Using Acceptance Tests to Validate Accessibility Requirements in RIA

Willian Massami Watanabe*, Renata P. M. Fortes and Ana Luiza Dias
Institute of Mathematical and Computer Sciences, University of São Paulo
400, Trabalhador São-carlense Avenue - Centro
P.O.Box 668. 13560-970 - São Carlos/SP, Brazil
watanabe_willian@yahoo.com, {renata, anadias}@icmc.usp.br

ABSTRACT

Accessibility stands as a quality requirement for Web applications. However, current accessibility automatic evaluation tools are not capable of evaluating DOM dynamic generated content that characterizes Ajax applications and RIAs - Rich Internet Applications. In this context, this paper describes an approach for testing accessibility requirements in RIA, by using acceptance tests. The authors had implemented a set of assistive technology user scenarios in the acceptance tests, in order to guarantee keyboard accessibility in web applications. As the scenarios were implemented as acceptance tests scenarios, they provide accessibility analysis over all layers of the software, from server-side to client-side implementations (JavaScript and dynamically generated DOM elements) in RIA. The test scenarios are automatically executed, and by doing so, fit the Continuous Integration process of constant delivery of new functionalities in Web projects.

Categories and Subject Descriptors

H.2.2 [**Design Tools and Techniques**]: User Interfaces; H.5.4 [**Hypertext/Hypermedia**]: User issues

General Terms

Design, Human Factors

Keywords

Web accessibility, Continuous integration, Acceptance testing

1. INTRODUCTION

Nowadays, the Web usage has been extended not only for the technical persons on the Computer Sciences area of work, but for all the population. These new users introduced the concept of the so called "Web 2.0", in which content is no longer proprietary and made available by specific stakeholders but decided over people and communities virtually present on the Web as users [21]. The communities brought new requirements to the Web which led to the development of Rich Internet Application - RIA, Social Web applications, e-businesses, among other kinds of services. These services have the potential to alleviate the world greatest tragedies, such as poverty, hunger, disease, violence, corruption, low-literacy; by providing people with an universally accessible repository of data, information and knowledge [7].

The term "Web 2.0" implies the use of evolved patterns into web applications development [14]. Its characteristics include: user as part of the authoring process, greater interactivity than before and a desktop like experience [9]. The greater interactivity and desktop like experience [24] are the result of a new set of technologies (DHTML, CSS, XMLHTTPRequest, DOM Events, among others) that allow the deployment of complete applications on the web architecture (RIA). As the interaction gets more complex, the user interfaces are also becoming more rich and interactive [15]. However the growth in interactivity is pushing the limits of assistive technologies. Changes and updates generated by JavaScript are not always presented to users interacting with the Web through assistive technologies. Traditional assistive technologies presents information, considering that the webpage follows a linearized structure. However, Ajax web applications break this assumption by allowing the webpage structure to be modified during the user interaction.

In order to address the accessibility issues presented by RIA, the WAI (*Web Accessibility Initiative*) established the ARIA (*Accessible Rich Internet Applications* [31]) specification. The goal of ARIA is to add semantic data into HTML elements to allow assistive technologies better present user interface components and dynamic updates in the document structure [15]. Although ARIA provides accessible solutions for RIA, there is no guarantee that these solutions will be designed in the final application [33]. Freire et al., for example, conducted a survey about accessibility awareness of people involved in Web development in Brazil [13] and web accessibility state of Brazilian Municipalities websites [11]. Their conclusions were that web accessibility is far from being actually considered in Brazil and that much work remains to be done. Developers are not always aware of the guidelines and therefore require tools and evaluation methodologies that help them while integrating accessibility solutions

*Software Engineer at Yahoo! Brazil

W4A2012 - Technical, April 16-17, 2012, Lyon, France. Co-Located with the 21st International World Wide Web Conference.

in their web applications.

These development issues are specially true considering Continuous Integration development environments that make use of automatic testing strategies for software requirements. Current accessibility evaluation and repair tools present limited crawling capabilities (analysing solely static HTML content) and inability to analyse dynamically generated DOM (*Document Object Model*) elements, necessary for evaluating RIA accessibility [26]. In this context, this paper reports on the development of an automatic accessibility evaluation strategy for RIA, based on acceptance tests. The proposal consist of a set of executable acceptance tests actions that simulate keyboard navigation in web applications. These actions are used to test the interface accordingly to the interaction model that is used by disabled users, realizing basic and standard navigation strategies (tabbing and arrow key presses navigation) in RIA. The approach is in the beginning of its development and do not evaluate all possibilities of interaction scenarios for assistive technologies users.

Next sections describe previous work on automatic evaluation approaches towards web accessibility (Section 2), the proposal of the paper based on the use of acceptance tests to evaluate accessibility requirements (Section 3), an approach for testing focus management in web applications (Section 4), a case study for evaluating the paper proposal (Section 5) and final remarks (Section 6).

2. WEB ACCESSIBILITY METRICS AND ARIA

The pre-eminent reference on web accessibility is the WCAG (Web Content Accessibility Guidelines) [18, 12]. The WCAG establishes a set of guidelines that address web accessibilities issues and provide design solutions for them [28, 29]. The current version of WCAG (2.0) was elaborated to be technologically neutral, being applicable to technologies available now and in the future [17]. The guidelines also provide objective testable criteria, that can be evaluated with a combination of automatic testing (using automatic evaluation tools, like Hera[1] and daSilva[2], and authoring tools like Bluefish[3] and Dreamweaver[4]) and human evaluation [23, 29].

Once a website is deployed, keeping track of its accessibility levels is of great importance. However, due to the nature of frequent updates that characterizes web applications, monitoring the evolution of accessibility require accurate metrics in order to avoid having an inaccessible application deployed [27]. In terms of the software engineering process, accessibility metrics may improve the accessibility of products or can be used as a way of introducing accessibility in early phases of the development [11]. Accessibility metrics frequently relay on the use of guidelines, such as WCAG, in order to calculate a quantitative value that represents the accessibility level of websites.

It is worth noticing that manual evaluations are costly in development environments in which the software updated and deployed frequently. Therefore, in these environments accessibility testings would rely mostly in automatic testing practices. Even though automated tests present advantages related to productivity and wide coverage of web pages, they rely on heuristics to determine violations of the guidelines [5]. Automated tests are not capable either of evaluating ARIA requirements imposed to "Web 2.0" applications, since they present limited crawling capabilities, analysing solely static generated HTML content, and its inability to verify dynamically generated DOM elements that are critical to RIA [26].

While considering the use of guidelines as an accessibility evaluation basis, some authors argue the validity of WCAG, in regards to its clarity and objectivity. Brajnik et al., for instance, reported that expert accessibility evaluators produced 20% false positive guideline conformance evaluation and miss 32% of the true problems of a website [6]. Kelly et al. also highlights the use of generic language and vague terms and definitions in the documentation that might hinder the guidelines objective [17]. In this context, the development process should consider the use of empirical methods, like **screening techniques** and **user testing** [5]. However, real users can be difficult to recruit to test all aspects of an evolving design [22], which is specially true in web applications development scenarios which require the constant management of software re-design and its deployment.

3. ACCEPTANCE TESTS FOR ACCESSIBILITY REQUIREMENTS

The high profile of Agile and *eXtreme programming* has popularized the concepts of Test-Driven Development [3] and Continuous Integration [10, 8]. These practices have been used to deliver functionality in the early phases of the Software Engineering process. Both concepts work towards "Web 2.0" design goals associated to the *perpetual beta* design pattern [25], by leading developers into an iterative process life-cycle based on feedback and re-design phases. In order to guarantee the stability of the evolving design and inclusion of new features, an automatic build is executed to identify errors in the software as quickly as possible. The execution of the automatic build characterizes Continuous Integration processes, in which developers are lead to integrate their work frequently, which implies in multiple integrations activities been done in a single day [10].

The automatic test build can be composed by unit-tests, integration tests, acceptance tests, among others [2]. For "Web 2.0" applications, the test results impact directly in the decision of whether a software should be deployed or not. If a report identifies errors the application is classified as unstable and the deployment is delayed until the build is fixed. If a report identifies no errors then a production deployment can be executed for delivering new functionalities to users. It is worth noticing, that these development scenarios rely solely in automatic tests strategies, since one iteration of this process can take less than one hour [20].

Accessibility is a quality attribute for web applications which impacts in the interaction strategies of disabled users. Since, users are responsible for the ultimate definition of the accessibility attribute in the web applications [19], an automatic accessibility evaluation tool could make use of the end-user perspective, while analysing the software. This statement led us into developing an automatic accessibility evaluation tool that uses Acceptance Tests to analyse websites accessibility.

Acceptance Tests are usually part of Continuous Integra-

[1]http://www.sidar.org/hera/index.php.en
[2]http://www.dasilva.org.br/
[3]http://bluefish.openoffice.nl/
[4]http://www.adobe.com/products/dreamweaver/

tion automatic builds and are defined as a formal test to determine if an application satisfies the acceptance criteria or not, and allow the customer to determine whether a software is acceptable or not [1]. Acceptance Tests operate in the highest layer of an application: the user interface [2]. They consider the user view of the software in order to verify quality attributes. Acceptance Tests compose the last phase of the software development process, and if they identify that the acceptance criteria are met, the software is ready to be released [19].

In this work, we use a behaviour strategy for the Acceptance Tests (*Behaviour Based Acceptance Tests* [16]) which examines the applications considering the external behaviour of the system [19]. In this strategy, the test cases reflect user requirements as *User Stories* [2]. It uses external interface assertions, while executing actions that represent real user interaction scenarios, that would be executed if the software was in production.

In order to use Acceptance Tests for evaluating accessibility requirements, it is necessary to identify a set of disabled users interaction actions frequently executed in web applications. These actions will compose acceptance test cases that represent disabled users websites usage strategies. Then, the test cases can be executed automatically to verify accessibility requirements in the software. If a test case reports a failure assertion, it means that the application is missing the implementation of an accessibility feature. And if a test case reports no failure assertions, it implies that the software is accessible for individuals that make use of the same interaction strategy that the test case represents.

As common disabled users interaction actions, we considered:

- Keyboard navigation using the TAB key, following focusable elements in the screen [30];
- Heading elements navigation for seeking content in a web page [32];
- *Landmarks* based navigation [31];
- Sequential elements navigation using the arrow keys.

It is worth noticing that an automatic accessibility evaluation approach based on Acceptance Tests, considers real usage scenarios and rely in the same interfaces and interactions that users will be faced with. Therefore, Acceptance Tests are capable of making test assertions on dynamic generated and updated content of webpages (considering changes and updates to the DOM structure, that characterizes RIA). This evaluation approach is illustrated in Figure 1 in comparison with HTML static evaluation approaches, which are not capable of analysing dynamic changes in the DOM structure of a webpage.

4. EVALUATING FOCUS MANAGEMENT

As an initial case study, we implemented a prototype tool which is capable of generating test cases for TAB keyboard navigation scenarios (navigating through focusable elements in the screen). This scenario consists of testing focus management in web applications which is an essential accessibility requirement for RIA [31].

Blind users rely solely on the keyboard device as navigation mechanisms. These users do not benefit from visual feedback, which is necessary for interacting with devices such as the mouse. Therefore, they navigate through links, input fields and other widgets using only the keyboard as input device and a voice synthesizer software as an assistive technology that reads text content inside webpages.

The focus event is generated in the DOM structure of web applications in order to identify interactive elements. In scenarios that users are not able of interact with the elements using the mouse, they must use the TAB key in order to search for the interactive elements. Every time a user presses the TAB key, a focus event is dispatched in an interactive element of the application. With that feature, users that rely solely on the keyboard as input device can search for interactive elements using the TAB key, and as soon they find the element they want to interact with, they can press the ENTER key to activate it (if the element is a link or button) or use the keyboard to input text into a textbox element.

When considering RIA, which is characterized by complex interaction mechanisms in web applications, the focus management requirement becomes even more critical, in order to guarantee that widgets and interface components are navigable by individuals that use the computer similarly to the scenario previously described.

The prototype was implemented using the Pyccuracy application[5]. Pyccuracy is a *Behaviour-Driven Development* software that assists the development process by allowing acceptance test cases to be written using natural language. With this approach Pyccuracy improves the visibility of the test cases, considering multi-disciplinary teams composed by developers, designers, product managers, in a way that every stackholder can understand the acceptance criteria by reading the test cases. A Pyccuracy test case script is presented next:

```
Scenario 1 - Going to the Flickr webpage
Given
    I go to ‘‘http://flickr.com/’’
When
    I wait for the page to load
Then
    I see ‘‘Welcome to Flickr...’’ title
```

Pyccuracy reads each natural language statement (defined as an action) from the test case and executes it in a web browser, in order to verify if the test case was successful (with no failed assertions) or failed (with failed assertions). The example test case previously specified (named *Scenario 1 - Going to the Flickr webpage*) represents a interaction scenario where the user accesses the home page of Flickr. The test case indicates that the user should insert the Flickr URL in the address bar of the browser (*I go to "http://flickr.com"* action). Next, the user should wait for the webpage to be completely loaded (*I wait for the page to load* action). And, finally, the user should verify that the title of the visited web page is "Welcome to Flickr..." (*I see "Welcome to Flickr..." title* action). It is worth noticing that the example test case makes two assertions that might imply failure in the scenario: the second action (*I wait for the page to load*) verifies if the webpage is completely loaded in a specific time interval (Pyccuracy standard interval is set to 30 seconds) and the third action (*I see "Welcome to Flickr..." title*) asserts

[5]http://pyccuracy.org

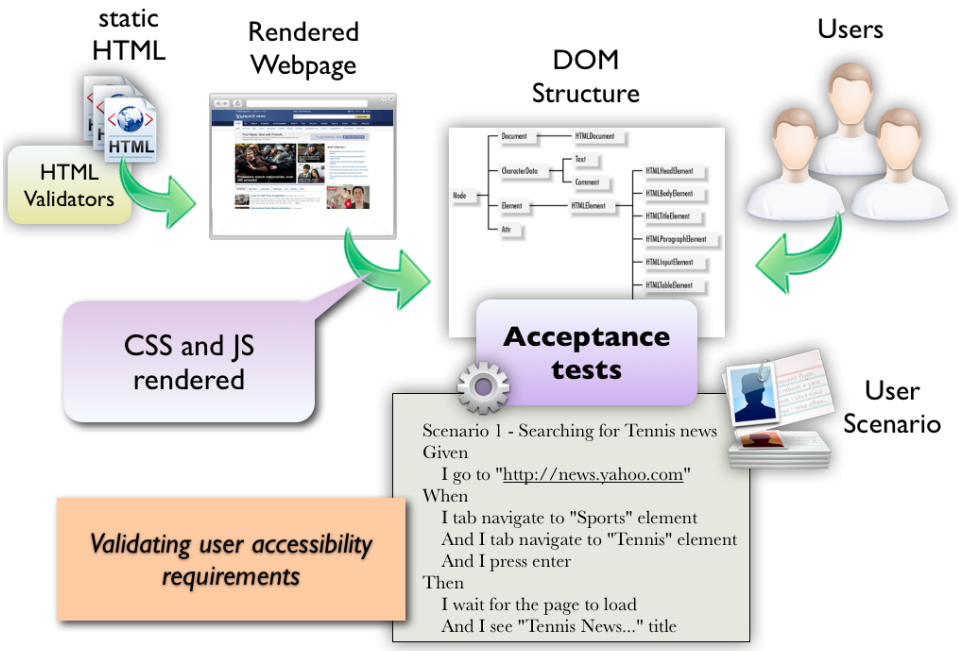

Figure 1: Acceptance Tests evaluation approach in contrast to HTML static evaluation approaches.

whether the title string of the visited webpage corresponds to "Welcome to Flickr...". In this context, the example test case would fail if the Flickr homepage did not load correctly (considering the second action) or if the title of the Flickr homepage does not match the string "Welcome to Flickr...". The first action (*I go to "http://flickr.com"*) establishes a simulation action, which compose the real usage scenario that the test represents.

In order to allow that Pyccuracy also analyses accessibility requirements that are associated to focus management in web applications, among its standard set of actions, we included two new ones. These new actions simulate disabled users that use the TAB key to navigate through interactive elements and make assertions about the website focus state, as the interaction is conducted. These actions are described next:

TAB key navigation : individuals who rely solely in the keyboard as input device for computers use the TAB key in order to search for interactive elements within webpages. As the TAB key is pressed in the keyboard, the focus of the application is moved between a set of interactive elements available in the web application. The TAB key navigation action implemented in Pyccuracy simulate the TAB ordering of elements[6] (elements in the DOM structure which havee TABINDEX attribute equal or greater than 0), moving the focus between these elements, respecting the order set by the TABINDEX attribute. This action search for elements in the DOM structure that are interactive and contain textual content or text alternative attributes that match one string given as input to the action. We present next the action usage inside a test case. In the example, the action statement execution by Pyccuracy search for an interactive element that contains the text "meme" using the TAB key navigation strategy. It is worth notiging that the action execution ignores elements in the DOM structure that are not perceivable by users (considering non-textual context, for instance), similarly to action execution by a disabled individual who interacts in the Web via Assistive Techonologies. The action execution asserts if a web page contains an interactive element with textual content that matches the inputed text ("meme", in the example). And if there is no interactive element with the inputed text, the action will report a failure in the test execution.

```
I tab navigate to ‘‘meme’’ link
```

Focused element activation : users that rely solely in keyboard as input device for computers, once they have set focus to the interactive element they are looking for, can use the ENTER key to activate this element. The activate action is similar to the visual interface event of mouse click (for BUTTON or link elements), however this operation requires that the interactive element which is been activated is the current focused one in the webpage. This action can also be used to submit HTML FORMS elements, if the focus is set to an INPUT element that is inside the FORM. Next, we present an example of the action. In the example, the action statement execution will first search for an interactive element that contains the text "submit". Next, the action execution will dispatch a ENTER keypress event targeting the current focused element. If the current focused element is a link, for instance, it will be activated and the user will be redi-

[6]http://wiki.codetalks.org/wiki/index.php/Docs/Keyboard_navigable_JS_widgets

rected to its referenced URL. It is worth noticing that this action does not make any assertion, however it simulates the navigation behaviour of users that use the keyboard in order to navigate in the Web.

```
I tab navigate to ‘‘submit’’ button
And I type enter
```

Next, we present a complete test case example to present how the actions are used. The test case example (named *Scenario 1 - Searching for Tennis news*) represents a scenario where the user searches an interactive element that contains the text “Sports” in the Yahoo! News Portal[7] webpage. If an interactive element containing the text “Sports” is found, the user will search for another interactive element containing the text “Tennis”. As that interactive element is found, the user will activate the element by pressing the ENTER key. After the execution of the activation action, the user should wait for a complete page reload. And, finally, it should be verified that the title of the newly loaded webpage is set to “Tennis News...”. It is worth noticing that, as the user find the interactive element containing the text “Sports”, the focus event dispatched in this element will trigger a DOM structure update. This update is associated to the presentation of the previously hidden interactive element containing the text “Tennis”. This interface component behaviour characterizes a **Fly-out menu** interaction design pattern[8] which is also addressed as a RIA drop-down menu widget [31]. Therefore this test case specifically describes how the use of Acceptance Tests to validate accessibility requirements can be used to automatically analyse RIA dynamic presentation, considering specifically the management of focus WAI-ARIA requirement. Figure 2 illustrates how the example test case is executed and interacts with the Fly-out menu interface component.

```
Scenario 1 - Searching for Tennis news
Given
    I go to "http://news.yahoo.com"
When
    I tab navigate to "Sports" element
    And I tab navigate to "Tennis" element
    And I press enter
Then
    I wait for the page to load
    And I see "Tennis News..." title
```

The prototype source code is available in http://github.com-/watinha/Pyccuracy-Accessibility-Actions.

Pyccuracy executes all actions that were specified in the test cases, and while executing them Pyccuracy identifies whether the action can be executed (if there are no failed assertions) or not (if there there is a failed assertion). If all actions are executed with no failed assertions, then the functionality (use scenario) the test case represents is classified as accessible for users that interact with the web application using the same navigation strategy (using TAB key presses to navigate through focusable elements). If any action generate a failed assertion, then the web application is not accessible for users that navigate using the same strategy the test case represents. A failure example for the test case (named *Scenario 1 - Searching for Tennis news*) would be raised if Pyccuracy, while running the test, did not find the interactive element containing “Tennis” string (considering the action *And I tab navigate to “Tennis” element*) after it had been given focus to the interactive element containing the string “Sports” (in the action *And I tab navigate to “Sports” element*). The failure example would have been caused by an error in the implementation of the keyboard navigation functionality of the Fly-out menu, in which the Fly-out menu only presents the menu sub-options if targeted by hover events, which are only available for users using the mouse.

It is worth noticing that the generated report of the Acceptance Tests is significantly different from the report generated by automatic accessibility evaluation tools that analyse only HTML static content. The Acceptance Tests are elaborated and executed considering the final user perpective of the software, simulating task-oriented usage scenarios written in natural language. Considering the webpage assessed by the example test case (*Scenario 1 - Searching for Tennis news*), automatic repair tools that analyse only static HTML, making use of no interaction scenario context information, are incapable of evaluating the presentation of the interactive element containing the text “Tennis”. The “Tennis” interactive element is only visible to users as the “Sports” element is focused, which characterizes an interaction scenario context that is not known if not through the scenario description in the Test Case. Therefore automatic repair tools that evaluate solely static HTML content are incapable of verifying this functionality. Acceptance Tests evaluation approaches are also capable of verifying all layers of the sofware (from client-side to server-side implementations).

Reports generated by Acceptance Tests also present the accessibility requirements failures and successes differently from other approaches. While HTML static automatic repair tools reports contains HTML code lines that are not conformance to WCAG or Section 508 guidelines, Acceptance Tests reports contain a list of usage scenarios that failed or succeeded. The usage scenarios failure means that the functionality is not accessible to users that interact with the applications with the same interaction strategy the scenario represents. The usage scenario success, on the other hand, means that the functionality is accessible to users that interact with the application using the same strategy the scenario represents.

Using Acceptance Tests to evaluate accessibility also contribute towards the design of accessible solutions by identifying accessible use scenarios for developers. This characteristic increase the implementation cost of the development process. Automatic repair tools that evaluate static HTML content do not require any setup or test case elaboration priorly to the evaluation. Acceptance Tests require that the test cases are written before the evaluation process could be run. The test cases frequently present domain specific information in their specification. Like the test case example *Scenario 1 - Searching for Tennis news* which requires domain-specific information such as the texts “Sports” and “Tennis”. However, the prior definition of the domain-specific content of the web application allow developers to analyse subjective accessibility criteria. Borodin et al., for instance, argues

[7] http://news.yahoo.com

[8] http://www.welie.com/patterns/showPattern.php?patternID=fly-out-menu

Figure 2: TAB navigation interaction schema for a Fly-out menu available at Yahoo! News homepage

that, although automatic static HTML evaluation tools are capable of identifying whether an IMG (image) element contains an associated textual alternative, they can not evaluate if this textual alternative actually describes the image they are associated with [4]. The prior elaboration of the usage scenarios of Acceptance Tests allow that the analysis of these subjective criteria be realized during its execution in the development process, by the developer or tester who is responsible for the test case implementation.

5. CASE STUDY

In order to evaluate the approach based on Acceptance Tests, we conducted a case study that compared how Acceptance Tests and HTML static automatic repair tools reported accessibility errors in a web application. The initial hypothesis of the case study was that, since Acceptance Tests are capable of evaluating dynamic content generation in webpages, they can analyse accessibility requirements that cannot be evaluated by HTML static evaluation tools.

It is worth noticing that the Acceptance Tests prototype implements is TAB navigation usage scenarios (implementing Acceptance Tests actions that search for interactive elements and activate them). Therefore, the case study metrics consider only accessibility reports related to user navigation and keyboard accessibility requirements.

In the next sections we present details about the case study methodology (Section 5.1), results (Section 5.2) and discussion (Section 5.3).

5.1 Methodology

In order to evaluate the accessibility reports generated from the Acceptance Tests approach (using the Pyccuracy set of accessibility actions) and the HTML static repair tools, considering only navigation and keyboard accessibility issues, we developed a set of webpages to be evaluated by both approaches.

Since this work aims at improving automatic accessibility evaluation tools in regards to RIA advancements, the set of webpages to be evaluated by the approaches contained widgets that implemented some sort of DOM structure updates as the users interact with them. These widgets implement the interaction design patterns of *Fly-out menu*, *Accordion menu*, *Overlay menu* and *Tabbed menu* defined by Welie[9]. The widgets implementation if completely available in http://watinha.com/pyccuracy_test_1/templates.

Each widget was implemented in more than one webpage, in a way that each widget had keyboard accessible and inaccessible versions in the webpages. The accessibility errors inserted in the inaccessible versions of the widgets were related to the use of interactive elements that are not capable of acquiring focus state or being activated via keyboard in the interface component. These problems are related to the guidelines 6, 9, 12 and 13 of WCAG 1.0 and guidelines 1.3, 2.1, 2.4 and 3.2 of WCAG 2.0. In this context, we filtered all report information generated from the evaluation approaches that were not related to these guidelines.

The distribution of webpages, widgets and accessibility errors associated to the interface component navigation are presented next:

- Webpage with a *Fly-out menu* widget that does not acquire *focus* events.
- Webpage with an accessible version of a *Fly-out menu* widget.
- Webpage with an *Accordion menu* widget that does not acquire *focus* events.
- Webpage with an *Accordion menu* widget that is not activated with ENTER *keypress* events.
- Webpage with an accessible version of an *Accordion menu* widget implemented with HTML heading elements (using TABINDEX attributes).

[9]http://www.welie.com/patterns

- Webpage with an accessible version of an *Accordion menu* widget implemented with HTML button elements.
- Webpage with an *Overlay menu* widget that does not acquire *focus* events.
- Webpage with an accessible version of the *Overlay menu* implemented with links.
- Webpage with an *Overlay menu* widget that is not activated with ENTER *keypress* events.
- Webpage with an accessible version of an *Overlay menu* widget implemented with HTML SPAN elements.
- Webpage with a *Tabbed menu* widget that does not acquire *focus* events.
- Webpage with a *Tabbed menu* widget that is not activated with ENTER *keypress* events.
- Webpage with an accessible version of a *Tabbed menu* widget.

It is worth noticing, that all widgets in the case study implement navigation mechanisms functionality in web applications. Therefore, we elaborated test case scenarios that consider these functionalities in our Acceptance Tests approach. The accessibility reports generated by the Acceptance Tests execution were compared with the accessibility reports generated by HTML static automatic repair tools.

As evaluation metrics between both approaches, we considered the number of correct and incorrect assertions raised in the reports. The assertions are represented as the final status of the report generated by each approach. For instance, if an evaluation approach identifies an accessibility error in the webpage and the webpage really contains an accessibility error in its implementation, then we count one correct assertion for that approach. If an evaluation approach identifies an accessibility problem in a webpage, however the webpage does not present any accessibility error in its implementation, then we count an incorrect assertion for that tool. On the other hand, if an evaluation approach does not identifies any accessibility error in the webpage, and the webpage does not contain any accessibility error in its implementation, then we count a correct assertion for the approach. If an evaluation approach does not identifies any accessibility error in the webpage but the webpage does present an implementation error, then we count an incorrect assertion for the approach. This evaluation metric was used in the study, considering Continuous Integration practices. In the development process, the automatic evaluation tools define the stability of the application and their results impact directly in the deployment and release schedule of the software. The results are reported as stable or unstable build, which defines is the applications is stable or not.

In the study we considered HTML static automatic repair tools the following applications: DaSilva (available at http://dasilva.org.br/), EvalAccess (available at http://sipt07-.si.ehu.es/evalaccess2/), WAVE (available at http://wave.webaim.org/) and fae - Functional Accessibility Evaluator (available at http://fae.cita.uiuc.edu/). These tools were selected for the study considering the set of automatic evaluation tools presented in the WAI website[10]. As selection criteria, we selected all tools that run in the web platform (that did not require installing the software) and manage to perform tests on all the webpages elaborated in the study.

[10]http://www.w3.org/WAI/ER/tools/complete

5.2 Results

In the case study, the HTML static automatic evaluation tools (DaSilva, EvalAccess, WAVE and fae) reported accessibility issues associated to the following accessibility recommendations:

- Create a logical navigation, considering the tabindex attribute, for links, form controls and objects: *checkpoint* 9.4 of WCAG 1.0 and Success Criteria 2.4.3 and 1.3.1 of WCAG 2.0.
- Divide blocks of information: *checkpoint* 12.3 of WCAG 1.0 and Success Criteria 2.4.1 of WCAG 2.0.
- Present coherent navigation mechanisms: *checkpoint* 13.4 of WCAG 1.0 and Success Criteria 2.4.3 and 3.2.3 of WCAG 2.0.
- Group related links: *checkpoint* 13.6 of WCAG 1.0 and Success Criteria 1.3.1 of WCAG 2.0.

The accessibility issues were reported by the HTML static evaluation tools as **errors** (considering design solutions that prevent groups of users from using the software) and **warnings** (considering design solutions that might prevent groups of users from using the software). It is worth noticing that the **warnings** reported by the HTML static evaluation tools are associated to *checkpoints* of WCAG 1.0 or Success Criteria of WCAG 2.0 that could not be verified by the tools and should be evaluated manually. In this study, both forms of reporting accessibility issues were counted as assertions.

The case study identified that the HTML static evaluation tools and Acceptance tests reported the following numbers as correct and incorrect assertions for each tool:

DaSilva : 5 correct assertions and 8 incorrect assertions.

EvalAccess : 7 correct assertions and 6 incorrect assertions.

WAVE : 6 correct assertions and 7 incorrect assertions.

fae report : 6 correct assertions and 7 incorrect assertions.

Acceptance Test : 12 correct assertions and 1 incorrect assertion.

The number of correct and incorrect assertions for each tool and evaluation approach is illustrated in Figure 3.

In order to improve the detail of the results, the number incorrect assertions for each tool and approach was divided in two groups: false positives, indicating if a tool or approach did not identified any issue in a webpage that contained accessibility flaws in its design; and false negatives, indicating if a tool or approach identified accessibility flaws in a webpage that did not contained any accessibility error in its implementation. These results are illustrated in Figure 4.

5.3 Discussion

Considering, initially, the number of correct and incorrect assertions made by each tool and approach, it can be observed that the Acceptance Tests presented the lowest number of incorrect assertions in the study. However, the

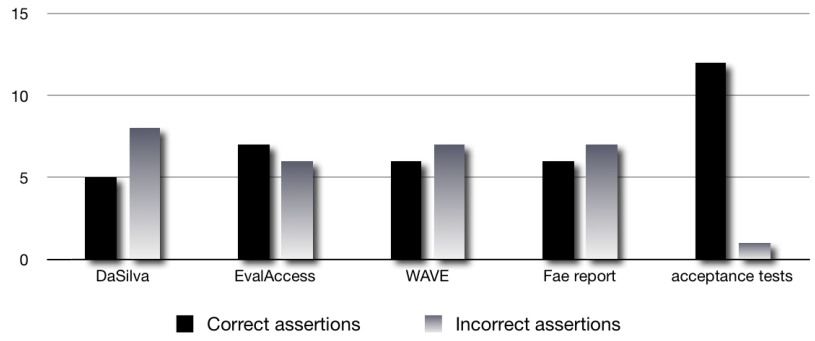

Figure 3: Numbers of correct and incorrect assertions for each evaluation tool and approach

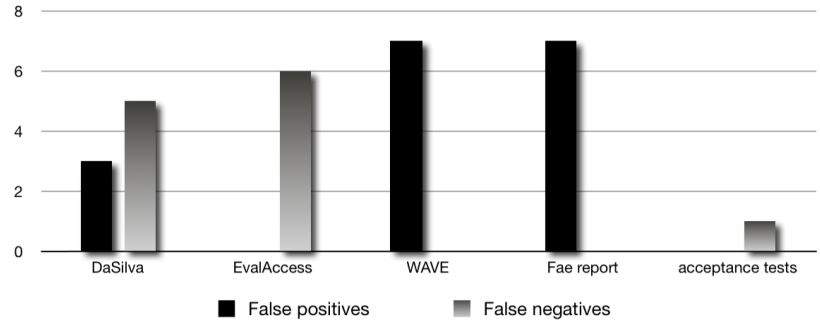

Figure 4: False negative and posivite numbers for each evaluation tool or approach

Acceptance Tests approach also demands more efforts, in order to be run, since this approach requires the elaboration of test cases priorly to the execution of the tool. On the other hand, in Continuous Integration development environments, the elaboration of test cases for Acceptance Tests is an inherent activity of the process and the inclusion of accessibility specific test cases would have a low impact in the cost for this activity.

It is worth noticing that correct and incorrect assertions for Acceptance Tests are dependant of what functionality the test case was elaborated to test. The simple fact that developers have written test cases for an specific webpage does not guarantee that the webpage itself is covered against accessibility errors. The Acceptance Tests are written accordingly to the user perspective and validate specific usage scenarios that could be executed by the user. Therefore, the test cases only cover the functionality that is specified in them, meaning that a single webpage would require one test case for each functionality it presents, in order to guarantee accessibility in the software. For the webpages used in the study, we elaborated test cases that made assertions in the exact functionality in which the accessibility errors were inserted. However, the authors do not feel the test cases elaboration could have biased the results of the case study. Since in Continuous Integration practices guide developers into the elaboration of the Acceptance Tests for all User Stories (that describe software core functionalities) that should be included within the software and, then, these functionality would certainly be covered by the Acceptance Tests approach.

The tools EvalAccess, WAVE and fae did not identified any distinction between webpages which presented accessible design solutions and the ones that did not. While the tools WAVE and fae did not report any type of error or warning (which can be verified in the high number of false positives assertions made by these tools), the EvalAccess tool presented accessibility warnings for all versions of webpages included in the case study. EvalAccess presented the highest number of false negatives assertions, numbers which represent all webpages that did not present any accessible design flaw in their implementation. In the Web 2.0 context, these tools would impact severely in the production deployment schedule of development processes. WAVE and fae tools, given the high number of reported false positives assertions, would not raise errors for accessibility design flaws like the ones used in the case study, and would increase the risk of deploying a software which is inaccessible. On the other hand, EvalAccess tool would not allow production deployments to be executed automatically, always requiring that accessibility evaluations were made manually (since all issues reported by EvalAccess consisted of **warnings**), even when the application was accessible.

It is also worth noticing, that all accessibility issues identified by the tools EvalAccess and DaSilva were reported as accessibility **warnings**, which means this issues would have to be manually verified by designers and engineers in order to grant the accessibility of the software. However this behaviour would characterize a semi-automatic evaluation method, differently from the automatic evaluation method based on Acceptance Tests proposed in this work.

6. FINAL REMARKS

This paper reported the development of an Acceptance Tests based approach for testing applications for accessibility. We described the implementation of a tool that runs test cases automatically and considers assistive technologies user scenarios for raising accessible design flaws. The approach assists the software development process by providing developers with an automatic tool that test all layers of the system (from client-side to server-side implementation), while assessing also RIA's dynamic behaviour and interaction complexity. As the solution is implemented as an automatically executable test, it fits the Continuous Integration software development process of constantly evolving web applications

The proposed solution is not capable of scaling as HTML validators, since it requires the prior elaboration of test cases in order to be executed. It would be necessary a specific test case for each functionality in the application. In a menu widget, for instance, it would be required that each menu item is properly tested in a single test case. However, the activity of elaborating acceptance tests is already part of a web application development in Agile methodologies (specially in long-term projects, as web applications).

The solution is under development, and require improvements related to addition of other keyboard navigation actions (headers navigation, for instance) and analysis for the inclusion of a automatically ARIA roles, properties and states validation functionality. It is worth noticing that the proposed approach alone is not enough to guarantee that ARIA roles, states and properties are been correctly placed in the dynamic elements of the webpage. In order to identify elements which require ARIA markup, the author suggests associating Actions execution in Acceptance tests with DOM Mutation Events that were triggered during the action runtime. However further studies are required in this area.

Other future works include: analysing how the assistive technology user scenarios differ from actual users, through usability testing, and comparing the results of this approach with the ones obtained from traditional automatic conformance evaluation tools and metrics currently available.

7. ACKNOWLEDGMENTS

Our thanks to FAPESP (process 2010/05626-7) and CAPES for supporting this work.

8. REFERENCES

[1] IEEE standard for software verification and validation plans. Technical report, 1986.

[2] B. C. Araújo, A. C. Rocha, A. Xavier, A. I. Muniz, and F. P. Garcia. Web-based tool for automatic acceptance test execution and scripting for programmers and customers. In *Proceedings of the 2007 Euro American conference on Telematics and information systems*, EATIS '07, pages 56:1–56:4, New York, NY, USA, 2007. ACM.

[3] K. Beck. *Test-driven development : by example*. Addison-Wesley, Boston, 2003.

[4] Y. Borodin, J. P. Bigham, G. Dausch, and I. V. Ramakrishnan. More than meets the eye: a survey of screen-reader browsing strategies. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*, W4A '10, pages 13:1–13:10, New York, NY, USA, 2010. ACM.

[5] G. Brajnik. Beyond conformance: The role of accessibility evaluation methods. In *Web Information Systems Engineering - WISE 2008 Workshops*, volume 5176 of *Lecture Notes in Computer Science*, pages 63–80. Springer Berlin / Heidelberg, 2008.

[6] G. Brajnik, Y. Yesilada, and S. Harper. Testability and validity of wcag 2.0: the expertise effect. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*, volume 1 of *ASSETS '10*, pages 43–50, New York, NY, USA, 2010. ACM.

[7] S. Bratt. Breaking barriers to a read/write web that empowers all. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*, W4A '10, pages 1:1–1:1, New York, NY, USA, 2010. ACM.

[8] F. Cannizzo, R. Clutton, and R. Ramesh. Pushing the boundaries of testing and continuous integration. In *Proceedings of the Agile 2008*, pages 501–505, Washington, DC, USA, 2008. IEEE Computer Society.

[9] M. Cooper. Accessibility of emerging rich web technologies: web 2.0 and the semantic web. In *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, W4A '07, pages 93–98, New York, NY, USA, 2007. ACM.

[10] M. Fowler and M. Foemmel. Continuous integration, http://www.martinfowler.com/articles/-continuousIntegration.html, 2005.

[11] A. P. Freire, R. P. M. Fortes, M. A. S. Turine, and D. M. B. Paiva. An evaluation of web accessibility metrics based on their attributes. In *SIGDOC '08: Proceedings of the 26th annual ACM international conference on Design of communication*, pages 73–80, New York, NY, USA, 2008. ACM.

[12] A. P. Freire, R. Goularte, and R. P. M. Fortes. Techniques for developing more accessible web applications: a survey towards a process classification. In *SIGDOC '07: Proceedings of the 25th annual ACM international conference on Design of communication*, pages 162–169, New York, NY, USA, 2007. ACM.

[13] A. P. Freire, C. M. Russo, and R. P. M. Fortes. A survey on the accessibility awareness of people involved in web development projects in brazil. In *W4A '08: Proceedings of the 2008 international cross-disciplinary conference on Web accessibility (W4A)*, pages 87–96, New York, NY, USA, 2008. ACM.

[14] J. Gehtland, D. Almaer, and B. Galbraith. *Pragmatic Ajax: A Web 2.0 Primer*. Pragmatic Bookshelf, 2006.

[15] B. Gibson. Enabling an accessible web 2.0. In *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, W4A '07, pages 1–6, New York, NY, USA, 2007. ACM.

[16] P. Hsia, J. Gao, J. Samuel, D. Kung, Y. Toyoshima, and C. Chen. Behavior-based acceptance testing of software systems: a formal scenario approach. In *Computer Software and Applications Conference, 1994. COMPSAC 94. Proceedings., Eighteenth Annual International*, pages 293 –298, nov 1994.

[17] B. Kelly, D. Sloan, S. Brown, J. Seale, H. Petrie, P. Lauke, and S. Ball. Accessibility 2.0: people, policies and processes. In *W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, pages 138–147, New York, NY, USA, 2007. ACM.

[18] B. Kelly, D. Sloan, L. Phipps, H. Petrie, and F. Hamilton. Forcing standardization or accommodating diversity?: a framework for applying the wcag in the real world. In *W4A '05: Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*, pages 46–54, New York, NY, USA, 2005. ACM.

[19] H. K. N. Leung and P. W. L. Wong. A study of user acceptance tests. *Software Quality Control*, 6:137–149, October 1997.

[20] F. Matheson. Designing for a moving target. In *Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles*, volume 1 of *NordiCHI '06*, pages 495–496, New York, NY, USA, 2006. ACM.

[21] M. Naftali, W. Watanabe, and D. Sloan. W4a 2010: a web accessibility conference report from the google w4a student award winners. *SIGWEB Newsl.*, pages 1:1–1:5, September 2010.

[22] J. Nielsen and R. L. Mack, editors. *Usability inspection methods*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[23] L. G. Reid and A. Snow-Weaver. Wcag 2.0: a web accessibility standard for the evolving web. In *W4A '08: Proceedings of the 2008 international cross-disciplinary conference on Web accessibility (W4A)*, pages 109–115, New York, NY, USA, 2008. ACM.

[24] P. Thiessen and S. Hockema. Wai-aria live regions: ebuddy im as a case example. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*, W4A '10, pages 33:1–33:9, New York, NY, USA, 2010. ACM.

[25] J. M. Umbach. Web 2.0 - the new commons. Feliciter. Canadian Library Association. HighBeam Research., January 2006.

[26] C. A. Velasco, D. Denev, D. Stegemann, and Y. Mohamad. A web compliance engineering framework to support the development of accessible rich internet applications. In *W4A '08: Proceedings of the 2008 international cross-disciplinary conference on Web accessibility (W4A)*, pages 45–49, New York, NY, USA, 2008. ACM.

[27] M. Vigo, M. Arrue, G. Brajnik, R. Lomuscio, and J. Abascal. Quantitative metrics for measuring web accessibility. In *W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, pages 99–107, New York, NY, USA, 2007. ACM.

[28] W3C. Web content accessibility guidelines 1.0. W3C Recommendation, May 1999.

[29] W3C. Web content accessibility guidelines (wcag) 2.0. W3C Recommendation, December 2008.

[30] W3C. Wai-aria 1.0 authoring practices - an author's guide to understanding and implementing accessible rich internet applications. W3C Working Draft, September 2010.

[31] W3C. Accessible rich internet applications - (wai-aria) version 1.0. W3C Candidate Recommendation, January 2011.

[32] T. Watanabe. Experimental evaluation of usability and accessibility of heading elements. In *W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, pages 157–164, New York, NY, USA, 2007. ACM.

[33] W. M. Watanabe, D. F. Neto, T. J. Bittar, and R. P. M. Fortes. Wcag conformance approach based on model-driven development and webml. In *Proceedings of the 28th ACM International Conference on Design of Communication*, SIGDOC '10, pages 167–174, New York, NY, USA, 2010. ACM.