

DECO3801 Test Plan Document

THEM - Typed HTML5 Evaluation Machine

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

Functional Test Plan

Testing Strategy

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

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

Test Case Transcript

Implications of Functional Testing

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

User Experience Goals

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

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

User Testing Plan

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

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

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

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

Summarise the results of your tests. For each scenario-based test: tabulate your metric results against each task describe or present your users feedback during/after the test Close with a general discussion that: summarises the issues raised identifies areas for improvement and design suggestions outlines any redefinition of functional and user test plans for final prototype (We dont care whether your prototype passes the tests. What is important is that your prototype is sufficiently broad to validate your test plan for the final product.)

Appendix - Python Test Code

Syntax Tests

```
1 from __future__ import absolute_import, division, unicode_literals
2
3 #from . import support
4 import unittest, html5lib
5 from html5lib import treebuilders
6
7 class TestSyntax(unittest.TestCase):
8     """
9     Provides a number of test cases to test the syntax used
10    in the document.
11    """
12
13    def setUp(self):
14        self.parser = html5lib.HTMLParser(tree=treebuilders.
15                                         getTreeBuilder("etree"))
16
17    def test_malformed_tag_name(self):
18        """
19        Test that the tag name isn't an invalid symbol.
20
21        Input:
22        A HTML fragment containing a tag with an invalid tag name.
23
24        Expected Results:
25        An error should be thrown reporting an invalid tag name.
26        """
27
28        inputFragmentEmptyName = "<html>< ></body>"
29        inputFragmentQuestionMark = "<html><?></body>"
30        inputFragmentRightBracket = "<html><></html>"
31
32        self.parser.parse(inputFragmentEmptyName)
33
34        self.assertIn(((6, 6), u'expected-tag-name', {u'data': u' '}),
35                      self.parser.errors, "Failed to report invalid tag name. Get ")
36
37        self.parser.reset()
38        self.parser.parse(inputFragmentQuestionMark)
39
40        self.assertIn(((6, 6), u'expected-tag-name-but-got-question-mark',
41                      {}),
42                      self.parser.errors, "Failed to report valid tag name. Got
43                      question mark instead.")
44
45        self.parser.reset()
46        self.parser.parse(inputFragmentRightBracket)
47
48        self.assertIn(((6, 7), u'expected-tag-name-but-got-right-bracket',
49                      {}),
50                      self.parser.errors, "Failed to report valid tag name. Got
51                      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
55             forwardslash.
56
57         Expected Results:
58         An error should be thrown reporting an invalid tag name.
59         """
60
61         inputFragment = "<html><a></a /></html>"
62
63         self.parser.parse(inputFragment)
64
65         self.assertIn(((9, 14), u'self-closing-flag-on-end-tag', {}),
66             self.parser.errors, "Failed to report misplaced forwardslash
67                 in closing tag.")
68
69     def test_invalid_self_closing_tag(self):
70         """
71         Test that the use of a self closing tag for a tag
72         which isn't considered a self closing tag returns
73         an error.
74
75         Input:
76         A HTML fragment containing a start tag with a trailing
77             forwardslash
78         (self-closing) for a tag type which isn't a self closing tag.
79
80         Expected Results:
81         An error should be thrown reporting the given tag type isn't a
82             self-closing
83             tag.
84         """
85
86         inputFragment = "<html><a /></html>"
87
88         self.parser.parse(inputFragment)
89
90         self.assertIn(((6, 10), u'non-void-element-with-trailing-solidus',
91             {u'name': u'a'}),
92             self.parser.errors, "Failed to report invalid self-closing
93                 tag.")
94
95     def test_attributes_in_end_tag(self):
96         """
97         Test that attributes occurring in a closing tag are
98         reported as an error.
99
100        Input:
101        A HTML fragment containing a closing tag which contains
102        at least one attribute.

```

```

98     Expected Results:
99     An error should be thrown reporting that the closing tag shouldn'
100         t contain
101     attributes.
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
109             tag.")
110
111 if __name__ == '__main__':
112     unittest.main()

```

Page Structure Tests

```
1 from __future__ import absolute_import, division, unicode_literals
2
3 #from . import support
4 import unittest, html5lib
5 from html5lib import treebuilders
6
7 class TestPageStructure(unittest.TestCase):
8     """
9     Provides a number of test cases related to basic page structure
10    for html5 documents.
11    """
12
13    def setUp(self):
14        self.parser = html5lib.HTMLParser(tree=treebuilders.
15                                         getTreeBuilder("etree"))
16
17    def test_singular_tags(self):
18        """
19        Test that the multiple-instance-singular-tag error is thrown
20        for cases where more than one instance of a singular tag block is
21        present.
22
23        Input:
24        Nested blocks of singular tags (html, body, head).
25        eg. <html><html></html></html>
26
27        Output:
28        All three test cases should report a multiple instance of both
29        the start and closing tags for each of the three singular tags.
30        """
31        multipleHTMLInstances = "<html><html></html></html>"
32        multipleHeadInstances = "<html><head><head></head></head><body></body></html>"
33        multipleBodyInstances = "<html><head></head><body><body></body></body></html>"
34
35        self.parser.parse(multipleHTMLInstances)
36
37        self.assertIn(((6, 11), u'multiple-instance-singular-tag', {u'
38            name': u'html'}),
39            self.parser.errors, "Multiple instances of starting HTML tag
40            not reported.")
41
42        self.assertIn(((12, 18), u'incorrect-placement-html-end-tag', {u'
43            name': u'html'}),
44            self.parser.errors, "Multiple instances of closing HTML tag
45            not reported.")
46
47        self.parser.reset()
48        self.parser.parse(multipleHeadInstances)
49
50        self.assertIn(((12, 17), u'multiple-instance-singular-tag', {u'
51            name': u'head'}),
52            self.parser.errors, "Multiple instances of starting HTML tag
53            not reported.")
```

```

47
48     self.assertIn(((25, 31), u'incorrect-placement-singular-end-tag',
49         {u'name': u'head'})),
50         self.parser.errors, "Multiple instances of closing head tag
51             not reported.")
52
53     self.parser.reset()
54     self.parser.parse(multipleBodyInstances)
55
56     self.assertIn(((25, 30), u'multiple-instance-singular-tag', {u'
57         name': u'body'})),
58         self.parser.errors, "Multiple instances of starting HTML tag
59             not reported.")
60
61     self.assertIn(((38, 44), u'unexpected-end-tag-after-body', {u'
62         name': u'body'})),
63         self.parser.errors, "Multiple instances of closing body tag
64             not reported.")
65
66 def test_missing_doctype(self):
67     """
68     Test that the expected-doctype-but-got-start-tag error is thrown
69     for cases where no DOCTYPE is declared.
70
71     Input:
72     Nested blocks of singular tags (html, body, head), all of which
73     are missing the DOCTYPE declaration.
74     eg. <html><html></html></html>
75
76     Expected Results:
77     All test cases should report a missing DOCTYPE declaration.
78     """
79     startTagBeforeDoctype = "<html><html></html></html>"
80     endTagBeforeDoctype = "</head></head>"
81     eofBeforeDoctype = ""
82
83     self.parser.parse(startTagBeforeDoctype)
84
85     self.assertIn(((0, 5), u'expected-doctype-but-got-start-tag', {u'
86         name': u'html'})),
87         self.parser.errors, "Failed to report missing DOCTYPE
88             declaration (start tag before doctype.")
89
90     self.parser.reset()
91     self.parser.parse(endTagBeforeDoctype)
92
93     self.assertIn(((0, 6), u'expected-doctype-but-got-end-tag', {u'
94         name': u'head'})),
95         self.parser.errors, "Failed to report missing DOCTYPE
96             declaration (closing tag before doctype.")
97
98     self.parser.reset()
99     self.parser.parse(eofBeforeDoctype)
100
101     self.assertIn((-1, -1), u'expected-doctype-but-got-eof', {}),
102         self.parser.errors, "Failed to report missing DOCTYPE
103             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
113                                tag.")
114
115        self.parser.reset()
116        self.parser.parse(multipleBodyInstances)
117
118        self.assertIn((-1, -1), u'no-closing-html-tag', {}),
119            self.parser.errors, "Failed to report missing closing HTML
120                                tag.")
121
122     def test_misplaced_tags_before_head(self):
123         """
124         Test that both start and closing tags occurring before the head
125         section are reported as being misplaced.
126
127        Input:
128        A number of instances of start and closing tags being placed
129        before
130        the head section.
131
132        Expected Results:
133        Report whether or not the tags preceding the head section are
134        reported
135        as being misplaced.
136        """
137        misplacedHeadTags = "<html><body></body><head></head></html>"
138        misplacedLinkTags = "<html><a></a><head></head><body></body></
139                                html>"
140
141        self.parser.parse(misplacedHeadTags)
142
143        self.assertIn((6, 11), u'incorrect-start-tag-placement-before-
144                                head', {u'name': u'body'}),
145            self.parser.errors, "Failed to report start body tag before
146                                head section.")
147
148        self.assertIn((12, 18), u'incorrect-end-tag-placement-before-
149                                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 occurring 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 occurring
185         after the start and closing HTML pair.
186
187         Expected Results:

```

```

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

```

```

239     self.parser.parse(inputFragment)
240
241     self.assertIn(((32, 34), u'unexpected-start-tag-after-body', {u'
242         name': u'a'})),
243         self.parser.errors, "Failed to report misplaced starting tag
244         found after the closing body tag.")
245
246     self.assertIn(((35, 38), u'unexpected-end-tag-after-body', {u'
247         name': u'a'})),
248         self.parser.errors, "Failed to report misplaced closing tag
249         found after the closing body tag.")
250
251     def test_missing_closing_html_tag(self):
252         """
253         Test that a missing closing HTML tag is reported.
254
255         Input:
256         A HTML fragment missing a closing HTML tag.
257
258         Expected Results:
259         An error should be thrown stating that the closing HTML tag is
260         missing.
261         """
262         inputFragment = "<html><head></head><body></body>"
263
264         self.parser.parse(inputFragment)
265
266         self.assertIn((-1, -1), u'no-closing-html-tag', {}),
267             self.parser.errors, "Failed to report missing closing HTML
268             tag.")
269
270     def test_early_termination_before_head(self):
271         """
272         Test that an early closing HTML tag before the head phase
273         is reported as an error.
274
275         Input:
276         A HTML fragment with the head and body sections placed after
277         a closed set of HTML tags.
278
279         Expected Results:
280         An error should be thrown stating that the closing HTML tag
281         has been found before the head phase.
282         """
283         inputFragment = "<html></html><head></head><body></body>"
284
285         self.parser.parse(inputFragment)
286
287         self.assertIn(((6, 12), u'early-termination-before-head', {u'name
288             ': u'html'})),
289             self.parser.errors, "Failed to report early termination
290             before head section.")
291
292     def test_early_termination_in_head(self):
293         """

```

```

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':
305         u'html'})),
306         self.parser.errors, "Failed to report early termination
307         before head section.")
308
309 def test_early_termination_before_body(self):
310     """
311     Test that an early closing HTML tag before the body phase
312     is reported as an error.
313
314     Input:
315     A HTML fragment with the closing HTML tag placed before the body
316     section.
317
318     Expected Results:
319     An error should be thrown stating that the closing HTML tag
320     has been found before the body phase.
321     """
322
323     inputFragment = "<html><head></head></html><body></body>"
324
325     self.parser.parse(inputFragment)
326
327     self.assertIn(((19, 25), u'early-termination-before-body', {u'
328         name': u'html'})),
329         self.parser.errors, "Failed to report early termination
330         before head section.")
331
332 def test_early_termination_in_body(self):
333     """
334     Test that an early closing HTML tag in the body phase
335     is reported as an error.
336
337     Input:
338     A HTML fragment with the closing HTML tag placed within
339     the set of body tags.
340
341     Expected Results:
342     An error should be thrown stating that the closing HTML tag
343     has been found in the head phase.
344     """

```

```

341     inputFragment = "<html><head></head><body></html></body>"
342
343
344     self.parser.parse(inputFragment)
345
346     self.assertIn(((25, 31), u'early-termination-in-body', {u'name':
347         u'html'})),
348         self.parser.errors, "Failed to report early termination
349         before head section.")
350
351 def test_tags_between_head_body(self):
352     """
353     Test that a set of tags placed after the head section
354     but before the body section is reported as an error.
355
356     Input:
357     A HTML fragment with a set of tags between the head
358     and body sections.
359
360     Expected Results:
361     An error should be thrown stating that the set of tags
362     can't be placed between the head and body sections.
363     """
364
365     inputFragment = "<html><head></head><a></a><body></body></html>"
366
367     self.parser.parse(inputFragment)
368
369     self.assertIn(((19, 21), u'start-tag-before-body-after-head', {u'
370         name': u'a'})),
371         self.parser.errors, "Failed to report start tag after head
372         phase but before body phase.")
373
374     self.assertIn(((22, 25), u'end-tag-before-body-after-head', {u'
375         name': u'a'})),
376         self.parser.errors, "Failed to report closing tag after head
377         phase but before body phase.")
378
379 def test_missing_starting_html_tag(self):
380     """
381     Test that a missing starting HTML tag is reported as an error.
382
383     Input:
384     A HTML fragment missing a starting HTML tag.
385
386     Expected Results:
387     An error should be thrown indicating that the fragment doesn't
388     contain a starting HTML tag.
389     """
390
391     inputFragment = "<head></head><body></body></html>"
392
393     self.parser.parse(inputFragment)
394
395     self.assertIn((-1, -1), u'no-starting-html-tag', {}),
396         self.parser.errors, "Failed to report missing starting HTML
397         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
402         invalid.
403     """
404     inputFragment = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN
405                     "http://www.w3.org/TR/html4/strict.dtd">'
406
407     self.parser.parse(inputFragment)
408
409     self.assertIn(((0, 89), u'unknown-doctype', {}),
410                  self.parser.errors, "Failed to report unknown doctype.")
411
412 def test_space_after_doctype(self):
413     """
414     Test that a doctype tag has a space between the doctype
415     declaration
416     and the doctype name.
417
418     Input:
419     A HTML fragment containing a doctype with no space between the
420     doctype
421     declaration and the doctype name.
422
423     Expected Results:
424     An error should be thrown reporting that there is no space
425     between
426     the doctype declaration and the doctype name.
427     """
428     inputFragment = '<!DOCTYPEhtml>'
429
430     self.parser.parse(inputFragment)
431
432     self.assertIn(((0, 8), u'need-space-after-doctype', {}),
433                  self.parser.errors, "Failed to report missing space after the
434                  doctype declaration.")
435
436 if __name__ == '__main__':
437     unittest.main()

```