It turns out that the answer to the ultimate question of life, the Universe, and CSS Tests isn’t a number. It is, in fact, multiple numbers. It is the answers to:

- How many test results are correct?
- How many results in the modules that we care about are correct?
- How many test results have changed in the last AH Formatter build?
The tests that I’m talking about is the CSS WG test suite. The CSS Working Group (WG) has a large and ever-expanding set of tests. Currently it contains over 17,000 tests.

The tests are divided into nearly 100 modules (counting CSS 2.1 as multiple modules). The tests are meant to be run in a browser, and the CSS WG provides a browser-based framework for running the tests and comparing them against a reference or, for many tests, checking JavaScript assertions about the test.

However, a browser-based framework and JavaScript assertions aren’t much use for a batch formatter that produces PDF, so this talk is about the evolution of an in-house system for running AH Formatter on the CSS WG tests and working with the results.
I’m not going to begin to explain the CSS modules to you. I will say, however, that the CSS WG categorizes their modules by specification level and stability.

Most of you will be familiar with the progression of a W3C specification from ‘First Public Working Draft’ through to ‘Recommendation’.

The stability levels are used only by the CSS WG. The levels range in increasing stability from ‘Rewriting’ and ‘Exploring’ through to ‘Stable’ and ‘Completed’. W3C Recommendations, by definition, are ‘Completed’, but the stability of modules at other levels can vary a lot.
This is a partial screenshot of the current CSS WG page that lists the modules ordered by stability level and showing their current and next specification levels.

You might think that a module would progress through all the stability levels on the way to being a completed Recommendation, but, as you can see, more ‘Testing’-level modules than ‘Stable’-level modules will progress next to Proposed Recommendation.
This is a screenshot of a CSS test in the CSS WG’s test framework. The framework can show the test and its reference page (if there is one).

You can indicate the state of the test result by clicking on one of these buttons.

As I said, there’s more than 17,000 CSS tests. The CSS Working Group have multiple frameworks for checking test results. Not surprisingly, these run in a browser. Which means, of course, that their framework isn’t much use when testing a batch formatter that produces PDF. There really was only one thing to do...
Okay, so there was really two things to do...
I was already using Antenna House’s own regression testing system for testing XSL-FO, so I just adapted that for use with the CSS WG tests.
I had automated the testing by using the Jenkins Continuous Integration (CI) server.

My Jenkins jobs execute Ant targets so that the Jenkins configuration is as simple as possible.

Ant runs AH Formatter and then runs AHRTS on the results.
This is an AHRTS overview report from a previous run. It’s a bit different from the standard AHRTS report since, as well as showing whether there are any differences between the base files and the new files, it’s also reporting on the conformance of the results.
The individual reports highlight any differences between the base and the new version of a file.
To make it easy to see what’s changed, the ‘Differences’ pane has the changes highlighted, with different colors for what’s in one version and what’s in the other.
My change to the individual reports was to add clickable links corresponding to the different states that the CSS WG uses to report test results. Each link runs a Jenkins job that logs the test’s result. The intention was to eventually be able to submit the test results to the CSS WG so more AH Formatter results could be listed alongside the browsers’ results.
However, when the developers in Japan looked at the AHRTS report, they wanted a format that was similar to what they’d used when testing MathML and SVG support.
This seemed straightforward enough: the extra fields could be captured using an Acroform inside the individual report PDFs that AHRTS produced. The form data would then be submitted to Jenkins.

However, this had several problems:

• I couldn’t find a Linux PDF reader that would update the form.
• Acrobat on Windows viewed filling in the form as changing the file, so closing the file brought up a dialog box asking if the changes should be saved. I was later told a workaround for this, but by then I had already moved away from using Acroforms.
• Acrobat would also pop-up a dialog box about storing the HTTP response from Jenkins.
• Lastly and most importantly, Jenkins would correctly store ASCII text from the form fields, but Japanese text was garbled in a way that I could not unscramble.
So the project moved to using eXist-db. I had always intended moving to an XML database once the data was complex enough. The data wasn’t particularly complex, but the need to handle Japanese text made a good reason to change.

I chose eXist-db partly because I knew some of the developers, which meant that I knew who to ask if I had problems, but the eXist-db mailing list was quite responsive whenever I’ve had a problem and I’ve never needed to approach anyone directly.
When eXist-db is added to the picture, Jenkins still runs Ant both to run AH Formatter on the tests and to run AHRTS on the test results. Now, however, the Ant targets also upload both the XML and individual PDFs produced by AHRTS to eXist-db.

This solved the problem with Japanese comments, but it didn’t solve the problems with Acrobat considering modifying the Acroform as modifying the file. The breakthrough came when, instead of putting the form inside the PDF, I put the PDF inside the form and created an eXist-db ‘app’ for viewing the AHRTS results in a web browser.
My first version of the app used an XForm for capturing the comment, etc.

The XForm looked rather dated, so I changed the form to use Bootstrap (an open source JavaScript framework).

The single greatest usability improvement in the form was when I added a ‘Submit and next’ button that both submits the current test’s results to eXist-db and advances to show the results for the next test.

The other usability improvement was modifying the XSL-FO for the AHRTS individual PDFs to show the difference page(s) before the summary page so that the differences (if any) can be viewed without having to scroll the PDF.

Some tests fail so completely that they don’t produce a PDF file, so I added XSLT for Jenkins to run that inserts the logs that AH Formatter generates into the XML reports that AHRTS generates so that eXist-db can include the log in each individual report’s HTML page.
A similar process happened with the summary report.

The initial version was just HTML generated from an XQuery. The first change was that my colleagues in Japan decided that they didn’t want to see the totals for each result category.

The report was then ‘bootstrapped’ to improve its appearance.

Then, instead of showing all of the results in one form, the results were delivered module-by-module.

Of course, having decided to not show all of the results all at once, showing all the results at once was added back as an optional view, along with views for other categories of test result.

My colleagues also wanted to see the Antenna House priority for a module plus the specification level and the stability for the module.

I also added options to export and import test results so that results could be shared between my colleagues in Japan and myself.
It turns out that the answer to the ultimate questions of Life, the Universe, and CSS Tests isn't even three numbers. It's actually lots of numbers.

As time went by, my colleagues wanted to see more information about the test results, so I added a dashboard that summarized the test results.

The dashboard started out as a single table, but, over time, it has expanded into three tables, and additional columns have been added to tables whenever someone has needed to know additional information.
There is a theory which states that if ever anyone has results for the entire CSS test suite, the test suite will instantly be replaced by something even more bizarre and inexplicable. There is another theory that states that this has already happened, since, last year, the CSS WG test suite was merged into the much larger ‘web-platform-tests‘ test suite.

This might not have been so bad, but as part of the change, some modules were renamed to align the module name with the short-name of their specification.

This required a one-off Ant task to copy and rename AHRTS test results to make reference files for the new directory structure.

It also required adding an option for a job in eXist-db to copy and rename the job’s results to become the results for the new job for the Web Platform Tests results.
I’m now going to talk about some aspects of the internals of the project.

When I started using eXist-db, I assumed that I should use XForms with my XML data. However, as you saw before, the XForms pages do not look particularly impressive.

eXist-db uses the Bootstrap JavaScript for its demo application, so I started using Bootstrap with the pages containing forms. I have now replaced all uses of XForms with Bootstrap except for the form for importing test result data which does a file upload.
Another feature of the eXist-db demo applications is its templating mechanism. You create the look of a page or a series of pages using a template HTML file. The individual HTML page slots into the structure established by the template page. The templating mechanism handles URI parameters, which further reduces the housekeeping code that you need to write.
This are pages from the app and, as you can see, they are based heavily on the eXist-db sample app.
This is a template for a HTML page. Pay attention to the `<div>` with `id="content"`. 
This is the HTML for a page that will be served by the app. This `<div>` replaces the `<div>` in the template with the matching ID.

Parts of the HTML are generated by evaluating an XQuery function.
This `<div>` replaces the corresponding `<div>` from the template when eXist-db serves the HTML page.

Obviously, different pages of the app can use the same template HTML by providing different content for the `<div>`.
A HTML element with a 'data-template' attribute is replaced by the result of the named function.

In this case, the 'app:list-jobs2' function returns the HTML markup for displaying a list of the known jobs.
The `app:list-jobs2` function executes the Xquery code for determining the known jobs and returns the HTML markup.

The HTML uses CSS class names from Bootstrap to set the appearance of the HTML.

The `$node` and `$model` arguments are common to every function that the templating mechanism calls.
\$node and \$model

- \$node – Current node from HTML
- \$model – XQuery map of application data
- Additional function parameters from HTTP request, HTML session, or static defaults in XQuery

‘\$node’ is the current node from the HTML.

‘\$model’ is a map of application-specific data.

The templating mechanism can also populate values for function parameters that correspond to HTTP request parameters.
The alternative to a function returning HTML markup is that the function can, instead, modify the map of application-specific data. The modified map is available to the functions that are run from the HTML that the HTML for this function wraps.
Templates are useful. They make it easy to separate the HTML from the XQuery code.

However, it can be tempting to put too much into the map. The ‘$new-exists’ that was added to the map by the code in the previous slide was actually only used in an ‘if’ statement. In the code shown here, it’s possible that ‘$new-exists’ is never used, despite being evaluated every time that the page is accessed.

Also, I could not get the templating mechanism to work with the page that uploads a Zip file, so that page is implemented using just XQuery.
This is a screenshot of the eXide editor that is part of eXist-db.

It runs in your browser, and it is the main tool for writing XQuery code.
eXide

- XQuery IDE
- Runs in web browser
- Operates on documents in eXist-db
- Can synchronize code to disk
- Used for exporting application package

eXide can edit documents that are in the eXist-db database. Two of its other useful functions are synchronizing your XQuery code to disk (so that the code can be checked in to a version control system) and exporting application packages as `.xar` files.
Since eXide can synchronize your XQuery code to disk but cannot synchronize files in eXist-db to match files on the disk, it’s not straightforward to transfer XQuery code between machines.

In practice, I ended up exporting the app from one eXist-db instance and then installing the app on the eXist-db instance on the second machine.

If necessary, I could edit the XQuery files on the second machine and synchronize those files to disk on the second machine (and check the changed files into the distributed version control system), but getting the changed files back onto the first eXist-db instance required exporting the updated app from the second instance and installing that on the first instance.

This wasn’t needed very often, otherwise I would have found a way to update the XQuery code from a version control system check-out, possibly using either the eXist-db Ant task or the eXist-db Java client.
The app is currently partially internationalized. It is only partial both because I could not work out how to localize the XForms controls and because I was asked to undo some of my translations.

This example illustrates the problem with plucking translations from an online translation tool. These messages are from test result comments. Can you see the problem?
This should make it clear to you.
The i18n library in eXist-db uses a catalogue file format that is structurally identical to Java XML property files.

I18n in eXist-db

- Catalogue file

```xml
<catalogue xml:lang="ja">
  <msg key="Comment">コメント</msg>
  <msg key="Date">日</msg>
  <msg key="Diff">相違</msg>
  <msg key="Issue">発行</msg>
  <msg key="Test">テスト</msg>
</catalogue>
```
The i18n library replaces the content of the `<i18n:text>` elements with the localized text.
The `<i18n:text>` element isn’t useful for localizing attribute values, and I didn’t like repeating the text in the ‘key’ attributes, so I wrote my own i18n library.

```html
<input type="text" class="form-control" name="issue" id="issue"
placeholder="{common:i18n-text('Issue numbers')}"
value="{$testresult/issue/string()}" accesskey="i"/>
```
The verdict from my colleagues in Japan was uniformly positive.

The Verdict

- “Using AHRTS for automatically identifying changed test results is a big advantage.”
- “Managing the tests is much more effective than with the HTML report.”
- “Having the test result status, comment, AHRTS report, and AH Formatter log on one page is useful.”
- “The extra functionality added during the course of the project has made it even more useful.”
- “The system has saved time and effort.”
Summary

- eXist-db + browser best alternative
- 100% Recommendation and Candidate Recommendation tests tested
- Iterative development delivered features when wanted
- Users satisfied
- Also testing XSL-FO and testing local fork
  – Two AHRTS diffs: against previous and trunk

In summary:

- Using eXist-db and putting the test results in the browser was the best alternative that I tried
- 100% of the tests for CSS modules at the Recommendation and Candidate Recommendation stages have been tested
- The iterative development delivered features as they were wanted, including features (such as listing results for the entire test suite) that were removed and reinstated
- The users are satisfied with the result
- My testing has gone full circle: after basing the CSS testing on how I tested XSL-FO, I now present XSL-FO test results based on how I present CSS test results
- Another variant for checking results from a local fork of the AH Formatter code shows two AHRTS individual reports for each test: one with the differences between the current and previous results for the forked code, and the other between the current results for the forked code and the current results for the trunk code
So long, and thanks for all the fish.