San Jose State University
Computer Engineering Department

Accessibility Defect Reclassification in Automation Testing

CMPE 172/272 - Enterprise Software Overview
Instructor: Rakesh Ranjan

Submission Date: May 9, 2011

Submitted By:
Meg Genoar, 004932292
Laurie Marmon, 007503756
Foram Patel, 007432464

Abstract

Enterprises can see the benefits of implementing accessibility and automation test separately, but consideration should be made to put these efforts together for increased gain. While accessibility provides a framework for using automation tools unintentionally, it is a means for gaining the needed ability to perform automation testing on a software product. However, the use of this framework presents some limitations related to a company's development decisions in how they implement identification tags within the layer. Through the authors' own experience in software automation test with Selenium, recent capabilities in today's automation tools and future opportunities in user interface (UI) tools being researched, the team offers a need for a reprioritization of defects related to accessibility implementation. Only through a joint effort between developers and testers can accessibility be truly beneficial to automation testing through decreased maintenance cost and time.

1 Introduction

Testing is an essential phase of the software development process. All enterprises, in order to create a working product, must put their products through rigorous testing before it is released to the public in order to minimize user-facing defects. This is especially true in complex systems. However, software testing takes both time and resources that can add to development costs quickly. And even if testing has been identified as necessary to the development process, there exists the pressure to minimize the cost while still producing quality in the product.

In order to accomplish these goals, software test engineers commonly look towards automation as a means of decreasing the overall cost and time of the testing effort, especially in the case of applications with a user interface (UI) [1]. With many automation tools on the market, testers have a vast selection to choose from. But using many of these solutions requires embedded test hooks be developed within the product. This need can be addressed by introducing the commonly overlooked accessibility layer. Implementing accessibility in their products is something enterprises are starting to realize may be the answer to solving an automation testing need. With the potential of achieving automation testing and access for all, the benefits make it a worthwhile opportunity to explore.

The concept of accessibility within software solutions is not new. Helped by legislation amending the Rehabilitation Act in 1998, Section 508 started a bigger push for software development firms to start thinking about accessibility and its effect on their products. While [2] mandates accessibility is required for all electronic and information technology solutions in use by the federal government, companies are beginning to realize the

importance of striving to make their products compliant voluntarily. Additionally, [3] points out that using the robust accessibility model for development can benefit the end customer in many ways, regardless of the presence of a disability usually resulting in the form of a better user experience for all.

Beyond this, the adoption of accessibility within a software product has brought benefits to those who test the software as well. Realized first in the creation and use of validation checkers developed for the sole use of testing Section 508 compliance, now software test engineers are beginning to think of what else accessibility and automation testing can accomplish together [4]. While these test tools were fairly limited in their scope of testing, the exercise of their creation and use made test engineers see the potential of the accessibility framework. As a dependable and robust means of access through accessible application programming interfaces (APIs), individual UI elements could be accessed following an expected control pattern simulating human interactions [5]. Teamed up with automation tools such as Selenium and SilkTest, automated testing using the accessibility layer can go beyond the capabilities of validation checkers and into testing some of the functionality behind a piece of software too.

This move to repurpose the framework's capabilities makes putting accessibility in a company's software even more worthwhile. Even the most simple implementation mechanisms for accessibility compliance can be used for the basis of automation testing. In the case of enterprises, this two for one deal makes the initial cost of implementation lead to a good chance to make a reasonable return on their investment in the form of software correctness and accessibility for all.

2 Background

Adding accessibility into software products has been helped with the efforts of major software development companies on a variety of platforms. For "even though there is not standard for accessibility API calls, different technologies offer similar [APIs], suggesting slow convergence towards a common programming standard for accessibility technologies" [6]. This similar implementation has formed a robust foundation upon which automation testing tools can depend.

For the desktop and later for web applications, Microsoft was one of the first to develop an accessibility API for developers making programs for their Windows operating system. Introduced with Windows 95, Microsoft Active Accessibility (MSAA) offered the ability for communication between the OS and assistive technologies (ATs), such as screen readers or braille displays, in the marketplace used to aid people with various disabilities [7]. By implementing MSAA into their software, enterprises allowed ATs to

access their own custom UI they developed for the Windows platform, increasing the number of people able to participate in the computing age through their products.

Microsoft later updated its accessibility API by introducing UI Automation (UIA) in 2005, which is still in use today [8]. UIA made some fundamental changes to the MSAA API that were required to keep up with the ever more complex controls and mechanisms found in today's software [7]. Today, UIA is considered both by developers and test engineers for its potential in automation possibilities beyond those of just accessibility compliance.

The underlying model used by these two Microsoft APIs makes the connection between automation testing and accessibility possible. Based on representing UI elements in a hierarchical tree structure, UIA, and MSAA before it, achieved the means for accessing these elements individually through software [8]. Termed programmatic access, it has been one of the key foundations in letting AT devices and automation test tools access the inner functionalities within a piece of software with a UI component [8] [9]. This programmatic access is achieved through the client-provider model of development with UIA, shown in Figure 1.

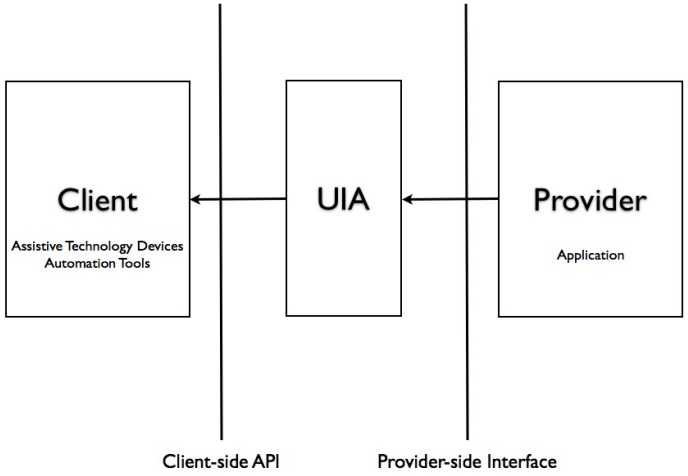

Figure 1. Client-Provider architecture block diagram. (Reproduced from [9])

In this model, the client can take on many forms, such as an AT or test script invoked by an automation tool, while the provider is the application with accessibility support implementation built in through the provided UIA interface. Through the UIA API, any client can request the accessibility information of the application's UI elements and then internally interpret this information for the task it was designed to do [9] [5]. In the case of a screen reader, the UIA element information will be used to speak through the interface elements to the user. Alternatively, if a test script takes the role of the client, the accessibility information can be interpreted into the computer's next navigational

move within the script. Regardless of the type of client accessing the system, it depends on the implementation put in place by the original developer to provide descriptive and correct information about each individual UI component.

A further breakdown of the accessibility model put in place with implementing UIA's API shows how UI elements are organized as nodes of the tree structure. Within the tree, UI elements are able to have both parents and children as a way to describe the entire UI [7]. Each node on the tree is an element that has a unique identifier, control type, and name to further describe the functionality behind the UI element [9]. This expected structure to UI elements is helpful for simulating keypresses and mouse movements, which both disabled users and automation tests can use to their benefit to move around an application.

Desktop software solutions are not the only ones that have a need for accessibility. The rise in smartphone adoption has meant a part of the accessibility efforts must also be considered for these new platforms. Both Apple's iOS and Google's Android are starting to bring over what has been done on the desktop to that of the mobile device. Apple's own version of a UI Automation framework was introduced in the release of iOS 4. A solution done in JavaScript, it provides developers with the first opportunity to officially test their accessibility and functionality programmatically with the ability to simulate touches on the touch screen [10]. This introduction was received well by developers using the iOS platform because of this added testing capability.

Whether on desktop or the new mobile platforms, the groundwork for implementing an accessibility framework has been put into place by major OS software companies. What they provide has made accessibility a reality on almost any platform an enterprise develops their solution for. In the case of automation tools, the same holds true. The tools developed upon the accessibility framework can fit in a variety of situations with different platforms and supporting languages.

3 Related Work

Many of the solutions that are in use today, as well as those that are in development through research, take the client-provider model of UIA as a foundation. Solutions such as Selenium, QuickTest Pro, Rational Functional Tester and SilkTest started to take advantage of the hierarchical tree structure when MSAA first came out [7]. Now with UIA only enhancing this interface, the capabilities of these tools in accessing UI elements have expanded even further.

There are a variety of automated testing tools available and suitable for enterprise use. The goal of these tools is to create and run test scripts for websites and web-based applications. This can be done in two ways: direct script editing done by the tester or UI activity recording [11]. Using an automated testing tool, such as Selenium, a tester is able to easily create and import test scripts that can be run in a web browser. While writing scripts for UI testing, test engineers can take advantage of the accessibility information present within the web page or application through the tool. By performing the desired execution stop, the script is recorded into the tool's interface for later execution reference. Additionally, a tester can also use a mouse and/or keyboard to navigate his or her way through a desired test case while the mouse actions and keystrokes are recorded. The UI activity can be seen in the tool's IDE, and a test script will be generated automatically [11]. The language used for either testing method works well with a variety of developers' skill sets, as the popular testing tools support most practical scripting languages [12].

The general scope of the most widely used automation testing tools is the same across all of the tools. Each tool allows test engineers to record keystrokes and run test scripts, and each also supports various languages. However, some of the popular tools have some features that are unique to themselves. For example, SilkTest records not only mouse clicks, but any mouse movement. IBM's Rational Functional Tester includes version control and syncability for testers working separately. TestDrive includes a scheduled playback feature, allowing testers to run scripts at any time, whether or not they will be at the computer at that time [13]. The success of these products has made them industry standards for testing the accessibility, functionality and usability levels of many websites.

But with any automation tool process comes the need to perform maintenance on the resulting scripts at some point in the testing cycle. It turns out that the accessibility layer implementation can be utilized for this type of exercise in software testing as well. Current research such as the REST tool presented in [1] tries to alleviate the time it takes for test engineers to keep regression test suites up to date for every new version. This is done through a process of analyzing the structure changes between the two versions, determining the affected components, identifying test script inconsistencies and inferring the next steps to make the suite compatible. This data is held in an object repository to match UI elements to the descriptions representing them in the scripts [1]. As a possible way to shorten the time test engineers spend in determining compatibility issues, the solution seems promising.

Another automation testing solution proposed in [6] tried to extend the reaches of programmatic access. Rather than just conceding that the accessibility layer was a

necessary component in making automation testing possible, the authors wanted to also put it to work in generating its own test cases. Using reference and target applications, the “Smart” system introduced a means for test engineers to reuse some of the test cases they had created for the referenced legacy application while performing the generation of the test suite for the new target [6].

What both of these solutions have in common is the interest in making the accessibility layer of implementation work for the test engineers. By decreasing the automation test generation time in the case of different versions of UI applications, the need for additional effort in test maintenance had also decreased. This also makes the initial time it takes to create the test suite retain and increase the return on investment value for the company.

But these research solutions also bring to light the need for an accessibility layer implementation that includes meaningful and descriptive help tagging to individual UI elements. Besides its use in describing a UI control to a disabled user, descriptive tagging has the ability to make test engineers looking at an automation script out of context understand it with greater ease. This underlying maintenance issue is what has brought about the idea and solution described in the next sections.

4 Problem Statement

Just adding accessibility into a software product is not something to be taken lightly. In many cases, the process to make it accessible needs to go beyond simple compliance to accessibility laws and guidelines. Instead, accessibility needs to become a part of the formal software development processes already put in use by the enterprise [14]. This means the skills of implementing accessibility are needed at all levels of the software development life cycle [15] [16]. Unfortunately, the accessibility implementation phase is more of an afterthought for many, which increases the risk for failure and increased cost that can happen at any level [17].

As one of the basics in making software accessible, the inclusion of identification tags proves to be an easy first step companies take in adding accessibility to their products. An identification tag, also called assistive information, is additional information inserted within the HTML markup to further describe the UI control’s functionality and purpose [18]. With the help of some development environments, programmatically adding these tags is possible without much effort [19]. But this feature is not to be counted on for everything, particularly custom controls. The added custom functionality they provide commonly produces instances that need additional thought and design consideration in

order to make them truly accessible. In these situations incorporating the needed identification tagging for these kinds of controls falls solely on the developer.

While basic accessibility can be achieved by simply adding a tag, making that tag useful to the user takes analysis and design on how the UI element will be used within the application. These tags propagate to both the end user through the use of an AT reading it aloud and to software test scripts which record the tag information as identification of the action it is to take. Thus, a developer plays an important role in determining whether the identifying information they provide to UI elements are both accessible and functional for end users.

5 Observations

As one of the end users to the accessibility implementation put in place by developers, software test engineers are one group that needs to be familiar with identifying tags describing UI controls. This access also has the potential to offer times when this descriptive information is not the most ideal. The lack of a clear description for the control can lead to confusion on behalf of the tester or down the line, to the disabled end user accessing the content by an AT.

These types of discrepancies can occur in a variety of situations. While performing testing upon a web application, this reoccurring theme within the website formed the basis for the exploration into the importance of correct identification tagging processes for automation script use. The following observations used a student created website [20] while performing testing with the testing tool Selenium.

The first issue noticed in testing was how the correlation of certain elements did not seem to match the given visual functionality description. This first instance of note is shown in Figure 2 below. Within this figure the part to note is how Selenium saves an action that is common for many websites. In this case the issue is how “login” turns into merely “save” in the Selenium script.

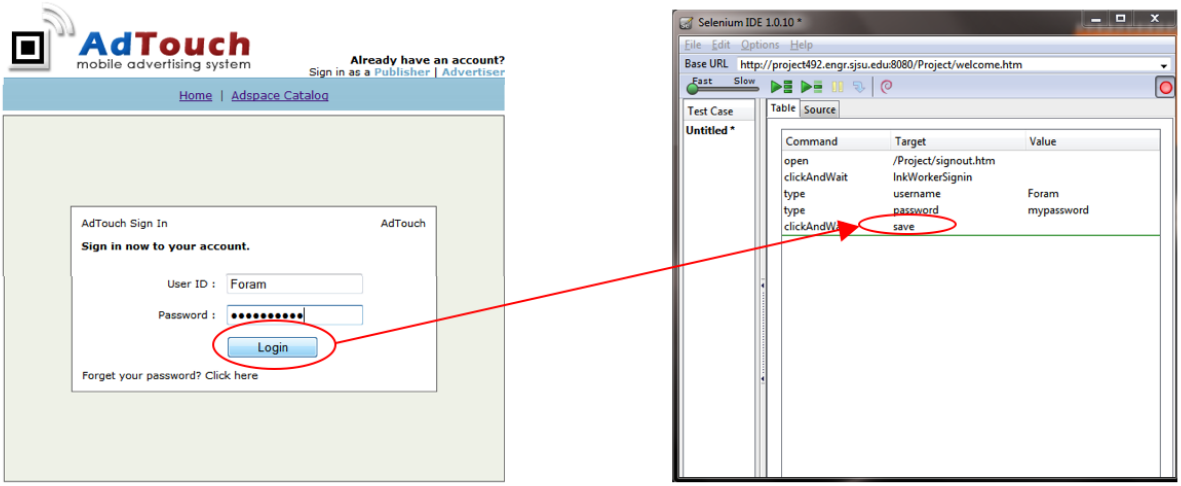

Figure 2. Discrepancy for the common script action to login.

In the context of accessibility, this type of interaction would be deemed accessible by validation checker standards. It has the necessary identifying information required for an AT to access it, but for the software tester, the problem comes up when this script needs to be maintained later. Without the information of where in the website the script step was performed, simply seeing “save” could mean various actions, which could or could not be remembered as a “login” action.

Another similar instance of tag mismatch was seen in the same site for the case of signing in and signing out. Figure 3 shows the website link and Selenium corresponding script step capture.

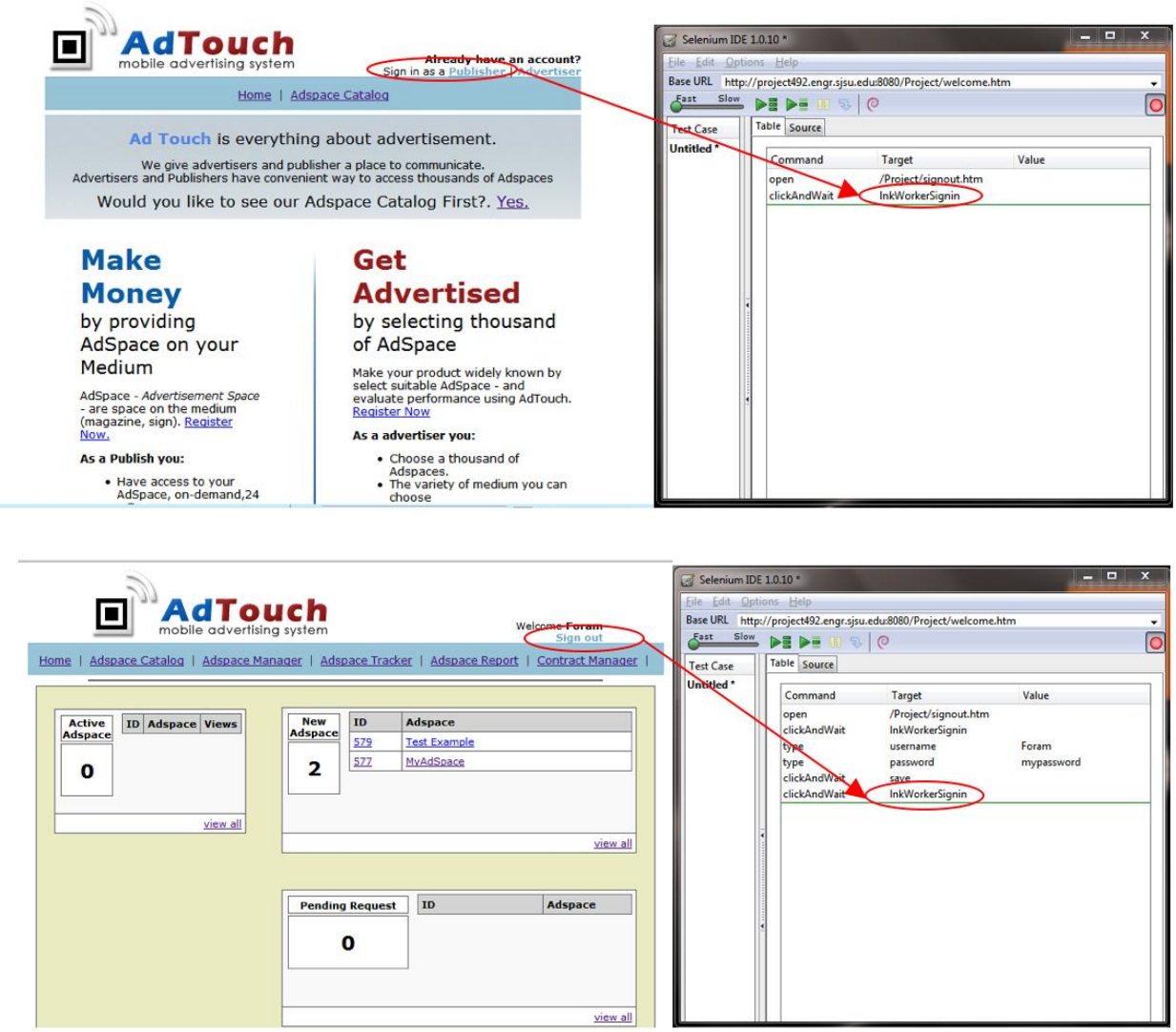

Figure 3. Signing in and out of the AdTouch system.

The difference in this situation exists in how two completely different actions are given the exact same identifying name. “lnkWorkerSignin” in itself does not directly correspond to either the “Publisher” or “Sign Out,” but also does not match the latter’s function behind selecting the link. Again, this case shows how the links are accessible from a compatibility standpoint, but it is not the optimal implementation.

For further consideration, a situation like what is seen for the signin and signout functionality makes it difficult for automation test script logic to be created as well. While Selenium is able to pick up the tagging information about these links, performing logic to determine whether a user needed to sign in or not during the course of the script becomes difficult. The ever-present “lnkWorkerSignin” UI control name is misleading to this determination.

Finally, one last situation encountered dealt with action buttons in the UI being referenced by the same name. This is shown in Figure 4.

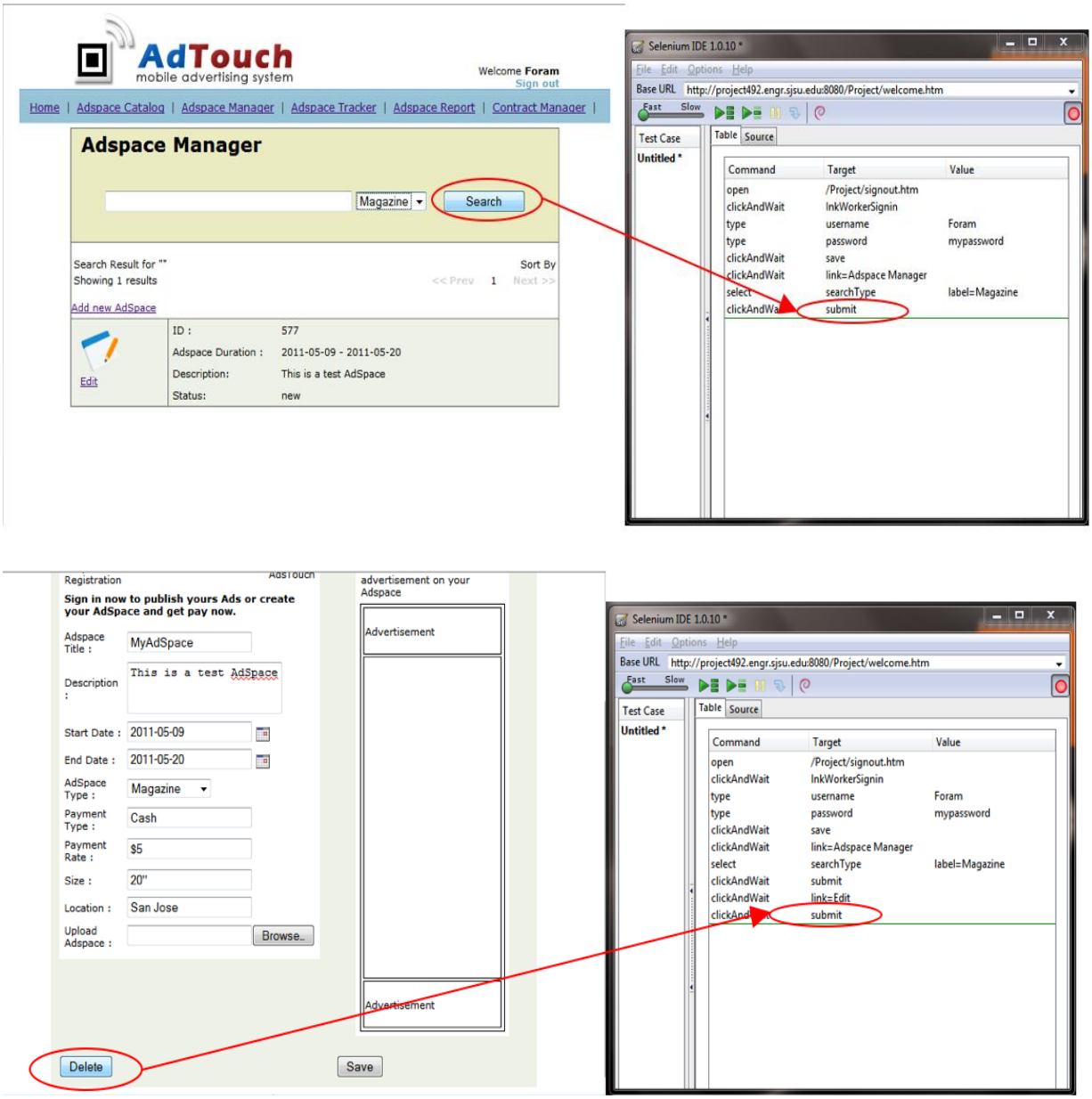

Figure 4. Final script actions of the same name.

This figure shows how two different scripts end with the same last script step. While there exists the ability to name the scripts per the functionality they are testing, inside the script the culminating result looks the same. Instead of “submit” being recorded for “search” and “delete,” the button names would be a better identifier for each button. As

the final script step, this discrepancy has the ability to confuse future work done in maintaining both scripts within the same test suite.

Unfortunately, if the above situations of inappropriate tags were to be reported as defects, they would not be considered as serious an issue. Due to the ability to access and work around the discrepancy, a lower priority than that of a small functional defect would be assigned. In building solutions that take advantage of the accessibility framework and its contribution to automation testing, a different process needs to be considered to reap the benefits of this combination.

6 Proposed Solution

In this paper, we propose to define a new, separate category of defects reported for accessibility problems of UI elements. The defects under this category include inappropriate UI element description tags, limited accessibility of programming components, and any other problem areas related to an accessibility implementation.

Making a separate category of these types of defects will encourage testers to find and report all possible discrepancies that they find during the course of their testing. And, additionally, developers will be able to differentiate a low priority “nice to have” defect from one that may be of a lower priority but has additional impact in both testing and end user ability to use the product with an AT. So while a workaround is possible in the short term, a fix is desired due to its influence in further downstream processes used by the testing department.

In order to put into effect this type of process change, consideration of adding this category in a compatible way to the current process must be done. Many enterprises employ the use of the Orthogonal Defect Classification (ODC) methodology for reporting bugs. Developed as a way to link cause and effect to the reason why a defect exists and the process that created it, an ODC category would be a possible way to incorporate this new defect category [21]. By introducing the cause-effect relationship of accessibility/automation test maintenance and the accessibility implementation process, this new category can be helpful in determining whether an enterprise needs to rethink their accessibility plan. This could include additional education for employees, refining the process, or determining the breakdown in communication related to accessibility.

This proposed solution also has a possibility to work well with the related work being researched in testing script maintenance described in a previous section. Both of the solutions described previously take into account the descriptive names of the UI elements used in the applications under test from version to version [1] [6]. The possible

increase in fixing defects reported against accessibility and its effect on automation testing can offer a greater chance in these object repositories containing even better information for determining better maintenance strategies of automation scripts.

Software testers and developers need to work together to develop any good software product. This applies to accessibility of software too. Both have to agree upon and use the best descriptive tags for each UI elements contained within the software. Testers should not just accept the values assigned to the tags if changing them makes the software better and easier for them to test it thoroughly. Fixing the tag name values is not a big change or issue for developers, so this small process change is something that can be considered to make the job of a tester easier.

7 Future Work

As recent research projects like [1] and [6] suggest, the future of automation tools is highly dependent on the accessibility layer implementation. It can only be assumed that this trend in automation tools will continue, so enterprises need to work into putting in the testing hooks accessibility can offer in order to leverage this trend.

In order to implement the proposed solution presented here, teams will need to look at their own defect reporting mechanisms and the categories and types they already include. In some instances, to add this necessary identification category will only involve adding the capability to handle another value. Once in place, testers will be able to use this new category for their reporting use and measurement in defect numbers for a given release before signing off on the finished product.

Moving forward, there exists a need to further educate enterprises and their developers about how accessibility can be a benefit that helps end users as well as those down the line in the software development cycle. The proposed solution presented here is one that has an impact on how defects and priorities are made in the development phase. When a team determines that accessibility will be included in the product, the activities necessary to include it need to become important as early on in the process as possible [17].

8 Conclusion

As computers continue to become increasingly prevalent and influential across the globe, the fusion of technologies among the development and use of products is inevitable. Although accessibility tagging in computer software began as a means of equalizing the access of information to people with disabilities, it has proven to be an

immensely promising development in the field of automation testing. Through its ability to save great amounts of time and, ultimately, money, automated testing has become a standard part of the enterprise software development process. Accessibility tagging has become a key component in the expanding usability of individual UI elements in increasingly complex applications on both the desktop, web browser, and mobile phone. In order to be effective, however, accessibility tag identifiers must be done in a logical, organized manner.

Through the addition of a new category for defect reporting and coordinated and cooperative development among testers and developers, testing tools will have the capacity to make great use of effectively implemented accessibility layers. These solutions have shown that there exists the potential in automation tools based off of the accessibility models that are already necessary to achieve the goal for opening up the solution to all possible customers. Whether on the desktop or a smartphone, the need to make a software product accessible has added benefits for developers, testers and end users alike. Thus, the return on investment for implementing an accessible solution can be deemed for many enterprises a worthwhile endeavor.

9 References

[1] Qing Xie; Grechanik, M.; Chen Fu; , "REST: A tool for reducing effort in script-based testing," *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on* , vol., no., pp.468-469, Sept. 28 2008-Oct. 4 2008
doi: 10.1109/ICSM.2008.4658108
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4658108&isnumber=4658028

[2] “Section 508 standards summary.” [Online]. Available: http://www.section508.gov/index.cfm?fuseAction=stdsSum

[3] Microsoft Corporation. (2010, Nov. 9). “Accessibility best practices.” [Online]. Available: http://msdn.microsoft.com/en-us/library/ff625908(v=VS.85).aspx

[4] T. Moss, (2007, Jan.). “The problem with automated accessibility tools.” [Online] Available: http://www.webcredible.co.uk/user-friendly-resources/web-accessibility/automated-tools.shtml

[5] Microsoft Corporation. (2010, Nov. 9). “Using UI automation for automated testing.” [Online]. Available: http://msdn.microsoft.com/en-us/library/ee684083(v=vs.85).aspx

[6] Conroy, K.M.; Grechanik, M.; Hellige, M.; Liongosari, E.S.; Qing Xie; , "Automatic Test Generation From GUI Applications For Testing Web Services," *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on* , vol., no., pp.345-354, 2-5 Oct. 2007
doi: 10.1109/ICSM.2007.4362647
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4362647&isnumber=4362597

[7] Wikipedia. (2011, Apr. 13). “Microsoft active accessibility.” [Online]. Available: http://en.wikipedia.org/wiki/Microsoft_Active_Accessibility

[8] Wikipedia. (2011, Feb. 2). “Microsoft UI automation.” [Online]. Available: http://en.wikipedia.org/wiki/Microsoft_UI_Automation

[9] T. Logan. “Developing for test automation and accessibility using programmatic access to the UI.” *Microsoft Professional Developer’s Conference, 2005*. [Online] Available: http://www.docstoc.com/docs/50630632/Developing-for-Test-Automation-and-Accessibility-Using

[10] Apple. (2011). “What’s new in iOS 4 - Apple developer.” [Online]. Available: http://developer.apple.com/technologies/ios/whats-new.html

[11] Selenium Project. (2011, May 5). “Selenium-IDE - Selenium documentation.” [Online]. Available: http://seleniumhq.org/docs/02_selenium_ide.html

[12] “Testdog - Test tool comparison.” [Online]. Available: http://testdog.com/community/test%20tool%20compare.html

[13] Mind Tree. (2008, Sept. 15). “10 regression/functional web testing tools.” [Online]. Available: http://www.hurricanesoftwares.com/10-regressionfunctional-web-testing-tools/

[14] G. Brajnik. “Engineering accessibility through corporate policies.” *Congresso Annuale AICA, 2005, Comunita Virtuale dalla Ricerca all’Impresa, dalla*

Formazione al Cittadino, Oct. 2005. [Online]. Available: http://sole.dimi.uniud.it/~giorgio.brajnik/publications.html

[15] G. Brajnik. “Using automatic tools in accessibility and usability assurance.” *Lecture Notes in Computer Science, Proc. of the 8th UI4ALL Workshop*. C. Stephanidis ed., Springer Verlag, Jun 2004. [Online]. Available: http://sole.dimi.uniud.it/~giorgio.brajnik/publications.html

[16] S. Trewin, et. al. “Accessibility challenges and tool features: an IBM Web developer perspective.” In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)* (W4A '10). ACM, New York, NY, USA, , Article 32 , 10 pages. DOI=10.1145/1805986.1806029 http://doi.acm.org/10.1145/1805986.1806029

[17] C. Shelly and M. Barta. “Application of traditional software testing methodologies to web accessibility.” In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)* (W4A '10). ACM, New York, NY, USA, , Article 11 , 4 pages. DOI=10.1145/1805986.1806002 http://doi.acm.org/10.1145/1805986.1806002

[18] TrevorH. (2010, Dec. 18) “XUL accessibility guidelines.” [Online]. Available: https://developer.mozilla.org/en/Accessible_XUL_Authoring_Guidelines

[19] Microsoft Corporation. “Accessibility support in ASP.NET.” [Online] Available: http://msdn.microsoft.com/en-us/library/ms228004(v=vs.80).aspx

[20] AdTouch, Inc. (2010). AdTouch mobile advertising system. [Online]. Available: http://project492.engr.sjsu.edu:8080/Project/welcome.htm

[21] R. Chillarege, et al., "Orthogonal defect classification-a concept for in-process measurements," *Software Engineering, IEEE Transactions on* , vol.18, no.11, pp.943-956, Nov 1992
doi: 10.1109/32.177364
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=177364&isnumber=4477