute.

        Expected Results:
        An error should be thrown reporting that the alt attribute is missing for the
        given img tag.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<img>
</body>
</html>
        """

        self.parser.parse(inputFragment)

        self.assertIn(((46, 50), u'img-element-missing-alt-attribute', {u'name': u'img'}),
            self.parser.errors, "Failed to report missing alt attribute for the given img tag.")

    def test_img_alt_attribute_empty(self):
        """
        Test that an img tag's alt attribute, when empty, is reported as an error.

        Input:
        A HTML fragment containing an img tag with an empty alt attribute.

        Expected Results:
```

```
        An error should be thrown reporting that the alt attribute is empty.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<img alt="">
</body>
</html>
        """

        self.parser.parse(inputFragment)

        self.assertIn(((46, 57), u'img-alt-attribute-empty', {u'attr': u''}),
            self.parser.errors, "Failed to report empty alt attriubte for img tag.")

    def test_missing_closing_tag_before_footer(self):
        """
        Test that any open tags (missing the closing tag) are reported if the
        footer section occurs before closing tag.

        Input:
        A HTML fragment containing an open 'a' tag which is missing the closing tag,
        followed by the footer section.

        Expected Results:
        An error should be thrown reporting that the closing tag wasn't found before
        the footer section.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<a>
<footer>
</footer>
</body>
</html>
        """

        self.parser.parse(inputFragment)

        self.assertIn(((50, 57), u'missing-end-tag-before-footer', {u'name': u'a'}),
            self.parser.errors, "Failed to report missing end tag before footer section.")

    def test_missing_closing_tags_footer_section(self):
        """
        Test that missing closing tags in the footer section are reported.

        Input:
        A HTML fragment containing an 'a' tag with a missing closing tag.

        Expected Results:
        An error should be thrown reporting that the closing tag is missing within the footer
        section.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<footer>
<a>
</footer>
</body>
</html>
        """

        self.parser.parse(inputFragment)

        self.assertIn(((59, 67), u'missing-closing-tags-in-footer', {u'name': u'a'}),
            self.parser.errors, "Failed to report missing closing tag in footer section.")

    def test_invalid_tag_name(self):
        """
        Test that tags with invalid tag names are reported as errors.

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

        Expected Results:
        An error should be thrown reporting that the tag name is invalid.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<blah></blah>
<footer>
</footer>
```

```
</body>
</html>
        """

        self.parser.parse(inputFragment)

        self.assertIn(((46, 51), u'invalid-element-name', {u'name': u'
            blah'}),
            self.parser.errors, "Failed to report invalid tag name.")

    def test_missing_closing_tag(self):
        """
        Test that any missing closing tags are reported as errors.

        Input:
        A HTML fragment containing an opening 'a' tag with a missing
            closing tag.

        Expected Results:
        An error should be thrown reporting that the closing tag is
            missing.
        """

        inputFragment = """
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<a>
<footer>
</footer>
</body>
</html>
        """

        self.parser.parse(inputFragment)

        self.assertIn(((46, 48), u'missing-end-tag', {u'name': u'a'}),
            self.parser.errors, "Failed to report missing closing tag.")

if __name__ == '__main__':
    unittest.main()
```

4.3 JSON-RPC Server Tests

```python
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)
        for i in range(numProcesses):
            results.append(pool.apply_async(getTime, (self.startTime,)))

        for result in results:
            times.append(result.get(timeout=5))

        return times

    def test_client(self):
        """
        Test that the server is currently running. Required for the
        remaining server tests to run.

        Input: Attempt to execute a known function on the server.

        Expected Result: No exception being raised, implying that the
            server
        is currently running.
        """

        exceptionRaised = False;
        try:
            getTime(self.startTime)
        except:
            exceptionRaised = True

        self.assertFalse(exceptionRaised, "The server isn't running.")

    def test_concurrent_connections(self):
        """
        Test that the server can handle the maxmimum number of concurrent
        connections while receiving a response in a similar time frame
        for all requests.

        Input: 5 concurrent function calls to the server.

        Expected Result: The remote function, test_concurrency, contains
            a 2
        second sleep call. The sum of the times taken to complete each of
        the function calls, relative to self.startTime should be between
            the
        range 11 > totalTime >= 10.
        """

        self.resetCurrentTime()

        totalTime = 0

        for time in self.getExecutionTimes(5):
            totalTime += time

        self.assertTrue(totalTime >= 10 and totalTime < 11, "Failed to "
            +
            "execute all 5 concurrent function calls within the expected
                " +
            "time frame.")

    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
```

```
            "}'

        self.assertEqual(response.read(), expectedResponse, "Wrong
            response.")

    def test_json_rpc_malformed_parameters(self):
        """
        Tests that the server responds with an error when
        a request contains incorrect parameters.

        Input: A JSON-RPC 2.0 request containing incorrectly
        formatted parameters to be passed on to the requested
        function.

        Expected Result: The returned JSON-RPC 2.0 response string
        should match the string expectedResponse, which contains
        a response representing an invalid parameters error.
        """
        conn = httplib.HTTPConnection("127.0.0.1:8080")
        fragment = base64.b64encode(b'<html></html>')
        params = []
        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 = '{"error": {"message": "Invalid parameters.",
            "code": -32602}, "jsonrpc": "2.0", "id": "A3s23"}'

        self.assertEqual(response.read(), expectedResponse, "Wrong
            response.")

    def test_json_rpc_unsupported_method(self):
        """
        Tests that the server responds with an error when
        a client attempts to make a function call for a function
        which hasn't been registered to the server.

        Input: A JSON-RPC 2.0 request containing a function name
        which hasn't been registered on the server.

        Expected Result: The returned JSON-RPC 2.0 response
        string should match the string expectedResponse, which
        contains a response representing an unsupported method
        error.
        """
        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":
            "not_registered",
             "params": params, "id": "A3s23"})
        header = {"Content-type": "application/json"}
```

```python

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

        expectedResponse = '{"error": {"message": "Method not_registered not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "A3s23"}'

        self.assertEqual(response.read(), expectedResponse, "Wrong response.")

    def test_invalid_filepath(self):
        """
        Tests that the server responds with an error when
        a client attempts to make a function call for a function
        which hasn't been registered to the server.

        Input: A JSON-RPC 2.0 request containing a function name
        which hasn't been registered on the server.

        Expected Result: The returned JSON-RPC 2.0 response
        string should match the string expectedResponse, which
        contains a response representing an unsupported method
        error.
        """
        conn = httplib.HTTPConnection("127.0.0.1:8080")
        fragment = base64.b64encode(b'<img src=../image.jpg><img src=directory2/image2.jpg>')
        params = [{"files": ["image.jpg", "directory/", "directory/current.html"], "document": fragment, "filename": "directory/current.html"}]
        request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method": "not_registered",
            "params": params, "id": "A3s23"})
        header = {"Content-type": "application/json"}

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

        expectedResponse = '{"error": {"message": "Method not_registered not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "A3s23"}'

        self.assertEqual(response.read(), expectedResponse, "Wrong response.")

    def test_invalid_filepath(self):
        """
        Tests that the server responds with an error when
        a client attempts to make a function call for a function
        which hasn't been registered to the server.

        Input: A JSON-RPC 2.0 request containing a function name
        which hasn't been registered on the server.

        Expected Result: The returned JSON-RPC 2.0 response
        string should match the string expectedResponse, which
        contains a response representing an unsupported method
        error.
        """
        conn = httplib.HTTPConnection("127.0.0.1:8080")
        fragment = base64.b64encode(b'<img src=../image.jpg><img src=
            directory2/image2.jpg>')
        params = [{"files": ["image.jpg", "directory/", "directory/
            current.html"], "document": fragment, "filename": "directory/
            current.html"}]
        request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
            "not_registered",
             "params": params, "id": "A3s23"})
        header = {"Content-type": "application/json"}

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

        expectedResponse = '{"error": {"message": "Method not_registered
            not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
            A3s23"}'

        self.assertEqual(response.read(), expectedResponse, "Wrong
            response.")

    def test_invalid_filepath(self):
        """
        Tests that the server responds with an error when
        a client attempts to make a function call for a function
        which hasn't been registered to the server.

        Input: A JSON-RPC 2.0 request containing a function name
        which hasn't been registered on the server.

        Expected Result: The returned JSON-RPC 2.0 response
        string should match the string expectedResponse, which
        contains a response representing an unsupported method
        error.
        """
        conn = httplib.HTTPConnection("127.0.0.1:8080")
        fragment = base64.b64encode(b'<img src=../image.jpg><img src=
            directory2/image2.jpg>')
        params = [{"files": ["image.jpg", "directory/", "directory/
            current.html"], "document": fragment, "filename": "directory/
            current.html"}]
        request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
            "not_registered",
             "params": params, "id": "A3s23"})
        header = {"Content-type": "application/json"}

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

```

```python
        expectedResponse = '{"error": {"message": "Method not_registered not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "A3s23"}'

        self.assertEqual(response.read(), expectedResponse, "Wrong response.")

    def test_invalid_filepath(self):
        """
        Tests that the server responds with an error when
        a client attempts to make a function call for a function
        which hasn't been registered to the server.

        Input: A JSON-RPC 2.0 request containing a function name
        which hasn't been registered on the server.

        Expected Result: The returned JSON-RPC 2.0 response
        string should match the string expectedResponse, which
        contains a response representing an unsupported method
        error.
        """
        conn = httplib.HTTPConnection("127.0.0.1:8080")
        fragment = base64.b64encode(b'<img src=../image.jpg><img src=directory2/image2.jpg>')
        params = [{"files": ["image.jpg", "directory/", "directory/current.html"], "document": fragment, "filename": "directory/current.html"}]
        request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method": "not_registered",
         "params": params, "id": "A3s23"})
        header = {"Content-type": "application/json"}

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

        expectedResponse = '{"error": {"message": "Method not_registered not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "A3s23"}'

        self.assertEqual(response.read(), expectedResponse, "Wrong response.")

    def test_invalid_filepath(self):
        """
        Tests that the server responds with an error when
        a client attempts to make a function call for a function
        which hasn't been registered to the server.

        Input: A JSON-RPC 2.0 request containing a function name
        which hasn't been registered on the server.

        Expected Result: The returned JSON-RPC 2.0 response
        string should match the string expectedResponse, which
        contains a response representing an unsupported method
        error.
        """
        conn = httplib.HTTPConnection("127.0.0.1:8080")
        fragment = base64.b64encode(b'<img src=../image.jpg><img src=
            directory2/image2.jpg>')
        params = [{"files": ["image.jpg", "directory/", "directory/
            current.html"], "document": fragment, "filename": "directory/
            current.html"}]
        request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
            "not_registered",
             "params": params, "id": "A3s23"})
        header = {"Content-type": "application/json"}

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

        expectedResponse = '{"error": {"message": "Method not_registered
            not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
            A3s23"}'

        self.assertEqual(response.read(), expectedResponse, "Wrong
            response.")

    def test_invalid_filepath(self):
        """
        Tests that the server responds with an error when
        a client attempts to make a function call for a function
        which hasn't been registered to the server.

        Input: A JSON-RPC 2.0 request containing a function name
        which hasn't been registered on the server.

        Expected Result: The returned JSON-RPC 2.0 response
        string should match the string expectedResponse, which
        contains a response representing an unsupported method
        error.
        """
        conn = httplib.HTTPConnection("127.0.0.1:8080")
        fragment = base64.b64encode(b'<img src=../image.jpg><img src=
            directory2/image2.jpg>')
        params = [{"files": ["image.jpg", "directory/", "directory/
            current.html"], "document": fragment, "filename": "directory/
            current.html"}]
        request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
            "not_registered",
             "params": params, "id": "A3s23"})
        header = {"Content-type": "application/json"}

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

        expectedResponse = '{"error": {"message": "Method not_registered
            not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
            A3s23"}'

        self.assertEqual(response.read(), expectedResponse, "Wrong
            response.")

    def test_invalid_filepath(self):
        """
        Tests that the server responds with an error when
        a client attempts to make a function call for a function
        which hasn't been registered to the server.

        Input: A JSON-RPC 2.0 request containing a function name
        which hasn't been registered on the server.

        Expected Result: The returned JSON-RPC 2.0 response
        string should match the string expectedResponse, which
        contains a response representing an unsupported method
        error.
        """
        conn = httplib.HTTPConnection("127.0.0.1:8080")
        fragment = base64.b64encode(b'<img src=../image.jpg><img src=directory2/image2.jpg>')
        params = [{"files": ["image.jpg", "directory/", "directory/current.html"], "document": fragment, "filename": "directory/current.html"}]
        request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method": "not_registered",
             "params": params, "id": "A3s23"})
        header = {"Content-type": "application/json"}

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

        expectedResponse = '{"error": {"message": "Method not_registered not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "A3s23"}'

        self.assertEqual(response.read(), expectedResponse, "Wrong response.")

if __name__ == '__main__':
    unittest.main()
```