MobiWebApp

Mobile Web Applications for Future Internet Services

Deliverable D3.1

Test Suites Report Year 1

Month 12 - Version 1

Project	
Grant Agreement number	257800
Project acronym:	MobiWebApp
Project title:	Mobile Web Applications for Future Internet Services
Funding Scheme:	Coordination & Support Action
Date of latest version of Annex I against which the assessment will be made:	March 17, 2010
Document	
Deliverable number:	D3.1
Deliverable title	Test Suites Report Year 1
Contractual Date of Delivery:	Project month M12
Actual Date of Delivery:	September 01, 2011
Editor (s):	Tom Williamsom
Author (s):	François Daoust
Reviewer (s):	Dominique Hazaël-Massieux
Participant(s):	ERCIM/W3C
Work package no.:	3
Work package title:	Interoperability
Work package leader:	François Daoust
Distribution:	PU
Version/Revision:	
Draft/Final:	
Total number of pages (including cover):	10
Keywords:	Testing, Framework, Interoperability

DISCLAIMER

This document contains description of the MobiWebApp project work and findings.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any responsibility for actions that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of the MobiWebApp consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 27 Member States of the Union. It is based on the European Communities and the member states cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors. (http://europa.eu.int/)

MobiWebApp is a project funded in part by the European Union.

TABLE OF CONTENTS

Disclaimer....3
Table of Contents....4
Summary....5
Introduction....6
1 Testing Activity at W3C....7
2 Testing Framework Deployment....10
3 Next Steps....15
Conclusion....17
Appendix 1: Requirements for Testing framework....18

SUMMARY

This report provides an overview of the achievements of the Interoperability Work Package during the first year of MobiWebApp.

MobiWebApp took an active role in preparing the foundations of a generic testing activity within W3C which includes mobile platform testing as a primary goal. As of August 2011, this activity is under review by the W3C Membership and should be launched in September.

In the meantime, MobiWebApp contributed to initial developments and deployments of the testing framework that welcome test suites and enable the generation of reports on the state of the mobile Web application platform. This report details the directions explored and the contributions made by MobiWebApp to this work.

The second year of the project will focus on actual developments and incorporation of test cases into new framework.

INTRODUCTION

With the advent of Web applications in general, and mobile Web applications in particular, interoperability has become a fundamental key for the success of the Web as an application platform. This platform typically relies on W3C specifications such as HTML5, CSS, SVG, device APIs, XMLHttpRequest, widgets, etc.

Working Groups in W3C create test suites as part of their work on Web standards to ensure interoperability between implementations, improve the overall quality of the specification itself, and meet entrance criteria for a specification to be published as a Web standard. This testing effort is done on a working group per working group basis, using ad-hoc testing frameworks, servers and methods to measure the coverage of a specification in terms of tests. Thus, the quality of the resulting test suites varies from group to group, with no easy way to retrieve and combine results from different test suites.

Back in 2005, the W3C Quality Assurance activity developed guidelines to write clear and unambiguous specifications [1]. More recently, in 2010, through the MobiWeb 2.0 EU project, the W3C Mobile Web Test Suites Working Group published A Method for Writing Testable Conformance Requirements [2] and adapted several existing test suites to work on mobile devices.

When the MobiWebApp project started end of 2010, the W3C was looking into starting an activity dedicated to testing, with a view to developing a common and multi-platform testing framework that could be re-used across working groups, and to promote the adoption of the method mentioned above by working groups. Given the cross-working group nature of this activity and the fact that W3C members expect the W3C staff to be assigned to specific working groups, finding resources in W3C to draft initial activity proposals and start developments is not easy.

Through the Interoperability Work Package, the MobiWebApp project funds the time of a W3C team member in a testing team dedicated to setting up and running a generic testing activity within W3C, with a focus on the development of a generic testing framework that can be reused across working groups. The role of this team member is:

- to ensure that the testing framework can be used on mobile platforms.
- to ensure that mobile actors take an active role in that testing activity
- to ensure that specifications that compose the mobile Web application platform [3] are covered
- to help produce “state of the Open Web Platform” reports focused on mobile devices

This report is structured as follows:

- Section 1 details the progress towards setting up a generic testing activity in W3C.
- Section 2 goes into details on the generic testing framework that was deployed in parallel as a starting point for this activity.
- Section 3 details plans for the second year

[1] http://www.w3.org/TR/qaframe-spec/
[2] http://www.w3.org/TR/test-methodology/
[3] http://www.w3.org/2011/05/mobile-web-app-state

1 TESTING ACTIVITY AT W3C

W3C is a consortium where each member can contribute to one or more groups. Provided they fulfill requirements defined in the W3C Process Document, groups are free to organize themselves as they see fit, when it comes to developing test suites in particular. W3C does not impose tools and methods unless W3C Members ask for it. Thus, for a generic testing framework to be adopted by working groups, a formal activity that involves W3C Members needs to be created. This activity allows:

- to gather requirements from W3C Members for the testing framework
- to provide a coordination point with working groups in W3C and external organizations, to advertise and promote the adoption of the framework as well as to gather further requirements that could arise over time in a timely manner.
- to prioritize work items derived from the testing framework (e.g. the generation of implementation reports)
- to ensure W3C Members endorse and contribute resources to the activity

As of August 2011, the proposed testing activity is under review by W3C Members and is expected to start in September. It is composed of a Web Testing Interest Group, responsible for the testing framework, and of a Browser Testing and Tools Working Group which is to develop an API to ease debugging and improve the number of test cases that may be automated in the long run. Both groups are presented below. MobiWebApp efforts will be invested in the Interest Group as its scope is directly aligned with the objectives of MobiWebApp's Interoperability work package.

1.1 Web Testing Interest Group

The purpose of the Web Testing Interest Group is to develop and deploy testing mechanisms and collateral materials for testing of Web technologies across different devices (desktops, mobile, TV, …), to support the testing efforts of working groups developing Web technologies, and to liaise with organizations outside the W3C who conduct testing and certification efforts for Web technologies.

The testing team prepared a draft charter [4] to set the scope of the group. MobiWebApp contributed comments to emphasize the importance of the mobile platform. The charter is now explicit that “tests developed as well as the testing framework should work on non-desktop devices such as mobile devices, web-enabled television sets etc.”, ensuring that the availability of test suites on mobile Web platforms is a primary goal of the initiative.

MobiWebApp got in touch with mobile network operators (AT&T, France Telecom, Telecom Italia, Vodafone) to ensure that the draft charter was aligned with their goals. As mentioned in Section 2.1 below, MobiWebApp also gathered and assembled an initial set of requirements for the generic Testing framework.

The deliverables of the Web Testing Interest Group include the testing framework, of course, as well as a test suite management system , test-case results reporting/output tools , specification annotation tools (to link test cases to testable assertions and thus provide some assessment of test coverage), and documentation on test-case authoring, execution, and results collection.

The group will not develop test cases but will provide assistance to working groups using the testing framework, and will review resulting test suites.

[4] http://www.w3.org/2011/05/testing-ig-charter.html

1.2 Browser Testing and Tools Working Group

Initial discussions with several Web browser vendors on the scope of the Web Testing Interest Group revealed an interest to go beyond using existing technologies to improve the efficiency of test suites. More precisely, Web browser vendors raised the need to standardize a common set of script APIs:

- for use in automated testing
- and for debugging of Web applications.

For automated testing, there is a specific need to simulate user actions such as clicking links, entering text, and submitting forms, as well as taking screenshots of a Web page (to automate reference tests where the visual rendering of a page is compared to that of a reference page).

For debugging of Web applications, the APIs under consideration include inspecting and modifying the DOM for a particular document/application, monitoring and analyzing network activity and memory usage, as well as programmatic access to error logs and other debug information emitted by the JavaScript engine.

Both sets of APIs are in scope of the Browser Testing and Tools Working Group, as defined in the draft charter [5] of the group.

From a MobiWebApp perspective, the first set of APIs is of interest as it means more tests can be automated when browsers support the API. It is particularly relevant on mobile devices where user interaction is more limited. It is a longer term goal since the work involves standardization. We don't expect that test cases will be able to use the API by the end of the project. We also do not anticipate spending time on the working group in MobiWebApp.

1.3 Involving mobile actors

Even though the charter of the Interest Group explicitly mentions mobile as a supported platform for the testing framework, the only way to guarantee that mobile requirements get addressed is to ensure mobile actors take an active role in the testing activity.

Thanks to MobiWebApp support, the testing activity has been promoted to mobile network operators. Discussions showed that, on top of the testing framework itself, mobile network operators are particularly interested in publishing regular views of the state of the mobile Web application platform, on a user agent per user agent basis.

Such reports are to be automatically generated from test results extracted from the testing framework. In the long run, they could be completed by external sources of results (results provided by Web sites such as the HTML5 Test Web site [6] for instance).

Vodafone, Telecom Italia and AT&T expressed the same needs. Telecom Italia and AT&T also insisted on the importance of completing and promoting the adoption of a methodology to write testable specifications, to ease the extraction of testable assertions, ease the development of test cases and assess the level of coverage. These goals are well aligned with the goals of MobiWebApp's interoperability work package.

We also exchanged with France Telecom through MobiWebApp collaboration with the MOSQUITO project.

As of August 2011, after various meetings and exchanges with people involved in testing in different mobile network operators, the situation is:

[5] http://www.w3.org/2011/08/browser-testing-charter.html
[6] http://html5test.com/

- AT&T is willing to take an active role in the Interest Group. The exact role is still under discussion: group co-chair, task force leader, or editor.
- Vodafone assigned one of their handset testing expert to take part in discussions on the public mailing-list public-test-infra@w3.org used by the testing team while the Interest Group gets created.
- People involved in mobile testing at Telecom Italia have indicated their interest in participating.
- France Telecom stated their intent to participate in the Interest Group in response to the call for review for the Testing activity.

MobiWebApp will continue to promote the Interest Group among mobile network operators. Active contribution from mobile actors is also a good way to build on synergies between W3C and organizations looking into certifying mobile products, such as WAC.

2 TESTING FRAMEWORK DEPLOYMENT

In parallel to setting up a formal testing activity at W3C, the testing team at W3C has started to work on the testing framework, from an initial set of requirements to the deployment of test server and a first testing framework.

A typical test case for Web technologies that are relevant to the mobile Web Application platform consists of a short Web page that uses the functionality under test and, as much as possible and practical, uses only the feature under test. In other words, typical test cases are atomic, checking one particular testable assertion in a given section of a given specification.

Depending on the feature under test, test cases may be of different types, for instance:

- JavaScript may be used to test and report the result to the user.
- Reference tests pass when the rendering of the test page matches that of a reference page. In turn, this comparison could be done on an automated basis or on a manual basis.
- DOM tree comparison tests pass when the DOM tree generated by the test page matches that of a reference page. Such tests are useful for parsing tests.
- Self describing tests ask users to indicate whether the test has passed or failed, e.g. « please check that the input field below is disabled ». Ideally, for scalability reasons, such test cases should be avoided when possible.

The role of the testing framework is to run test cases in order and collect results, automatically when possible or manually through user interaction when not. The testing framework sits as a higher level layer on top of test cases. Test cases should not depend on this upper layer so that the framework may be updated or adapted to other environments without impacting test cases.

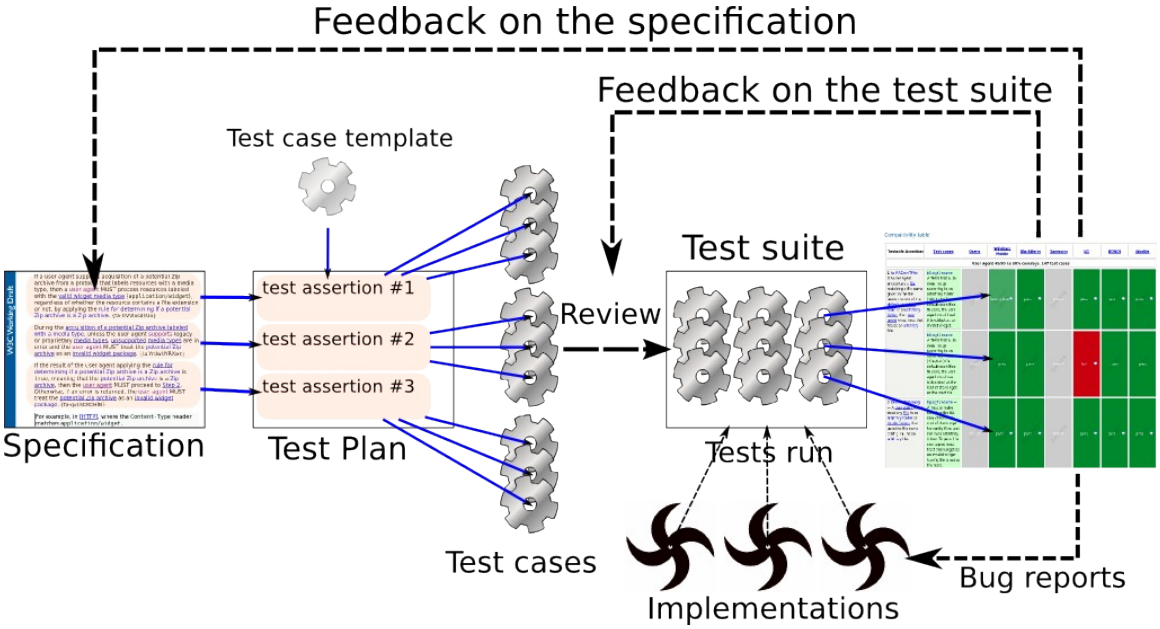

Figure 1 – Integration of test suite development and specification authoring

The testing framework goes beyond running test cases: it needs to be integrated as early as possible in the design of technical specifications to provide feedback loops that will improve the quality of the resulting specification (illustrated in Figure 1).

2.1 Requirements

To design the testing framework, the testing team has started to gather requirements from several working groups, starting with the HTML and Web Applications working groups. Gathering initial requirements allowed to refine the architecture and components of the testing framework. It consists of:

- a **Web test server** to serve test cases over the Web
- a **test runner** to run a series of tests and gather results for all of them
- a **test case review mechanism** to ensure the correctness of submitted test cases
- a **test suite management system**
- a **reporting tool** to produce implementation and interoperability reports
- a **spec annotation tool** to assess the level of coverage of a specification

The current list of requirements is available on W3C's public Wiki [7] and listed in Appendix 1. This list is likely to be adjusted by the Interest Group throughout the second year of the MobiWebApp project. In particular, priorities will need to be set to strike the right balance between the amount of resources available to work on the testing framework and the testing framework people would like to have.

MobiWebApp gathered requirements from mobile actors, proposed to partition the testing framework into functional components and adapted requirements from desktop, mobile and accessibility worlds into actionable statements for each component of the testing framework, for instance: “The test runner must allow for complete and partial execution of tests”.

2.2 Test server

Given that most test cases are composed of small Web pages, the first requirement that the testing team addressed was to deploy a Web server dedicated to hosting and serving test cases.

The system needed to scale well with the number of persons contributing test cases, as most working groups invite the community at large to contribute to the development of their test suites. Also, the system needed to track versions and allow to roll back changes whenever needed. Eventually, as much as possible, test cases should be deployed to the Web server as soon as a user commits changes.

The setup of the test server [8] is as follows:

- Test cases are defined in a distributed version control system, using Mercurial [9]. This setting allows users to work on local copies of the test suite, fork content as needed and push updates when ready. Mercurial has been precisely developed to cope well with scaling issues triggered by the number of users that may push changes to the central repository.
- To enable the automatic deployment of content pushed by users while preserving security, the test server uses its own dedicated domain name w3c-test.org. This makes it impossible for potential attackers to gain access to protected W3C resources on w3.org through the injection of malicious JavaScript code in test cases that could have been badly designed via so called cross-site scripting (XSS) attacks.

[7] http://www.w3.org/wiki/Testing/Requirements
[8] http://w3c-test.org/
[9] http://mercurial.selenic.com/

- « Static » files committed to the Mercurial repository get automatically deployed to the test server, and immediately become available under http://w3c-test.org/[testsuite]/.

The automatic deployment feature only applies to static files: most test cases consist of HTML/CSS/JavaScript files and, given the use of a dedicated domain name, can thus be deployed on the server without undergoing a manual review.

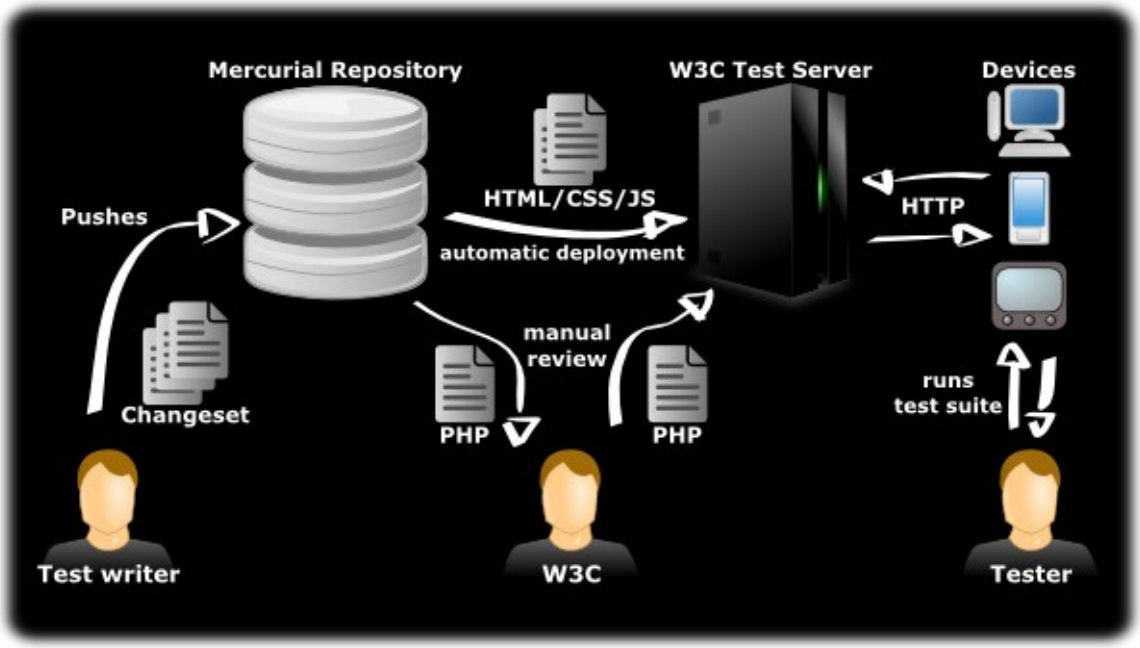

Figure 2 – Test case deployment on W3C test server

There are situations where test cases need a more complex server-side component. For instance test cases for the XMLHttpRequest specification developed by the Web Applications working group use PHP files to store user identifiers. These PHP files run server-side and de facto have access to internal server resources. To ensure the integrity of the server, these PHP files should not be automatically deployed when committed. They need instead to be reviewed by someone from the W3C team.

In collaboration with the systems team at W3C, the MobiWebApp project has deployed a review process and a supporting tool, whereby all static files get automatically published on the Web server while dynamic files are put on hold until someone from the team approves the files. In practice, whenever a user commits a test case that contains PHP files, an email alert gets sent to a dedicated team within W3C who must approve the files through a simple Web-based approval system. Once approved, the PHP files are deployed to the Web server. Figure 2 describes this workflow.

As of August 2011, the test server hosts 7 test suites of 4 working groups, including the test suite of the HTML5 specification, serving about 1200 different test cases. The repartition is described in Table 1.

Working Group	*Specification*	*Nb. test cases*
Device APIs WG	Contacts API	9
HTML WG	HTML5	924
Web Applications WG	DOM Core	80

Working Group	*Specification*	*Nb. test cases*
Web Performance WG	DOM Events	62
	Element Traversal	35
	XMLHttpRequest	70
	Navigation Timing	17

Table 1 – Test cases hosted on W3C test server (August 2011)

2.3 Initial Testing Framework

Based on the initial set of requirements, the testing team reviewed existing open source testing framework, looking into tools developed by working groups as well as external tools. The CSS testing framework [10] used by the CSS Working Group to maintain its extensive test suites for the CSS Level 2.1 specification (over 9000 tests) quickly stood out. It features a simple and easily extensible user interface and stores test results on a user agent per user agent basis.

Even though the codebase has been extensively updated since then, this framework is a derivative of the mobile test harness developed as part of the former MobiWeb 2.0 project.

Following discussions with Peter Linss, the developer and maintainer of the project, the testing team decided to use the CSS testing framework as a starting point for the generic testing framework.

The codebase got updated to drop dependencies on CSS specifications and move to a generic system that may handle any type of test suite. The result has been deployed on the W3C test server [11] as an alpha version. The first test suite to be incorporated is that of the Navigation Timing draft developed by the Web Performance Working Group. This particular test suite was chosen because:

- the specification is simple enough for a first trial
- the group had just started working on the test suite
- existing test cases so far are script tests
- test cases use the same JavaScript test harness as the one adopted by the HTML working group, the Web Application working group, and the Device APIs working group (the role of the test harness is to run a test case and report the result to the test runner).

In other words, the test suite is representative of the type of test suites that the testing framework needs to support.

Another early update was made to ensure the framework can act upon test case result reported by the script and record those results automatically. This means that, while preserving the possibility to handle other types of test cases, including self-describing tests, running the Navigation Timing test suite becomes a matter of clicking on a link.

As of August 2011, the testing framework is an alpha version. A lot of functionalities are missing or not yet implemented properly. The framework is not mobile-friendly for instance. Part of this is easy to solve: it is a matter of updating styles to enlarge links and buttons on touch screens. Part of this is harder: a test runner needs to embed test cases somehow, and embedding content in Web pages on mobile devices creates user interaction issues.

[10] http://test.csswg.org/
[11] http://w3c-test.org/framework/

Also, most other modules are missing, such as the test suite management tool which would de facto ease the integration of further specifications and test suites into the framework.

MobiWebApp contributed initial code updates to automate the collection of test case results when possible and will continue to take an active role in developments, since the testing framework is the main pre-requisite to the availability of test suites.

2.4 Test cases authoring

To promote the adoption of the test server and of test case formats that are compatible with the testing framework within the Device APIs Working Group, MobiWebApp developed the first 9 test cases for the Contacts API test suite.

3 NEXT STEPS

During the first year, MobiWebApp focused on preparing the creation of a formal testing activity in W3C, ensuring all required blocks were available: draft charters, actors, initial set of requirements, and starting point for the framework.

During the second year, MobiWebApp will focus on developing the testing framework, promoting its adoption in working groups, developing and gathering test cases for specifications that compose the Mobile Web Application platform and start to generate reports on the state of that platform based on test results.

3.1 Framework developments

As mentioned in Section 2, the testing framework that was deployed is but a starting point. Many features still need to be implemented. Priorities for MobiWebApp are:

- to ensure that the testing framework is mobile-friendly
- to automate the collection of test results as much as possible to avoid user interaction on mobile devices.
- to write documentation and guidelines to write test cases for the framework (that also work well on mobile devices)
- to develop missing components that would ease the adoption of the framework by working groups. Most of these components require to define some test metadata, so particular attention needs to be given to the list of metadata that needs to be submitted along with a test case (requiring too much metadata would negatively affect test authoring)

3.2 Increasing the Number of Test Cases

While developing the testing framework, MobiWebApp should also focus on increasing the number of test cases available in that framework. There are multiple directions that need to be run in parallel:

- Migration of existing test suites to the new framework. That would typically be useful for the test suite of the Geolocation Working Group.
- Gathering input from the community at large. The MobiWebApp project had a fruitful meeting with the MOSQUITO project back in May where people in MOSQUITO learned to write test cases for the new framework. The MOSQUITO project should submit test cases to W3C as a result. MobiWebApp and MOSQUITO are to organize a test-fest event in December where developers from the community will be invited to come and write test cases for W3C technologies that compose the mobile Web application platform.
- Developing test cases for specifications, typically to set working groups in the right direction when they start working on test suites, as was done for the Contacts API in the Device APIs Working Group.

3.3 Reports on the Mobile Web Application Platform

Once the testing framework is up and running, MobiWebApp should expose the results of the test suites in reports that feature the state of the mobile Web application platform. These reports should help create a virtuous circle where:

- The community at large benefits from contributing and running test cases because of the visibility they get on existing implementations. This is particularly relevant for mobile developers who need to be aware of interoperability problems on existing platforms.
- Browser vendors get alerted about interoperability problems and priorities by the community.
- The interoperability of implementations and the clarity of specifications are improved as problems get identified as early as possible.

CONCLUSION

The wide availability of test suites that can be run on mobile browsers requires working groups to take the mobile platform into account when developing test suites. In turn, the best way to ensure this happens is to provide working groups with a generic testing framework that works equally well on desktop and mobile devices and meets groups requirements. The goal is thus to create a virtuous circle where all actors benefit from the availability of a common testing framework and contribute test cases.

The scope of such a generic testing framework exceeds the objectives of the Interoperability Work Package of MobiWebApp, and the effort needs to be sustainable in any case. Thus, a formal testing activity needed to be created in W3C to ensure W3C Members endorse and contribute to the developments.

During the first year, MobiWebApp helped ensure that all required blocks were available for that activity to start on solid grounds:

- Draft charters were adjusted to include explicit mentions of mobile support.
- Mobile network operators were contacted on multiple occasions to check the adequacy of the testing activity as defined with their internal goal, and ask them to take an active role in the activity.
- Initial requirements refined the description of the testing framework.
- A Web test server (hosting 1200 test cases from 4 working groups) and a first testing framework was deployed.
- Developments on the testing framework started.

MobiWebApp now needs to focus on developing the testing framework and gather test cases.

Appendix 1: Requirements for Testing framework

This appendix is a reformatted version of the initial set of requirements prepared by MobiWebApp for the generic Testing framework that appears on the W3C Wiki. The requirements were extracted on 30 August 2011. The Wiki page may contain a more recent version:

http://www.w3.org/wiki/Testing/Requirements

Overview

The testing framework is used below to mean the whole W3C test-suite framework that is being considered. It consists of:

- a **Web test server** to serve test cases over the Web
- a **test runner** to run a series of tests and gathering results for all of them
- a **test case review mechanism** to ensure the correctness of submitted test cases
- a **test suite management system** to ease management
- a **reporting tool** to produce implementation and interoperability reports
- a **spec annotation tool** to assess some level of spec coverage

Requirements

1 Requirements for the testing framework

1.1 The testing framework must be intended for Candidate Recommendation and post Candidate Recommendation phases

The test suite should be suitable to evaluate if the spec is implementable, but it should also be used to promote interoperability.

This includes:

- testing of precise technical requirements such as parsing and validity rules
- testing of technical requirements that can only be tested in the context of other requirements.
- testing of more general requirements for specification conformance that cannot be evaluated with simply unit tests.

1.2 The testing framework must support simple and complex tests

It should be possible to run unit tests (e.g. testing the value of an attribute) as well as complex tests (e..g acid or stress tests).

1.3 The testing framework should be intended for user agent conformance testing

It may not be an immediate goal to perform user agent conformance testing, but the creation of a test harness naturally meets many of the requirements for this, and there is likely to be interest in using the test harness for this purpose.

1.4 The testing framework should help improve interoperability

While a W3C goal is to test specifications conformance, more important to the community may be interoperability testing. Knowing which user agents produce what results for a given test, regardless of specification requirements related to that test, allows identification of areas of generally consistent and generally inconsistent user agent behaviour.

1.5 The testing framework must distinguish the roles of test files, test cases, test suites, test results and provide respective repositories

The architecture must expose these classes even though some of these layers may be merged in practice to improve automation.

1.6 The testing framework must allow many-to-many relationships between test files, test cases, and test results

There should not be an assumption of one-to-one relationship between elements at the various layers. A given test case may require several test files. A given test file may be used by several test cases. A given test execution may be repeated by different users and results stored separately.

1.7 The testing framework must equally support test case metadata definitions in test files and external

To improve reuse of test files, test case metadata should be stored separately from test files when possible. Metadata stored within test files could also potentially introduce side effects on the test outcome.

Not withstanding the above, the harness must allow test case metadata to be included in test files as that can facilitate automation in various ways (authoring, review, execution).

1.8 The testing framework must be explicit about the test license

Contributors and users of the system must be clear about the license applied to content submitted to the repository.

1.9 The testing framework must allow for multiple test licenses

See multiple test licenses (http://www.w3.org/Consortium/Legal/2008/04-tessuite-copyright)

1.10 The testing framework must allow testing of different layers

For instance, network (HTTP, low bandwidth/latency, server throttling), syntax, DOM, layout model, rendering.

1.11 The testing framework must be able to serve test cases over the Web

See below for requirements for the Web test server.

1.12 The testing framework must use a decentralized version control system for test files and test cases

W3C uses Mercurial.

1.13 The testing framework must include a test runner

See below for requirements for the test runner.

1.14 The testing framework must provide a mechanism for test case review

See below for requirements for the test case review mechanism.

1.15 The testing framework must provide a user-friendly tool to ease test suite management

See below for requirements for the test suite management system.

1.16 The testing framework must provide a reporting tool

See below for requirements for the reporting tool.

1.17 The testing framework must provide "coverage" information

In order to know which areas of a spec are well-tested and hence have a sense for (an upper bound on) the completeness of a test suite as well as the areas where it would be most profitable to direct new testing effort, it would be beneficial to produce an annotated version of the spec that associates each testable assertion in the spec with a link to onr or more test cases for that assertion.

See below for requirements for spec annotation.

1.18 The testing framework must allow for direct contributions from external individuals or entities

The public at large should be able to submit test files, test cases, as well as test results.

2 Requirements for the Web test server

2.1 The Web test server must be able to run server-side scripts

The exact list of languages that the Web test server must support remains to be defined. PHP and Python should be available.

XMLHttpRequest, CORS, EventSource, HTML5, Widgets WARP, and WCAG will all need a setup like this.

Note: We support PHP on w3c-test.org. There is a builtin review process of the PHP code in the mercurial respository.

2.2 The Web test server should pull out content from test case repository automatically

Test cases submitted to the test case repository should appear automatically on the Web server, except for test cases that make use server-side scripting, which should first be approved for security reasons.

Also client-side test cases need pre-approval for several reasons; and the review status of test cases must be clearly indicated to the repository user.

2.3 The Web test server must run on a dedicated domain name

For security reasons, the server must use a dedicated domain name.

The W3C Web test server, launched in February 2011, uses w3c-test.org.

2.4 The Web test server should allow to tweak configuration settings on a per test case basis

For instance, the Web test server should leave full control over media types and charsets, e.g. through the use of .htaccess configuration files.

2.5 The Web test server may need to run additional libraries

Some test suites may require the use of specific libraries. For instance, to test the Web Sockets protocol and its client API, a Web Sockets library needs to be installed such as http://code.google.com/p/pywebsocket/

2.6 The Web test server must be available through different domain names

Different domains, e.g. http://foo.example.org vs http://bar.example.org, but also http://example.org vs http://example.invalid (different as far as http://publicsuffix.org/ is concerned)

W3C Web test server exposes the following domain names for testing purpose as of 2011-06-07:

- http://w3c-test.org/
- http://www.w3c-test.org/
- http://www1.w3c-test.org/
- http://www2.w3c-test.org/
- http://天気の良い日.w3c-test.org/
- http://élève.w3c-test.org/

2.7 The Web test server must be available through different ports

e.g. http://example.org:80 vs http://example.org:81

HTTP servers for w3c-test.org are available on ports 80, 81, 82, and 83.

2.8 The Web test server must be available through HTTPS

Different certificates may be needed, such as a certificate with Extended Validation and an invalid certificate.

With SSL support:

- https://www.w3c-test.org/

3 Requirements for the test runner

The test runner is responsible for running a series of tests and gathering results for all of them.

3.1 The test runner must support multiple test methods (including self-describing, reftest, and script)

The following test methods are considered.

i) Self describing

(also known as human or manual tests)

This is the most basic level. A human is provided with one or more test files and a corresponding test procedure (which may be included as part of the test files), and is asked to indicate if the test passes or fails. Ideally, we should avoid those types of tests as much as possible since it requires a human to operate. Some folks want to have a comment field as well.

ii) Plain text output

This is equivalent as doing saveAsText on two files and comparing the output.

iii) Reftest

Two pages are displayed and the rendered pages are compared for differences.

For comparison, we might be able to use HTML5 Canvas, or an extension to get screenshots. Worth case scenario is to use a human to compare the rendered pages.

See also

- CSS reftest convention (http://wiki.csswg.org/test/reftest)
- Creating reftest-based unit tests (https://developer.mozilla.org/en/Creating_reftest-based_unit_tests#title)

iv) Descriptive dump

Some engines could dump their in memory view/layout model, ie the one directly affecting the rendering.

v) Script

The test result is established through scripting:

- compare two DOM trees using Javascript for differences,
- test the result of a javascript function or attribute,
- etc.

We're looking at using testharness.js for those (see http://w3c-test.org/resources/testharness.js). Note that it doesn't preclude human intervention sometimes, such as authorizing geo information, pressing a key or a button, etc.

3.2 The test runner must be able to load tests automatically based on manifest files

Manifest files should contain the metadata necessary to load the tests (URI, type, etc.)

3.3 The test runner must be able to order test cases smartly

Purely automated tests should be grouped together to avoid a situation where the user is solicited on a random basis. This may be done when creating manifest files.

3.4 The test runner must allow for tests to be run in random order and repetitively

The goal is to detect failure under certain conditions

3.5 The test runner must allow for complete and partial execution of tests

Selection of subset can be based on the metadata describing the test; for instance, to select all tests that apply to a certain feature, element, or other aspect of the test.

3.6 It must be possible to create test runners that work on various platforms

Test runners should be available that work on main operating systems (e.g. Windows, MacOS, Ubuntu), most user agents, and on various types of terminals (e.g. desktop, mobile).

Some environments might require specific developments. For instance, on mobile devices, test suites might need to be splitted or packaged differently after a certain size to cope with the limitations of the platform.

This requirement might be met by providing different test runners for different environments.

3.7 The test runner must provide some way to output collected results

This might either take the form of a raw text file format, XML, JSON, or internal database storage.

3.8 The test runner must allow for automatic and manual gathering of context information

This context information includes the browser versions, the OS platform, as well as relevant configuration settings and assistive technology if applicable.

3.9 The test runner must include context information in collected results

Result records must be complete with information about the test case, the tester, the revision if applicable, the user agent, etc.

3.10 The test runner must support positive and negative testing

- It must be possible to define positive tests of specification requirements.
- It must be possible to define negative tests that actively test failure to meet specification requirements or test error handling behaviour.

3.11 The test runner must support testing of time based information

The requirement is needed for SVG animation, HTML video for instance.

3.12 The test runner must allow a test to report its result automatically

Some hook must be available so that automated tests can report their results without human intervention.

3.13 The test runner must allow humans to report on manual test outcome

There should be some pass/fail/unknown submission procedure available for manual tests.

3.14 The test runner must allow reftests to be run by humans

Even if reftests can be automated, the test runner should provide a way for humans to report on a reftest, possibly switching between test view and reference view several times per second and asking if the user sees flickering.

Automatic running of reftests requires browser-specific code and is explicitly out of scope.

3.15 The test runner should allow for humans to comment on a test outcome

Allows a text comment field for human evaluator notes (e.g. test conditions, failure notes) on the individual test result that can be included in the reporting. E.g. they might write: "the authoring tool implements this SC with a button that automatically sends the content being edited to the XXX Checker accessibility checking service".

3.16 The test runner must allow tests to be created on smaller tests

This would allow one action to be repeated several times within the same test, for instance to detect failure under certain conditions.

3.17 The test runner must be usable by external entities and individuals

Note though that some test suites may need specific conditions to run.

4 Requirements for the test case review mechanism

4.1 The test case review mechanism must enable review without putting a Working Group on the critical path for every single test

See the work of the WCAG 2.0 Test Samples Development Task Force (TSD TF) which included the development of a review process that allowed the Task Force to pre-review tests yet allow the Working Group to make the final decision.

We may also want to pursue public review and rating systems (though there are several concerns including critical mass to make the system useful, avoiding spam, avoiding disruptive or bogus entries.

4.2 The test case review mechanism must provide an easy way to submit a test

A Web author should be able to submit a test to the W3C. See also the Policies for Contribution of Test Cases to W3C (http://www.w3.org/2004/10/27-testcases).

4.3 The test case review mechanism must allow anyone to easily give feedback on tests

In particular, this should not be restricted to named reviewers or people with W3C accounts

4.4 The test case review mechanism should integrate with Mercurial

The distributed version control system should be used as much as possible.

5 Requirements for the test suite management system

5.1 The test suite management system must scale to a large number of tests

There may be more than 100,000 test cases per specification.

5.2 The test suite management system must track the state of test cases

Test cases may be:

- under review

- approved
- rejected

5.3 The test suite management system should allow association of a test case with issues, action items or mailing-list threads

Integration with W3C tracker tool?

5.4 The test suite management system should allow stable dated release of test suites

Test suite revisions will be used in particular to link back collected results to the appropriate versions of a test suite and to create snapshots when needed (e.g. for an implementation report).

6 Requirements for the reporting tool

6.1 The reporting tool must be able to produce a machine-readable report

The actual format needs to be precised. It could be XML or non-XML. The Evaluation and Report Language (EARL) provides a machine-readable format for expressing test results in RDF with an XML serialization, for instance.

The output should be reusable by other applications. It should also be usable to answer questions such as:

- Is feature X supported on Browser 4.3?
- What does Browser 4.3 support?

6.2 The reporting tool should be able to produce an agglomerated report

Multiple test results may be available for a given test case. The reporting tool should be able to combine them and report most likely test outcome.

6.3 The reporting tool should support authoritative result

When multiple test results for a given test case exist, there must be a mechanism to compare results and determine an authoritative results. This must be limited to privileged users.

7 Requirements for the spec annotation tool

7.1 The spec annotation tool must map each test case onto a part of the spec

In turn, this creates a requirement on the metadata test cases must define. The definition of "part" is up to the spec under test. It may mean:

- the section that contains the conformance statement
- the paragraph that contains the conformance statement
- the conformance statement itself

7.2 The spec annotation tool must react smoothly to spec modifications, deletions, insertions and rearrangements

A one-word update should not invalidate the mapping.

8 Requirements for test cases and test files

8.1 Test cases must not depend on the test runner

A test may be able to generate its result automatically (such as Script test) or not (such as Self describing test). If it is automatic, it is the responsibility of the test to report its result to the test runner above it using some hook. Otherwise, it is the responsibility of the test runner to gather the result from an alternate source (such as a human).

8.2 Test cases should be designed for multiple purpose

Test files and test cases should be designed as neutrally as possible so they can be repurposed. Multiple Working Groups may have reasons to re-use test files and should not be forced to create redundant versions. Even within a specification, a given test file may be used to test multiple things.

8.3 Test cases must have a unique ID

Test cases (and test files) must have a unique ID. A URI may be sufficient for test files. The ID should not be expected to contain metadata about the test in its lexical form, although as a convenience many IDs may have some structure.

8.4 Test cases must identify the relevant specification section(s) and/or conformance statement(s) under test

The targeted granularity may vary depending on the specification. For some specification, it may be enough to link back to the section that contains the conformance statement. For other specifications, a more precise link to the actual conformance statement may be needed.

Note a test file may apply to more than one specification.

8.5 Test cases may apply to the same conformance statement as other test cases

There may be more than one test case per conformance statement.

8.6 Test files may depend on other test files

Test files consisting of a single file (singleton test files) are preferred for simplicity and portability, but it must be possible for test files to have dependencies on external resources such as images, scripts, etc.

8.7 Test files may depend on shared resources

It must be possible for resources, such as images, scripts, etc., to be shared by multiple test files. The test file repository structure must accommodate actual "test files" as well as resources that are not themselves considered test files.

8.8 Test files may generate test files

Some of the test files may be generators for a collection of test files and test cases created e.g. by varying a single parameter.

Note this may interfere with the requirement for unique and constant identifiers for test cases.

9 Requirements and ideas not yet categorized

- allow more than one way to test functionality
- tests that require a top level browsing context
- be suitable for HTTP 1.1, HTML5, CSS 2.1, CSS 3, ES5, Web APIs (HTML DOM, DOM L2, Selectors, Geolocation, XHR, etc.), MathML 1.0, SVG 1.1, Web sockets Protocols, etc.
- ideally, the browser vendors should help us getting what we need to run the tests on their products.
- How can the framework help ensure the completeness of a test suite with regards to a particular specification?
- regroup a set of existing tests from different sources (DOM, CSS, SVG, HTML, etc.). Can we create a test runner to run them all? Is it possible to convert them?
- regroup the set of metadata needed/provided in the existing testing framework/tests.