Semi-autonomous, site-wide A11Y testing using an intelligent agent

Briggs, K.A., Santiago, D., Adamo, D., Daye, P., King, T.M.

keith_briggs@ultimatesoftware.com

Abstract

Most countries across the globe have now ratified and/or signed the United Nations Convention on the Rights of Persons with Disabilities, the culmination of an effort started in the 1980's to protect the rights and dignity of people with disabilities. W3C's Web Content and Accessibility Guidelines documents, WCAG 2.0 (2008), ISO/IEC 40500 (2012), and WCAG 2.1 (2018), establish conformance requirements, which most nations have adopted as their de jure standard for determining website compliance with the UN convention. Although several tools support static WCAG web page analysis, present tools lack the automation required to automatically evaluate an entire website. In addition, current techniques are capable of discovering less than 30% of WCAG Level A and Level AA conformance issues. This paper presents an intelligent testing agent, Agent A11Y, capable of semi-autonomously exploring a website and more thoroughly evaluating its compliance with WCAG guidelines. To expand the system's automated WCAG guideline coverage, Agent A11Y employs novel dynamic testing strategies in combination with existing static web page analysis solutions. Additional tooling focuses manual testing activities on WCAG requirements that are difficult to fully automate. Supporting the generation of effective and actionable reports, Agent A11Y utilizes site exploration information to improve the aggregation and categorization of site-wide testing results. In conjunction with a discussion of the aforementioned Agent A11Y exploration, testing, and reporting methods, the paper presents experimental results and analysis to illustrate the comparative usefulness of the described techniques for conducting site-wide accessibility assessments.

Biography

Keith Briggs is a Software Architect at Ultimate Software, a leading cloud provider of human capital management solutions. With over 35 years of experience across diverse electrical, bio-electrical, software and systems engineering applications, Keith brings a unique perspective to his current AI for software testing research.

Dionny Santiago is a Quality Architect at Ultimate Software. Dionny is focused on research and development efforts to apply artificial intelligence and machine learning to software testing.

David Adamo Jr. holds a PhD in Computer Science from the University of North Texas and multiple years of experience conducting research and developing tools for automated software testing.

Philip Daye is a Software Test Lead for Ultimate Software. In his current role, Philip contributes to the development of patterns and practices with a special emphasis on accessibility.

Dr. Tariq M. King is the Senior Director and Engineering Fellow for Quality and Performance at Ultimate Software. Tariq heads Ultimate Software's quality program and is a frequent presenter at conferences and workshops. He has published more than thirty research articles in IEEE and ACM sponsored journals, and he has developed and taught software testing courses in both industry and academia.

1 Introduction

Approximately 1.125 billion people, or 15% of the worldwide population, experience significant difficulties in their everyday lives due to disability (World Bank 2018; World Health Organization 2011). Recognizing that people with disabilities face many barriers to full and effective participation in society, the United Nations adopted the Convention on the Rights of Persons with Disabilities, see Figure 1. Its primary aim is “to promote, protect and ensure the full and equal enjoyment of all human rights and fundamental freedoms by all persons with disabilities, and to promote respect for their inherent dignity.”

Focused on promoting universal access to web technologies, the World Wide Web Consortium (W3C) developed Web Content Accessibility Guidelines (WCAG). In 2008, WCAG 2.0 (2008) was formally adopted; in 2012, WCAG 2.0 became an international standard, ISO/IEC 40500, as approved by the International Organization for Standardization and the International Electrotechnical Commission; finally, in 2018, WCAG 2.1 added 17 new criteria to address issues with mobile accessibility, people with low vision, and people with cognitive and learning disabilities. Not only do website owners have a moral obligation to construct universal access websites and web applications, in most countries they have a legal obligation. In 2018, at least 2258 lawsuits alleging websites were not adequately accessible were filed in the United States, an increase of 177% over 2017. Starting on January 1, 2021, publicly accessible websites operating in Ontario, Canada must comply with WCAG 2.0 AA, except for WCAG success criteria 1.2.4 and 1.2.5. Failure to do so may result in fines of up to $100,000 per day for the organization and up to $50,000 per day for its officers and directors.

Despite the moral and legal imperative to adopt the WCAG standards, few web applications fully comply. A study, conducted in 2019 by Yan and Ramachandran, found that 94.8% of evaluated websites violated

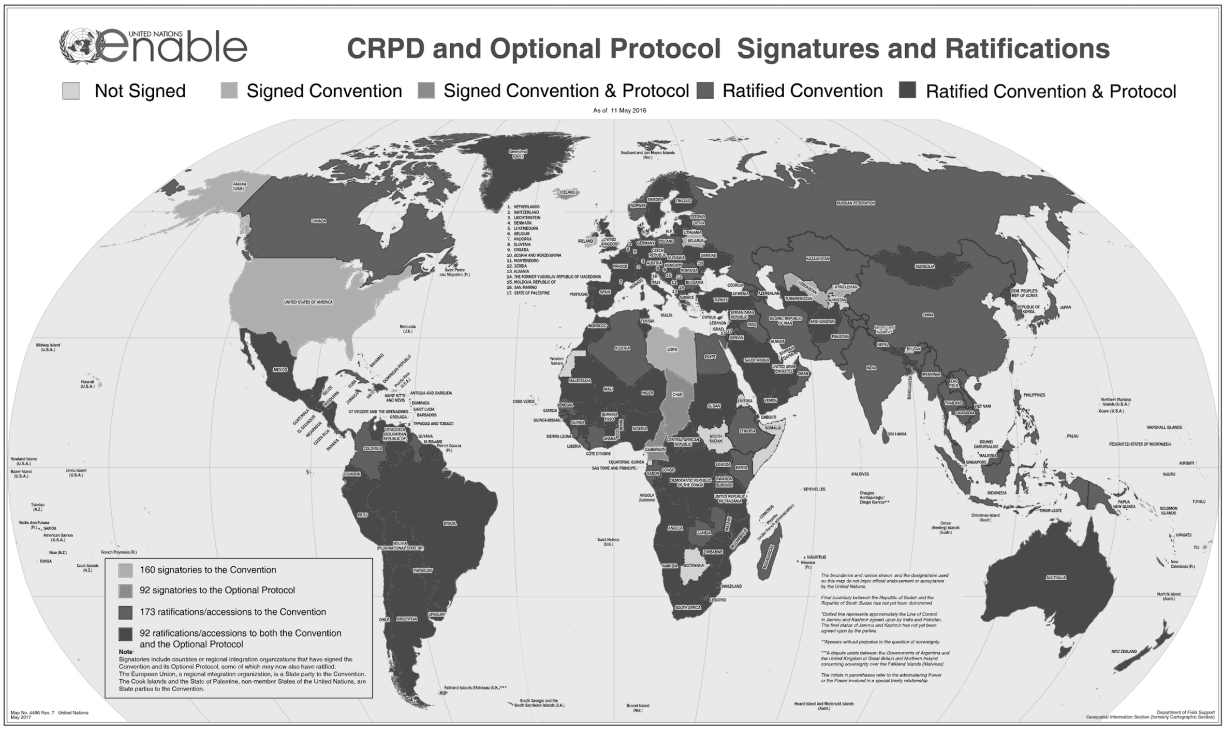

Figure 1: Most of the world has adopted, or is generally following, as in the case of the United States, the UN Convention on the Rights of Persons with Disabilities (CRPD).

WCAG 2.0 guidelines.

To help companies address accessibility compliance, several automated evaluation tools have been developed. In 2019, Vigo, Brown and Conway assessed six of the most popular tools in use today, and they found that 50% or less of the WCAG success criteria were evaluated. And, for the criteria which were assessed, the tools discovered less than 40% of the known problems. These tools determine if a web page complies with a set of best practices, or rules, derived from the WCAG guidelines. In general, the evaluations are conducted statically against the DOM in much the same way as LINT tools statically evaluate source code. Unfortunately, as is evident in the above study, the current static analysis tools cannot identify a majority of WCAG compliance failures.

Some existing tools can perform their analysis on a set of URLs, or they employ a crawler to develop additional web page targets from a single URL. However, the current techniques employed by the most popular accessibility conformance testers are not particularly effective at navigating websites with significant dynamic content, required forms, or difficult to reach pages.

This paper presents a new approach for dynamically evaluating WCAG conformance using an intelligent testing agent, Agent A11Y. The agent is capable of semi-autonomously exploring a website, developing a limited understanding of page context, and activating dynamic content to more completely evaluate compliance with the accessibility guidelines.

To expand the system's automated WCAG guideline coverage, Agent A11Y employs novel dynamic testing strategies in combination with existing static web page analysis solutions. Additional tooling focuses manual testing activities on WCAG requirements that are difficult to fully automate. Supporting the generation of effective and actionable reports, Agent A11Y utilizes site exploration information to improve the aggregation and categorization of site-wide testing results.

In conjunction with a discussion of the aforementioned Agent A11Y exploration, testing, and reporting methods, the paper presents experimental results and analysis to illustrate the comparative usefulness of the described techniques for conducting site-wide accessibility assessments.

2 Web Content Accessibility Guidelines

While the WCAG are imperfect, they represent a critical resource for companies as they attempt to ensure their websites and web applications are legally compliant with local laws such as the American with Disabilities Act (ADA) and the Accessibility for Ontarians with Disabilities Act (AODA), amongst others. Over the last decade, the WCAG have become either the de facto or de jure standard for assessing website compliance throughout much of the world.

WCAG 2.0 consists of four overarching principles, twelve guidelines, and 61 success criteria. WCAG's success criteria are testable statements which may be applied to websites independent of any specific technology. The following sections provide a brief overview of the WCAG principles and a few selected guidelines relevant to the paper. For a more complete understanding, please visit the Web Accessibility Initiative's (WAI) home page, https://www.w3.org/WAI/. This page includes links to a number of resources for content writers, designers, developers, and testers including the full WCAG 2.0 and 2.1 documents.

2.1 Perceivable

In order for a website to be usable, the information presented must be provided in a manner that can be perceived by the user. Several guidelines follow from this principle:

1. Text alternatives should be provided for non-text content,
2. Time-based media should provide captions or textual descriptions,
3. The content should be adaptable to different presentations without loss of information, and
4. Data should be distinguishable.

For data to be distinguishable, mechanisms must allow individuals with limited hearing or sight to adequately perceive the data. For example, one success criterion specifies that color should not be the

sole method of communicating information, as colorblind individuals may not be able to properly perceive differences in the data as presented.

2.2 Operable

It is not enough to specify that a user is able to perceive a website component, they must also be able to use its features. Several guidelines follow from this principle.

1. Some disabled users lack the motor control to be able to effectively utilize a pointing device; consequently, all functionality should be accessible using a keyboard alone.
2. Many disabled users require more time to complete standard functions; therefore, websites should provide enough time for users to read and use the content.
3. Flashing images or animations may cause difficulties for some users; thus, these features of a website, if employed, should be able to be turned off.
4. To operate a website, users must be able to navigate through the website. And,
5. WCAG 2.1 adds a fifth guideline addressing input modalities associated with mobile applications.

The fourth guideline concerning website navigation is divided into ten success criteria. The third criterion states, “If a Web page can be navigated sequentially and the navigation sequences affect meaning or operation, focusable components receive focus in an order that preserves meaning and operability.

On its surface, this success criterion is straightforward. If a user tabs through a website, the focus shouldn’t shift from the first item in a list to the last item in a list and then to a middle item. However, in practice, this criterion can be met through a wide range of web programming constructs. To simplify the problem, many accessibility validation tools define “best practices,” which build on WAI’s definition of sufficient techniques, see Figure 2.

Another sufficient technique specifies that the content order matches the visual order. If the DOM follows this approach, everyone may read and interact with the content in the same order. Assuming this technique is used, developers should not use positive HTML tabindex numbers, as the use of positive tabindex values may cause the focus order to follow something other than the order in the DOM. However, it is possible to independently implement a logical focus order using tabindex values greater than one to programmatically define the order in explicit terms. Current validation tools, such as aXe generate an error in such cases, even if the focus order is logical and satisfies the WCAG criteria.

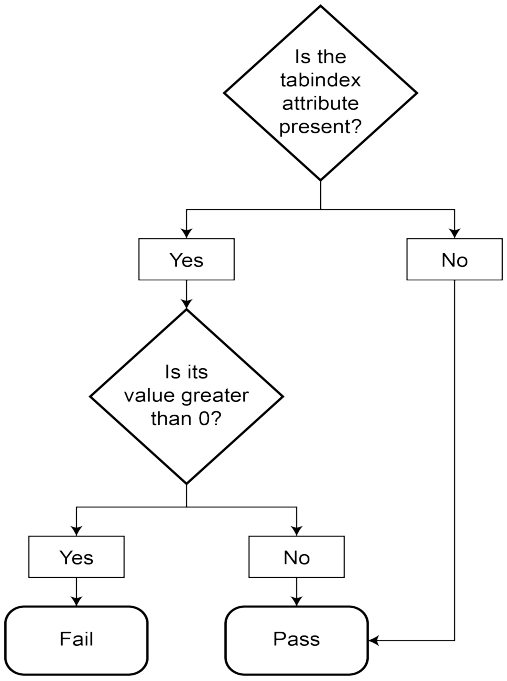

Figure 2: While some validation tools will indicate a webpage violates WCAG 2.4.3 if it uses a positive tabindex value, webpages can use positive tabindex values in a compliant manner. This should not infer that positive tabindex values are recommended. Rather, this example serves to expose the opinionated checking performed by compliance tools.

2.3 Understandable

The third principle states, “Information and the operation of user interface must be understandable.” The success criteria derived from this principle are difficult for current static accessibility tools to evaluate. They generally require active, rather than passive, assessments, i.e. clicking on a button, which current tooling does not support.

As shown in Figure 3, six of the most popular tools offered only minimal coverage of this principle. (Vigo 2019).

WCAG's Understandable principle is divided into three guidelines:

1. Content must be readable and understandable. This guideline primarily focuses on language definitions and word use (unusual words, abbreviations, etc.).
2. Web pages should appear and operate in consistent ways. That is, similar pages and components operate similarly across a website. And,
3. A website should help users avoid and correct mistakes.

The evaluated tools do not compare pages across a site; thus, many success criteria associated with the second guideline, Predictable, cannot be validated.

None of the tools studied support the testing of the third guideline, Input Assistance, which aims to ensure that users with disabilities have the tools to avoid and correct mistakes. The automated testing of this guideline requires that the testing system support: 1) the natural language processing of input recommendations and error correction suggestions, and 2) an ability to actively follow happy and sad testing paths based on a functional understanding of page inputs.

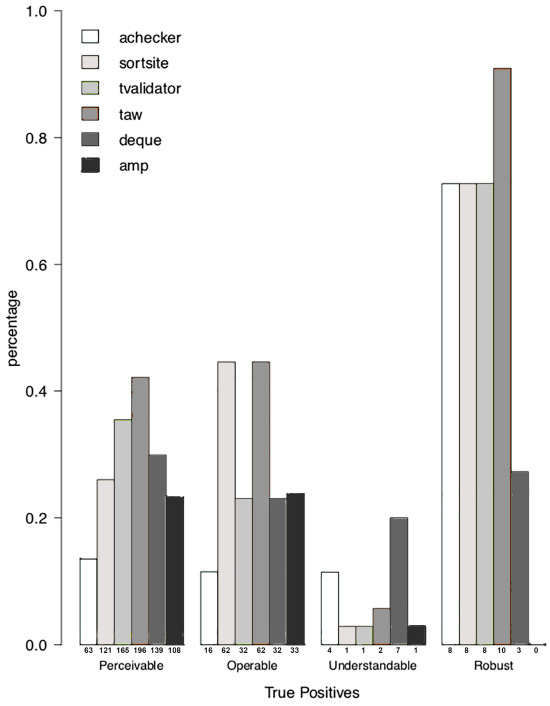

Figure 3: Completeness per tool and principle, where completeness is the number of true violations found by tools with respect to the actual number of violations reported by experts.

3 Current A11Y Testing Strategies

Comprehensively and efficiently evaluating the accessibility conformance of a website requires the use of multiple techniques. Effective strategies to reduce a11y issues within a website application should employ multiple testing techniques. Static assessments may detect issues such as an insufficient contrast level; whereas, dynamic, workflow testing is necessary to evaluate many other usability issues, such as a cognitive issues which may prevent a user from completing a task.

Ultimately, to keep costs down, companies should try to employ accessibility training and testing such that issues can be addressed as early in the design and development process as possible. Training content writers and developers on the specific issues associated with a set of persona profiles is helpful, as content authors and developers will be more apt to address accessibility design issues before they reach the testing process (Henke 2019).

Once in testing, verification tasks can be broken down into technical accessibility evaluations, and usable accessibility evaluations, see Table I (Bai 2017).

Because of the nature of A11y validation and testing, no one strategy can catch all possible accessibility defects. In 2017, Bai measured the percentage of defects discovered by various testing methodologies. Interestingly, the automated tools in use failed to discover any critical or cognitive failures, see Table 2.

TABLE I. ACCESSIBILITY TESTING GROUPS.

#	*Group*	*Technical accessibility*	*Usable accessibility*
1	Automated checkers	x	
2	Checklist and guidelines	x	x
3	Simulation kits		x
4	Assistive technology	x	
5	Walkthrough		x

While Table 2 indicates that manual techniques can be effective in discovering accessibility defects, it also illustrates the significant limitations of current automated tools. The automated tools studied by Bai only performed static analysis. The automated tools only caught 7.3% of all discovered issues. Further, they were not able to identify any of the errors that were classified as critical or cognitive. At the same time, even though manual techniques are capable of discovering such defects, manual accessibility testing is expensive. Further, as Bai has shown, multiple manual techniques are required to catch the errors. No single technique is sufficient.

Table 2: Percentage of errors discovered by method.

	Total	Critical	Cognitive
Automated tools	7,3 %	0,0 %	0,0 %
Checklists	24,6 %	29,4 %	15,5 %
Assistive Technology	20,3 %	31,0 %	14,1 %
Simulation kit	31,4 %	17,5 %	36,6 %
Expert walkthrough	16,5 %	22,2 %	33,8 %

Given the high lifecycle costs of manual testing, if companies are to reduce the costs of accessibility testing substantially, it will be necessary to improve upon available automated techniques.

3.1 Surveying Available Accessibility Testing Tools

Prior to embarking on the development of Agent A11y, Ultimate Software, a leading provider of Human Capital Management (HCM) software, conducted a broad survey of commercially available accessibility tools. The list of tools reviewed was derived from the Web Accessibility Evaluation Tools List published by the W3C (W3C 2016).

In general, five categories of tools were considered.

1. Content creation tools: These products aid users in the development of accessible content, primarily Word and PDF documents;
2. Color, contrast, and visual acuity evaluators: These tools aid a user in determining if a given combination of background and foreground content satisfies the WCAG guidelines for color, contrast and text readability;
3. Page scanners: These tools support the semi-automated evaluation of a web page using various static analysis techniques primarily focusing on DOM structure, proper page source formatting, and color, contrast, and visual acuity evaluations;
4. Accessibility testing libraries: These libraries allow the web-based functionality of plug-ins or stand-alone tools to be controlled via an Application Program Interface (API), thus allowing the tools to be integrated with other automated testing platforms; and,
5. Out-sourced evaluation services: Due to the contract nature of their work, the quality and capabilities of out-sourced evaluation services were not considered beyond what was available from their published marketing materials.

Ultimate Software's assessment primarily focused on the page scanners and associated accessibility testing code libraries. Web based page scanners often encounter difficulties when working with sites that require a login, and they may have limited or no ability to navigate a complex site with forms, required fields, etc. Since many of the scanners run in the vendor's own datacenter, there is the additional risk of exposing confidential data during the testing process. However, some of these tools are built on code libraries, which can be executed locally and via programmatic control. While the accessibility testing libraries can be used to scan pages, they do not generally support site exploration, form entry, or scripting. As a consequence, they must be used in conjunction with other GUI test automation tools.

As part of the study, Ultimate Software evaluated numerous tools. The conclusions drawn from the assessment were in general agreement with those reported by Bai, Henke, and Vigo as reported in the sections above. The study identified several approaches for conducting web page analysis, assuming the desired System Under Test (SUT) page was discoverable. These solutions were capable of identifying a wide range of static formatting and page display issues to include missing alternative text, insufficient contrast, improperly structured data, and missing guides such as skipped heading levels or landmarks.

It was found that all of the tools evaluated lacked seven key capabilities, which were critical to determining the accessibility compliance of enterprise scale, web applications in a cost-effective and repeatable manner. The tools did not support:

1) An automated, or semi-automated (i.e. scripted), ability to navigate to all application states, or at least a representative set of application states, within a web-based, enterprise application;
2) A capability to assess site-wide accessibility criteria, such as WCAG success criteria 3.2.3, Consistent Navigation;
3) The ability to dynamically interact with a webpage to actively assess compliance with certain WCAG guidelines, such as Guideline 2.1, Keyboard [Accessibility];
4) Company specific accessibility design verification, which may look to the enforcement of site-specific guidelines for a website or company;
5) Natural language or machine learning capabilities capable of evaluating page layout, form labeling, help content, and error text;
6) An ability to effectively deduplicate and present aggregated, actionable results across multiple states of an application feature or page; and,
7) A workflow assessment capability capable of determining the accessibility of a series of pages and actions necessary to complete a task within an application.

Agent A11y targeted items 1, 3, and 6 for further evaluation. The approach and findings are presented in the following sections.

3.2 Automated Testing

Static page analyzers, such as Deque's aXe and Level Access' Continuum (AMP), are capable of discovering a number of important WCAG violations, but they cannot detect critical or cognitive accessibility defects, see Table 2. As reported by Bai (2017), these tools failed to detect issues associated with more than 50% of the WCAG success criteria, see Figure 4. Further, even where success criteria defects are detected, the tools cannot determine full compliance.

Fundamentally, these tools are limited in what they can evaluate, as they presently only perform static evaluations of a webpage's DOM and limited rendering tests. As most enterprise applications utilize dynamic web pages to support complex workflows, statically focused, single page, static evaluation systems cannot evaluate most critical application features. The researched, commercially available validation tools share these common limitations:

1. The evaluated systems are not able to evaluate multi-state or multi-page defects. As these systems operate on pages independently, even if they are utilized in conjunction with a web crawler, they cannot effectively assess cross-page conformance requirements. For example, WCAG success criterion 3.2.3 specifies that a website use consistent navigation such that, "Navigational mechanisms that are repeated on multiple Web pages within a set of Web pages occur in the same relative order each time they are repeated, unless a change is initiated by the user."

2. The evaluated systems cannot assess dynamically changing content. As the systems are statically employed, and as they determine compliance through an analysis of a single DOM instance, they are unable to evaluate the results of a button push, form submission, or other action. This limits the ability of the systems to detect violations of success criteria such as

Tool	P	O	U	R	overall
AChecker	3 (38%)	3 (25%)	1 (25%)	1 (50%)	8 (31%)
SortSite	3 (38%)	5 (42%)	1 (25%)	1 (50%)	10 (38%)
TV	3 (38%)	3 (25%)	1 (25%)	1 (0.5)	9 (35%)
TAW	3 (38%)	5 (42%)	2 (50%)	2 (100%)	13 (50%)
Deque	3 (38%)	3 (25%)	4 (100%)	1 (50%)	11 (42%)
AMP	2 (25%)	3 (25%)	1 (25%)	0 (0%)	6 (23%)

Figure 4: The evaluated tools were unable to detect any failures associated with 50% of the WCAG success criteria.

WCAG 3.2.2, On Input. This criterion specifies, “Changing the setting of any user interface component does not automatically cause a change of context unless the user has been advised of the behavior before using the component.”

3. The evaluated systems do not attempt to understand the functional operation of the website in performing their evaluation. As such, they are not able to develop the models and contextual information needed to evaluate whether or not an error described to the user is adequate as required by WCAG 3.3.1.

Further, when evaluating multi-page websites and websites with dynamic content, it is critical that automated testing solutions provide actionable, consolidated information to guide developers in correcting the failure. While many of the tools mentioned provide impressive reporting mechanisms and guidance when applied to individual pages, none offer the ability to consolidate accessibility errors generated across different states of the same page, and neither can they evaluate information that may be presented across multiple similar pages, such as error messages. Lacking these capabilities, if current automated solutions are directly applied to multiple instances of the same page in different states, or similar pages with duplicative issues, the user may be inundated with massive quantities of largely duplicative data. This can make finding and acting on web application design issues particularly difficult.

3.3 Evaluating Design Quality Versus Simple Conformance

None of the solutions evaluated were capable of evaluating the sufficiency, rather than the existence, of a particular solution. Consider the case of a button label; the evaluated tools only test that a label is provided. They cannot determine if the information can effectively support the operation of the web application by a person who uses a screen reader or other assistive device. For example, a webpage may contain several buttons whose context within the page establishes its functionality. While a shopping page might have several buttons labeled “add” on a page, in many cases, each instance of the button may add a different item to a user’s shopping cart. For a sighted user, the function of each “add” button may be easily determined from the location of the button in relation to the surrounding context. However, unless additional information is provided in the accessible version of the label provided to a screen reader, a person who is blind may be unable to determine which item will be placed in their cart when any of the multiple “add” buttons are selected. Consequently, merely testing for the existence of a label is often insufficient to determine if a given label provides adequate accessibility.

While neither the evaluated tools nor Agent A11y can currently address this deficiency using automated testing techniques, Agent A11y’s approach allows for the introduction of dynamic tests, techniques and oracles which could be capable of addressing this issue in the future. Agent A11y could click on all commonly labeled buttons (using the accessible label) on a page and flag an error if the resulting action is not identical. Such a solution would build on the existing technology and capabilities of the Agent framework already incorporated in functional testing versions of the system and its predecessors. Not only is the framework capable of clicking on buttons and modeling the results, Agent and its predecessors can extract semantic information from labels, error messages, links and workflows. However, applying these techniques to support evaluating the effectiveness of accessibility designs and labels remains a work-in-progress. While a machine learning enabled solution which completely eliminates the need for manual verification of such content may be decades away, considerable opportunities to reduce the amount of manual verification required may be attainable within much shorter timescales, see Section 5, Future Work.

4 Agent A11y

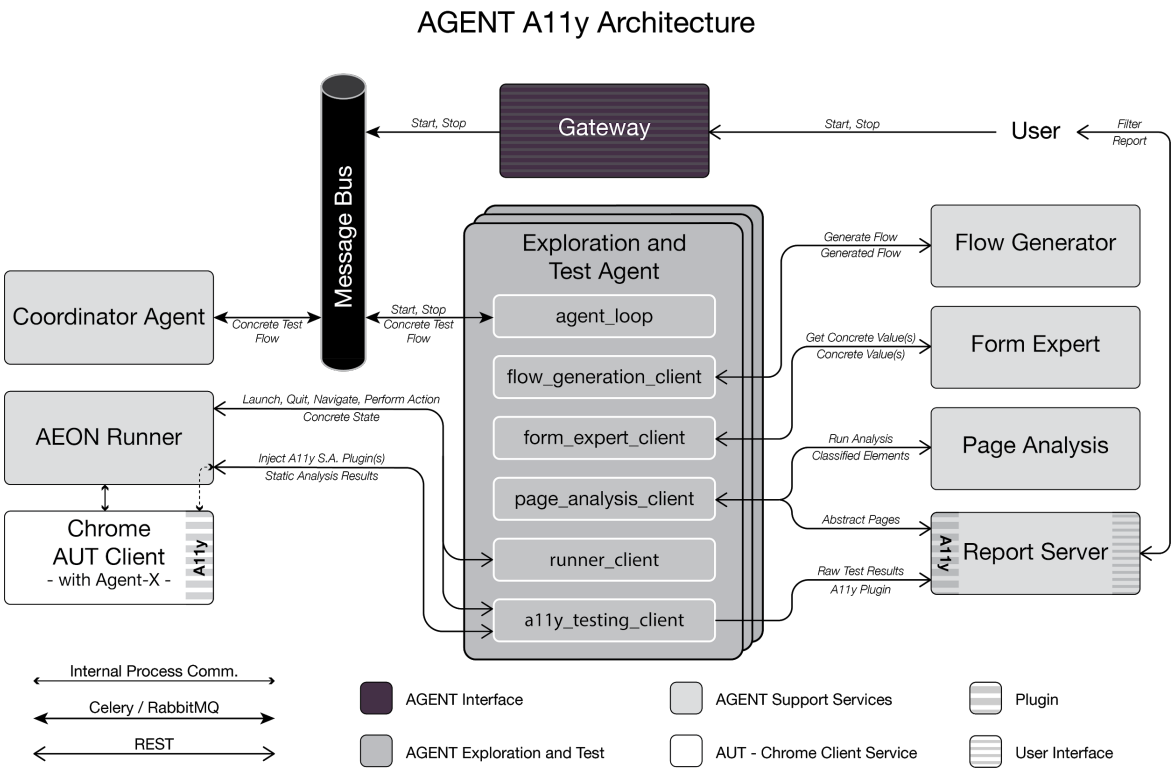

Figure 5: Agent A11y automatically explores a web application, conducts both static and dynamic accessibility testing, and reports consolidated test results.

Based on Ultimate Software's survey of available accessibility testing tools, a review of current research, and an evaluation of ongoing internal automated testing research projects, the team began efforts to integrate available commercial technologies with associated AI driven testing research. The project sought to combine both static and dynamic accessibility testing solutions with an agent based, autonomous testing system. The resulting research, Ultimate Software's Agent A11y project, is the focus of this paper. Other efforts have pursued the integration of the non-agent-based testing features into the company's Aeon open source project. Aeon is available under an Apache 2.0 license on GitHub.

To enable context sensitive and site-wide accessibility testing features, it is necessary to evaluate the capabilities of the SUT over multiple states, preferably a collection which abstractly represents the full functionality of the product. Without the benefit of the contextual information gathered across multiple product pages and states, an autonomous testing solution cannot reasonably determine if adequate accessible information is available in a form suitable for people with a variety of disabilities. With this in mind, the Agent A11y system, based on the open sourced AGENT (AGENT 2019), is constructed around an Exploration and Test Agent capable of exploring a web application without significant direction or human intervention. Agent A11y is capable of identifying, navigating, evaluating, and acting on the actions, fields, and forms constituting an application's pages and workflows. Agent A11y deploys one or more exploration and testing agents to navigate a system, and it applies appropriate test flows as testable patterns are recognized.

A coordinator dispatches testing assignments through a message queue. The queued tasks are distributed to multiple distributed, concurrent execution and testing agents. Each agent is capable of independently interacting with the SUT via an instrumented Chrome client and AEON runner. This AI for Software Testing solution, designed for research and development applications, illuminates a new path for advancing the state-of-the-art in testing automation.

As the agents explore the site, they conduct static accessibility evaluations for every, sufficiently new, discovered state. In addition, the agents execute supported dynamic accessibility tests as relevant testable patterns are recognized.

4.1 Automatic Exploration

The automatic exploration process begins with a *start request* from an end-user of Agent A11y. The *start request* includes information about how many exploration agents to instantiate, a completion criterion (e.g. a specific number of exploration actions per agent or a fixed amount of time) and a seed URL from which the exploration process should begin. Each agent's goal is to visit as many distinct pages of the SUT as possible.

Traditional web crawlers use a breadth-first or depth-first strategy to collect and visit links (typically identified by HTML anchor tags). Agent A11y interacts directly with buttons, text fields, checkboxes, scrollbars and other web page elements in ways similar to actual users. Automated exploration in Agent A11y is a precursor to automated accessibility evaluation of each web page / state that constitutes a SUT.

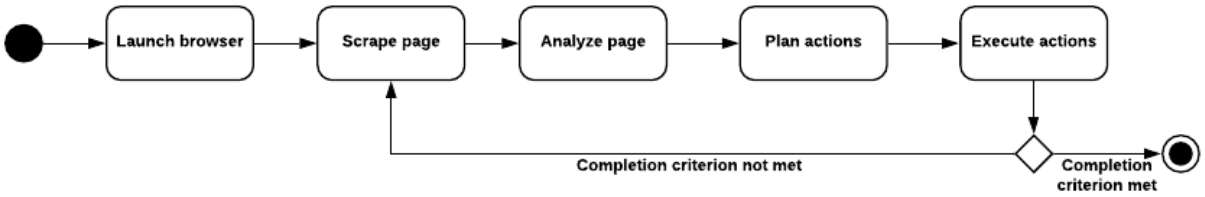

Figure 6: The automatic exploration process with a single browser.

The *start request* from an end user initiates an exploration session, which can be uniquely identified and later retrieved with all associated data. As shown in Figure 6, automatic exploration involves a sequence of steps (browser launch, page scraping, page analysis, action planning and execution) that occur in a loop until a specified completion criterion is met.

Browser launch: At the start of the exploration process, Agent A11y spawns one or more exploration agents, launches web browsers, and navigates to a specified seed URL.

Page Scraping: Agent A11y extracts a Computed Render Tree (CRT) representation and stores a screenshot of each web page it visits. The CRT is generated by injecting JavaScript into each web page during exploration. The CRT contains data about the structure, content and visual characteristics of each web page. This is in contrast to HTML source which focuses on structure and content but has limited information about visual characteristics of a web page. Web page screenshots are useful for further analysis and reporting at different stages of the exploration process.

Page Analysis: Agent A11y uses information derived from CRTs to generate unique identifiers for similar pages, to classify web page elements (e.g. page titles, labels, commit buttons, error messages, etc.), to identify groups of elements (e.g. forms, navigation bars, label-form field mappings), and to identify actionable elements and construct selectors for actionable elements.

Action Planning and Execution: The information derived from page analysis serves as input for action planning. During exploration, Agent A11y uses page analysis information to determine a sequence of actions that have a high likelihood of leading to previously unseen web pages. Once such a sequence of actions has been chosen, Agent A11y uses Selenium and JavaScript injection to execute the chosen sequence of actions. Executing one or more actions typically results in navigation to a different web page. It is important to note that this stage also includes actions used to assess the accessibility of each web page (e.g. execution of an accessibility analysis tool).

The exploration agent loop tracks exploration state and determines whether the completion criterion has been met. Agent A11y, as implemented, is a highly scalable, distributed system capable of running multiple exploration and accessibility evaluation processes in parallel across multiple agent/browser instances. The goal of automatic exploration is to visit as many distinct pages as possible in as little time as possible. The comprehensiveness of accessibility evaluations across an application, in large part, depends on how deeply Agent A11y is able to explore the SUT.

4.2 Form Recognition

During the exploration process, it is important for the system to be able to recognize forms and fill out valid values for form elements. This is especially true for enterprise applications that contain many forms with complex form validation requirements. The inability to complete a form may prevent the Agent from reaching subsequent portions of the SUT. Once the system learns to successfully complete a form, Agent A11y may recognize applicable test flows, which may then be executed to validate the accessibility of the form. A comprehensive accessibility evaluation requires testing the form under both positive (i.e. with valid values for all form fields) and negative (i.e. at least one field with an invalid or missing value) conditions. Negative tests are required to determine the SUT's compliance with WCAG Guideline 3.3, Input Assistance. Agent A11y is capable of learning and applying both positive and negative tests on recognized forms.

To understand a form, it is necessary to first extract the actionable widgets that comprise the form. The *page analysis client* employs ML techniques to recognize different types of form elements (e.g., labels, error messages, form and page titles). The problem of understanding web page components is framed as a supervised learning classification problem (Santiago 2018). A standard web-based render tree includes information such as: (1) hierarchy; (2) element types; (3) element attributes; and (4) style sheet information. We extend this basic structure by collecting the render tree only once the web browser has finalized rendering, and by calculating additional information such as: (1) element positions; and (2) element sizes. The result is a Computed Render Tree (CRT) representation of the webpage.

Using the CRT representation, a feature synthesis step is then performed. An element-wise pass is done through all of the elements in the CRT, synthesizing several features for each element. Although the synthesis is done at the local level for each element, the full global context (information on the entire set of elements) is used to compute several features. The feature synthesis results in the generation of training data that can be annotated for the purpose of training ML models. A set of example synthesized features is described in Table 3.

Table 3: An example set of features that can be collected or synthesized from the CRT.

Feature	**Description**
HTML Tag	The tag for the given element.
Parent HTML Tag	The tag for the given element’s parent.
“For” Attribute	The existence of a value for the HTML "For" attribute.
Num. Children	The number of HTML nodes that are children of the given element's node.
Num. Siblings	The number of HTML nodes that are siblings of the given element's node.
Depth	The depth of the given element's node within the CRT.
Horizontal Percent	The relative horizontal position (in percentage) of the given element.

Vertical Percent	The relative vertical position (in percentage) of the given element.
Font Size	The relative (normalized against the full set of elements) font size of the given element.
Font Weight	The relative (normalized against the full set of elements) font weight of the given element.
Is Text	Describes whether a given element is a text node.
Nearest Color	The closest color computed using CIEDE2000 algorithm.
Nearest Background Color	The closest background color computed using CIEDE2000 algorithm.
Distance from Input	The relative (normalized against the full set of elements) distance to the closest input widget from the given element.
Text	The actual text associated with the given element.

Using the synthesized feature values, the Agent A11y ML-based classification system is capable of recognizing key web elements (Page Title, Widget Labels, and Error Messages) critical to navigation within the SUT and the recognition of form elements, see Figure 7. As mentioned, the ability to properly fill-in forms is necessary to the navigation of the SUT, as invalid values can block further progress into the SUT. Form recognition is an important first step in the process of completing a form.

As the system must first navigate to a SUT state to evaluate the accessibility of the page in a given state, Agent A11y must be capable of successfully completing encountered forms. Additionally, the recognition of form components and their associated error messages is required to evaluate a web application's compliance with WCAG's Input Assistance Guideline (WCAG 3.3). To comply with this guideline, a web application must provide suitable labels, instructions and help so that a user who is disabled may properly understand how to fill-out the form. Further, when an error is generated, WCAG 3.3 requires that the error suggestions correctly identify the user error so that it can be corrected. While the parsing and evaluation of error messages for the purpose of evaluating WCAG 3.3 compliance has not yet been implemented in Agent A11y, related work associated with understanding form semantics and interpreting error messages provides encouragement that future implementations may be capable of validating the accessibility of common error messages.

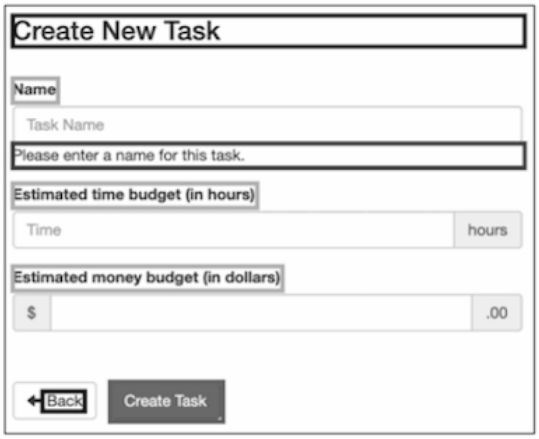

Figure 7: The Agent A11y ML-based classification system is capable of automatically identifying page title, widget label associations, and error messages (not shown) on an untrained web page.

In addition to identifying a page title or form label, the ML classifiers also construct bounding boxes for each of these elements. While the system is currently able to recognize individual web page elements; oftentimes, it is useful to recognize a group of elements as a single semantic entity, see Section 5, Future Work.

4.3 Understanding Form Semantics, Form Filling

The information collected from the previous step (Form Recognition) enables additional reasoning concerning the functional and semantic context of the form under exploration. To help define the domain of valid input values associated with an element, Agent A11y attempts to first determine which data types may be associated with the element, and subsequently, any relationships that may exist between associated form elements. This process involves creating an abstraction of the page under exploration.

Abstraction

A full CRT contains information about every DOM element for a given web page. To effectively support further reasoning about key elements, Agent A11y creates a simplified representation, or abstraction, of the page. This abstraction mechanism also supports Agent A11y's SUT navigation task by reducing the number of duplicative paths which must be explored, and by improving the effectiveness of Agent A11y's reinforcement learning algorithms. Theoretically, multiple approaches to state abstraction are not only possible, but appropriate, as the state abstraction methodology directly impacts the mechanics of exploration and the type of test flows which the system can detect. For this implementation, the Exploration and Testing Agent constructs a hierarchical *semantic model* leveraging the information collected during the form recognition stage.

Each node within the semantic model contains information that identifies a specific form control. Nodes are hierarchically organized based on element positioning relative to form headers and page titles. Generally, web forms are composed of inputs and labels. As such, we employ a label extraction process that associates each control with the nearest label that may be used to describe the control (Becce 2012, Lin 2017). The labels may range from singular words to phrases or sentences. Continuing with the form from Figure 7, the semantic model identifies the key components and one or more suggested data types for completing the form, see Figure 8. While not shown in this example, Agent A11y is also capable of identifying other important form characteristics such as required fields, hints, and error messages.

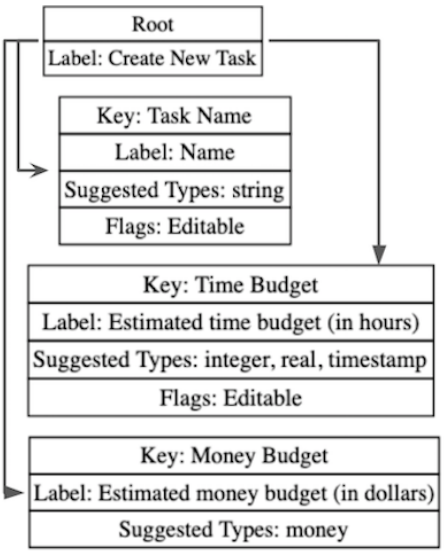

Figure 8: The Agent A11y ML-based classification system supports the generation of a semantic model representation, which identifies the key form components and likely input types.

Actionable State

Agent A11y constructs a class representing a page state of the SUT as a collection of widgets, the *actionable state*, from the semantic model. A hashing function that only depends on the actionable widgets for a given page is applied to the semantic model. The output of the hashing function is a hash that is used to determine uniqueness among all explored abstract states. Concretely, any variant of the form explored in Section 4.2 (e.g., one of the fields is filled out, or the field is empty, or multiple fields are filled out) will be represented by the same abstract state (that is, all form variants are represented by the "same" semantic model). As mentioned above, multiple abstraction techniques may be applied to the CRT, and its associated semantic models. This particular abstraction technique, which focuses on the actionable components of the CRT, has proven useful for identifying states important to the completion of forms and the successful navigation of a SUT.

Data Type Detection

Agent A11y's Form Expert Client consumes discovered semantic models and determines potential valid values for completing the form. The form expert includes a natural language processing (NLP) engine built on top of the Stanford CoreNLP Toolkit (Manning 2014). The NLP engine recursively processes each node of the model. For each node and parameter therein, the engine has one primary goal: to suggest likely types of each parameter. The NLP engine processes the text label of each parameter using an internal knowledge base which maps likely types to names. The output is an array of suggested types from most to least probable. For example, a text label "First Name" containing the substring "Name" is most likely a *string* data type. Conversely, a label such as "Starting Date" containing "Date" may refer to either a *date* control, a *timestamp* control, or a *string*, in order of likelihood.

If an exact match is not found in the knowledge base, a text similarity search is conducted with the label against the members of the knowledge base to see if it is similar in concept to any member of the knowledge base. If so, the search result is used as a proxy, with an adjusted confidence based on how strong the similarity score is. As part of the NLP process, the semantic model is augmented with the new, derived information representing the likely data types.

Form Filling

The augmented semantic model is then passed into the form filling subsystem of the form expert client. The form filling subsystem faces two distinct challenges: (1) providing correct values given labels on a form; and (2) contextually generating values for a given form. Typically, form controls are provided with labels, and natural language descriptions of the value types. Some of these controls maintain contextual relationships. For instance, a form's structure may change when selecting a country that may or may not have states or provinces. Many forms, such as registration or payment forms, contain similar controls, differing by only a small number of controls. Given the augmented semantic model, the form filling subsystem leverages ML (specifically, the K-Nearest Neighbor algorithm) to find the semantic model within a training dataset that is closest to the provided model. Levenshtein distance is used to measure label distances across label sequences (Levenshtein 1966). To support form filling given a limited training set of data, an analogy-based reasoning approach is employed (Aamodt 1994, Veloso 1992).

The form filling subsystem may not find a semantic model that matches one-to-one for all form fields. It is entirely possible that the training data includes enough information to fill out a subset of the target form under exploration, but not enough data to fill out the entire form. To compensate, the form filling algorithm employs a recursive subset filling approach. That is, the algorithm explores a search space of all possible form field subset combinations until it converges to a combination for which as much of the form is filled as possible.

4.4 Static Tool Execution

Given the ability to automatically explore the SUT, the AGENT system serves as a platform that supports the integration of additional testing tools. For accessibility testing purposes, aXe Core and Continuum static analysis tools are used. Both of these tools provide a JavaScript library for simple injection and

execution against any web page. Since the AGENT web execution system is based on Selenium, the system is able to leverage the existing JavaScript execution capabilities available in the Selenium toolchain.

In order to seamlessly leverage multiple tools such as aXe Core and Continuum, it is necessary to create a common A11y domain model to capture accessibility issues. The system maps the specialized aXe Core and Continuum results into this common A11y domain model. This allows the system to generate an aggregated report that leverages the results of multiple A11y static analysis tools to improve rule coverage while limiting report complexity.

As mentioned in Section 4.3, the AGENT platform is able to create abstractions for all of the pages found during exploration. ML classifiers are used to recognize the page title for each page. Common pages are then clustered together. This approach allows the system to aggregate A11Y issues across all known variants of a given page abstraction, see Figure 9. If the system did not aggregate results based on an abstract understanding of a SUT, and rather reported the accessibility test results associated with each concrete page instance discovered, the report would be tens or hundreds of times longer, as each abstract page is visited many times in a typical session.

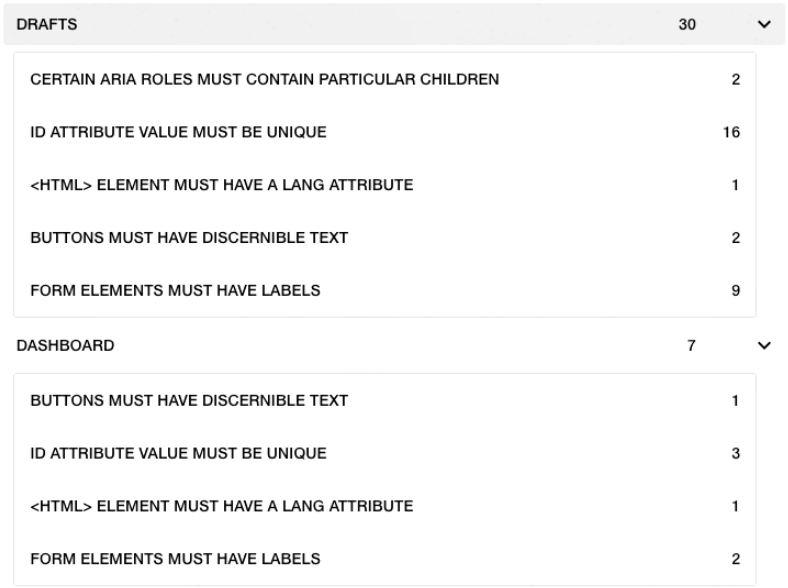

Figure 9: This screenshot from the Agent A11Y accessibility reporting page illustrates the identification of multiple a11y issues as discovered on multiple SUT abstract pages.

4.5 Dynamic Accessibility Testing

As described in Section 4.4, Agent A11Y is able to utilize a range of static analysis tools including the popular aXe Core and Level Access Continuum plug-ins. Further, the system is capable of aggregating the results into a single report, which organizes discovered issues according to the abstract SUT page on which they were discovered. The system aggregates duplicative information discovered on multiple concrete pages, or by multiple testing tools to reduce information overload. While such a capability is powerful and useful in it of itself, static analysis tools are currently only able to cover a small percentage of all a11y compliance concerns. As such, there is a need for dynamic a11y testing. Our system extends the static a11y testing tooling by introducing dynamic tests which are executed on every abstract page encountered during exploration.

An example of a dynamic a11y test involves the testing web element focus order. While it is difficult to build an automated oracle for what the correct focus order should be on any arbitrary web page, it is

possible to develop oracles for simpler problems such as ensuring there are no keyboard traps on the page. A keyboard trap occurs when a person using a keyboard is unable to shift focus away from a certain element or control. Agent A11Y is able to test for keyboard traps by employing the following algorithm:

1. Scrape the contents of the page under test
2. Identify the actionable elements on the page
3. Based on the number of actionable elements, determine an upper bound for the number of times the Tab key should be hit
4. Identify the currently focused element and add it to a running list of focused elements. At the end of the algorithm, this running list will represent the element focus order on the page
5. Hit the tab key
6. Scrape the contents of the page under test
7. Jump to Step 4 as long as we have not exceeded the upper bound on the number of times to hit the Tab key. Else, continue to Step 8
8. Check the element focus order list for cyclicity, and for the presence of a keyboard trap (duplicate controls, especially when contiguous)

In addition to testing for keyboard traps, Agent A11y supports other oracles capable of verifying if all actionable controls can be reached (i.e. focused on) by tabbing. The system employs a similar algorithm that identifies the “actionable” elements which are reachable from a given concrete state and then the system checks to ensure each element is accounted for within the focus order list.

There are many promising avenues for future work to expand the range of dynamically testable, accessibility compliance concerns, see Section 5, Future Work.

4.6 Web Page Clustering and Accessibility Reports

Agent A11y generates and stores large amounts of data as a result of its automated exploration and accessibility evaluation processes. Reporting this data to end users requires techniques that enable derivation and presentation of useful information that end users can understand and act upon.

Agent A11y organizes and presents accessibility reports in a page-centric manner. The goal is to help users identify, in as specific a manner as possible, web pages in the SUT that have accessibility issues. During automatic exploration, Agent A11y may visit many slightly different instances of the same page, and it is potentially undesirable to present the results of accessibility evaluation for every instance of each visited page as part of an application-wide report. A key objective of Agent A11y’s reporting scheme is to organize accessibility evaluation data into representative groups based on characteristics shared between multiple instances of each web page. Other key aspects of Agent A11y reports include web page screenshots and descriptions of identified accessibility issues.

Agent A11y utilizes an unsupervised machine learning algorithm, *k*-medoids (Shubert), to identify clusters (i.e. representative groups) of visited web pages and associated accessibility evaluation data. Each member of a cluster has more characteristics, or features, in common with other members of the same cluster than with members of other clusters. The centroid of each cluster is the most representative instance of a single web page that potentially has several instances. Rather than present accessibility information about all members of each cluster, Agent A11y focuses on the centroid of each cluster to provide tractable reports.

Clustering algorithms generally require a vectorized representation of each entity that is to be analyzed. In the context of accessibility evaluation, the entities are web pages. Table 4 shows the list of features that are used to represent each web page as part of the clustering process.

Table 4: Web page features for clustering.

Feature	Description
Tag frequency	The number of times each HTML tag appears in a web page.
Word frequency	The number of times each word appears in a web page.
URL	Levenshtein distance between a web page's URL and the URL of all other explored web pages.
Document Object Model (DOM)	Tree edit distance [1] between the DOM tree of a web page and the DOM tree of all other explored web pages.

One of the inputs to the *k*-medoids clustering algorithm is the number of clusters into which data points should be grouped. In the context of accessibility reporting for web pages, it is unlikely that the appropriate number of clusters is known prior to accessibility evaluation for a given application. Algorithm 1 illustrates how Agent A11y determines an appropriate number of clusters to use for reporting.

```
function identifyRepresentativeWebPages(vectorizedWebPages, maxClusters)
    bestNumberOfClusters = 2
    bestSilhouetteScore = -1
    bestClustering = vectorizedWebPages
    currentNumberOfClusters = 2
    while currentNumberOfClusters <= maxClusters:
        clusters = kMedoids(vectorizedWebPages, currentNumberOfClusters)
        silhouetteScore = calculateSilhouetteScore(clusters)
        if silhouetteScore > bestSilhouetteScore:
            bestSilhouetteScore = silhouetteScore
            bestNumberOfClusters = currentNumberOfClusters
            bestClustering = clusters
        currentNumberOfClusters = currentNumberOfClusters + 1

    representativeWebPages = getCentroids(bestClustering)
    return representativeWebPages
```

Algorithm 1: Web page clustering with silhouette coefficients

Algorithm 1 takes the set of featurized web pages and a user-defined maximum number of clusters as input. The algorithm produces a subset of representative web pages from the superset of all explored web pages. For a given SUT, Agent A11y repeatedly clusters the set of visited and evaluated web pages until it identifies a number of clusters within a specified bound (e.g. 2 to 50) that yield the best silhouette coefficient for clustering (Selecting 2019). The silhouette coefficient is a measure of how close each entity in a cluster is to points in neighboring clusters. The silhouette coefficient provides a way to assess the quality of a set of web page clusters. Once an appropriate number of clusters has been identified, Agent A11y uses the resulting subset of representative web pages to produce a report that is assumed to be more tractable than one based on the superset of all visited pages.

5 Future Work

While Agent A11y has demonstrated the capability to autonomously evaluate a broad set of WCAG success criteria, the system is still in its infancy and many opportunities to expand upon its supported test strategies remain.

5.1 Evaluating Information Content Versus Existence

In most cases, current solutions, including those currently supported by Agent A11y, evaluate the existence of a label and not the sufficiency of its content. While the construction of a system capable of interpreting the meaning of all labels, help messages, captioning content, and errors is well beyond the present state-of-the-art, cross-site comparison algorithms, NLP techniques, and machine learning classification systems may be applied to assess the sufficiency of certain information content. Consider these examples:

- A site uses many common form elements across a site, such as *name*, *address* and *phone number*. Using several examples, a testing module is trained to determine if adequate instructions and help are provided. Later, when evaluating the SUT, Agent A11y encounters a field labeled, “emergency contact phone number.” While this particular variation on a phone number field label may not have been seen by Agent A11y before, the trained system recognizes that the field represents a phone number, which it has previously learned should be paired with accessible input guidance consistent with that provided within its training set.

- Input guidance for the entry of a password specifies the length, types and number of characters that constitute a valid password. Using NLP and/or REGEX techniques, Agent A11y programmatically interprets the input guidance and actively validates that a password constructed according to the input guidance is accepted by the SUT.

- An image alt-text label is compared to the classification returned by an image classification module. If the alt-text label conflicts with the returned image classification, the alt-text label – image pair is flagged for further evaluation by a manual tester.

- A link to a common web page is provided in several places throughout the SUT. The accessible labels for the links are compared and inconsistencies flagged, see WCAG 3.2.4, Consistent Identification.

5.2 Form Recognition

While the system is currently able to recognize individual web page elements such as *First Name* and *Last Name*; oftentimes, it is useful to recognize a group of elements as a single semantic entity. For example, groupings of fields such as *Contact Information*, *Shopping Cart*, or *Midterm Grades* can provide important semantic information concerning the function, types, and value ranges of the group’s contained fields.

5.3 Dynamic Accessibility Testing

There are many promising avenues where new dynamic testing algorithms might successfully expand the range of WCAG success criteria which could be addressed through automated testing. WCAG’s Input Assistance Guidance category represents one area where substantial gains may be achieved as current static methods are insufficient. In particular, the Error Identification, Suggestion, and Prevention success criteria under WCAG 3.3 could all benefit from appropriate dynamic testing algorithms. While full verification of these success criteria is beyond the state-of-the-art of current machine learning and NLP capabilities, it is likely that many aspects of these success criteria may be evaluated using available techniques, at least for some significant percentage of situations.

6 Conclusions

At its core, AGENT A11y's ability to automatically explore a SUT, with all of its page and form variants, broadly supports autonomous, site-wide, accessibility testing. With Agent A11y, it is possible to simultaneously support multiple accessibility testing modules, strategies, oracles, and report aggregation capabilities to achieve more comprehensive WCAG coverage. Perhaps most critically, Agent A11y's modular architecture and site-wide exploration capabilities establish a framework for the application of new multi-page and dynamic testing algorithms suitable for evaluating additional WCAG Guidelines which presently require expensive manual testing. Certainly, much work remains to be done to develop testing techniques and oracles capable of evaluating many of the more complex WCAG criteria. However, Agent A11y has already demonstrated that an autonomous, self-exploring agent-based solution can expand the capabilities of current single-page, static accessibility assessment tools.

References

Chillarege, Ram, I.S.Bhandari, Jarir Chaar, M.J.Halliday, D.S.Moebus, Bonnie Ray, M.-Y.Wong. 1992. "Orthogonal Defect Classification - A Concept for In-Process Measurements." *IEEE Transactions on Software Engineering* 18: 943 - 956.

Aamodt, A., Plaza, E. 1994. “Case-based reasoning: Foundational issues, methodological variations, and system approaches.” *AI Communications* 7(1): 39–59.

AGENT. 2019. “AI Generation and Exploration in Test.” https://github.com/ultimatesoftware/AGENT (accessed August 16, 2019).

Bai, A., Camilla, H., Viktoria, S. 2017. “A Cost-Benefit Analysis of Accessibility Testing in Agile Software Development Results from a Multiple Case Study.” *International Journal on Advances in Software* 10(1,2). http://www.iariajournals.org/software/, (accessed August 27, 2019).

Becce, G., Mariani, L., Riganelli, O., Santoro, M. 2012. “Extracting widget descriptions from GUIs.” *International Conference on Fundamental Approaches to Software Engineering*: 347–361.

Levenshtein, V. I. 1966. “Binary codes capable of correcting deletions, insertions, and reversals.” *Soviet Physics* 10(8): 708–710.

Lin, J.-W., Wang, F., Chu, P. 2017. “Using semantic similarity in crawling-based web application testing.” *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*: 138–148.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., McClosky, D. 2014. “The Stanford CoreNLP Natural Language Processing Toolkit.” *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*: 55–60.

Pawlik, M., Augsten, N. 2015. “Efficient Computation of Edit Tree Distance.” *ACM Transactions on Database Systems*, vol. 40 (1), No. 3. https://dl.acm.org/citation.cfm?id=2699485, (accessed August 27, 2019).

Santiago, D. 2018. "A Model-Based AI-Driven Test Generation System.” *FIU Electronic Theses and Dissertations*: 3878. https://digitalcommons.fiu.edu/etd/3878, (accessed August 27, 2019).

“Selecting the number of clusters with silhouette analysis on k-means clustering.” https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html (accessed August 27, 2019).

Shubert, E., Rousseeuw, P.J., “Faster k-medoids clustering: improving the PAM, CLARA, and CLARANS algorithms.” https://arxiv.org/abs/1810.05691 (accessed August 27, 2019).

Veloso, M. 1992. ‘Learning by analogical reasoning in general problem solving.” *Carnegie Mellon University,* 40. https://apps.dtic.mil/dtic/tr/fulltext/u2/a256573.pdf (accessed August 27, 2019).

Vigo, J., Brown, J., Vivienne, C. 2013. “Benchmarking Web Accessibility Evaluation Tools: Measuring the Harm of Sole Reliance on Automated Tests.” *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*.

Acknowledgements

We thank Brian Muras and Justin Phillips for their contributions to Agent A11y and this paper.