Guidance for the development of accessibility evaluation tools following the Unified Software Development Process

Yod-Samuel Martín , Juan C. Yelmo

Abstract

Automated and semi-automated accessibility evaluation tools are key to streamline the process of accessibility assessment, and ultimately ensure software products, contents, and services meet accessibility requirements. Different evaluation tools may better fit different needs and concerns, accounting for a variety of corporate and external policies, content types, invocation methods, deployment contexts, exploitation models, intended audiences and goals; and the specific overall process where they are introduced. This has led to the proliferation of many evaluation tools tailored to specific contexts. However, tool creators, who may be not familiar with the realm of accessibility and may be part of a larger project, lack any systematic guidance when facing the implementation of accessibility evaluation functionalities. Herein we present a systematic approach to the development of accessibility evaluation tools, leveraging the different artifacts and activities of a standardized development process model (the Unified Software Development Process), and providing templates of these artifacts tailored to accessibility evaluation tools. The work presented specially considers the work in progress in this area by the W3C/WAI Evaluation and Report Working Group (ERT WG).

Keywords: accessibility evaluation, automatic evaluation tools, Unified Software Development Process, Evaluation and Report Language, business process model, domain model, use case model, analysis model, deployment

1. Introduction

Imagine you are a technology consultant who has just been commissioned by a large web provider to develop a module for a web application platform, so that non-technology-savvy content and service creators can be assisted to determine whether their creations are equally accessible for people with diverse abilities. That is, you need to develop an automatic accessibility evaluation tool, which will be introduced in a specific workflow and will need to abide by some business constraints. Maybe you are already acquainted with software development, but this is the first time you deal with this kind of tool. Or you have used them before in other projects, but have never tackled the development of your own evaluation tool. Which functionalities are expected of it? What is exactly meant by "evaluating accessibility"? How are the evaluation results to be handled? What should you start with? Although a definite answer to these questions depends on the varied requirements of the specific evaluation tool to be developed, dozens of such tools already exist, which allow extracting useful guidance on how the process of developing an automatic accessibility evaluation tool can be undertaken.

Based on our experience with existent evaluation tools, we aim at providing some insight to non-accessibility-experts on how to develop accessibility evaluation tools, following a standardized software development process. The work herein presented has taken into special consideration the Guidance on the Development of Web Accessibility Evaluation Tools (working title), a draft document which is currently being composed by the W3C/WAI Evaluation and Report Working Group (ERT WG). (Further information on that document and its evolution may be found at http://www.w3.org/WAI/ACT/deliverables#eval_tools.)

The next section will introduce our vision of accessibility evaluation tools and the process model we will follow; the rest of the paper will provide the specific guidance for different models employed in the process, namely: business process model, domain model, use case model, analysis model and deployment model. We have deliberately omitted references to specific tools, as we want this guidance to be as broadly applicable as possible.

2. Context and goals

Strictly speaking, the accessibility of a product can only be stated regarding specific user needs, goals and contexts of use[2]. If a certain user can access it under those conditions, it will be accessible *for him or her*. It is unreasonable to test this in each possible context of use (real or emulated), so a set of practical criteria have been agreed, based on community experience, as a reference to determine whether a product is accessible or not. Even then, it is not viable to manually test each and every criterion, so **accessibility evaluation software tools** have come to relieve the evaluator of that burden, by automatically analyzing a product and issuing judgments regarding its compliance with accessibility criteria. These tools greatly simplify the process, but cannot provide a definite replacement for human judgment. This is why they are sometimes called semi-automatic evaluation tools. For instance, a software tool may detect whether an image is served with an alternative text, but cannot decide if both convey an equivalent meaning, yet it can point the evaluators towards the image to get their expert assessment.

Same as any other software product, the development of an evaluation tool is usually structured as a project involving a set of activities, worker roles and artifacts. The organization of this project may respond to different process models, one of the most widespread being the **Unified Software Development Process (USDP)**. USDP[1] is characterized by being: driven by use cases, which are used as a tool to capture user requirements and intertwine the subsequent design, implementation and testing activities; centered on the architecture, which represents the internal form the product will take; and incremental and iterative, so that risks are mitigated early at a lower cost.

USDP can be considered a generic framework for software development, which is instantiated in each specific development project. In between, "project templates" can be created[3] which constrain, inherit and extend the process defined by USDP so as to account for specific software engineering concerns. This is the approach we will follow in this paper: show examples of USDP-defined artifacts particularized to the development of accessibility evaluation tools. The **artifact templates** we propose deal with the different disciplines dealt with by USDP: a business model and a domain model to kick off the process of requirement elicitation, a use-case model to represent the functional requirements, an analysis model to reflect an intermediate view of the architecture, and a deployment model to describe the physical bindings. These artifacts will still be abstract, in that they do not reflect the conditions of a concrete accessibility evaluation tool, but they offer general directions that can be applied to any of those tools whatsoever. That is why we talk about ***guidance***, which will still need other inputs from the specific business to come with the concrete artifacts in each case.

USDP was first published in 1999, condensing decades of experience in managing software development projects, while accounting for the latest industry best practices. Today, USDP is a modern-yet-mature process model which provides a detailed description of the process steps, and accommodates disparate organizational contexts. That is why we have chosen USDP to provide this development guidance, although it may be easily applied as well to other process models. For instance, the disciplines in each USDP iteration can be mapped to sequential phases in waterfall development models; conversely, USDP activities may be grouped differently, according to the definitions of the Agile Unified Process (with a special emphasis on test driven development). Likewise, although we use object-oriented design terminology to exemplify different software artifacts, other programming paradigms may be followed. For instance, from a procedural perspective, accessibility evaluation can be modeled as a dataflow that creates an evaluation report departing from some criteria and some content to be evaluated. Or, from a declarative perspective, accessibility criteria could be regarded as rules, etc.

Moreover, beyond the abstract functional specification we provide, the process template does not enforce any additional technological or architectural constraints. Although inspiration mainly came from web accessibility evaluation, it is defined in a technology-agnostic fashion and can be applied to other Information and Communication Technology realms. It can also be applied in different business and exploitation scenarios.

In summary, the **process template** we propose could be applied to the development of any (semi-) automatic accessibility evaluation tool —"auxiliary" evaluation tools which do not emit judgments at all lay out of this work.

3. Business process model

From the perspective of software engineering, the accessibility of a software product can be regarded as a category of quality properties[4] or requirements, which ensure that it can be used in equivalent ways (considering aspects such as comfort, security or cost) by people in the broadest range of contexts of use, specially accounting for users with different abilities. Thus, an organization typically uses accessibility evaluation tools in the activities dealing with requirements (e.g. specification, testing). The workflow followed by that organization may be captured by USDP in a **business process model**, which shows the different business actors and the activities they undertake to deal with work units and internal process data. This may be specific to the strategic goals pursued by the entity which commissioned the tool, yet we can define a set of activities where the tool will be typically used.

To define the business process for accessibility evaluation, we place it in the context of the business process of software development. Fortunately, USDP is itself described using the business process modeling technique, so we just need to identify the activities where accessibility evaluation is involved. Readers should note that more fine-grained detail could be provided for specific evaluation contexts. Likewise, some activities may not appear in all the scenarios, but we aim at providing a generic framework that may be as reusable as possible. Figure 1 shows an activity diagram where the different workers (business actors) are represented by different swim lanes, which show the workflow between the activities they perform, together with the entities shared to pass work units along. Activities related to accessibility evaluation are highlighted. We must remark this business process depicts the ***tool usage***, although it takes place itself ***within a larger process***: the development of the product under evaluation.

A development process starts with the workers in charge of requirement capture and elicitation specifying the system requirements (for which they will usually interact with the client teams), including accessibility requirements as well. Requirements are conveyed to the workers responsible for the bulk of the development work (system analysis, design and implementation). Further on, they may use the tool to validate that the products they create abide by the requirements previously defined by the analyst. Later, test engineers will perform tests in the large (e.g. integration tests) to the system version, using once more the evaluation tool to validate accessibility conformance. After internal tests pass, the system is released to the client, who may perform their own (acceptance) tests; and to external auditors, who may perform their own accessibility tests to issue conformance certificates.

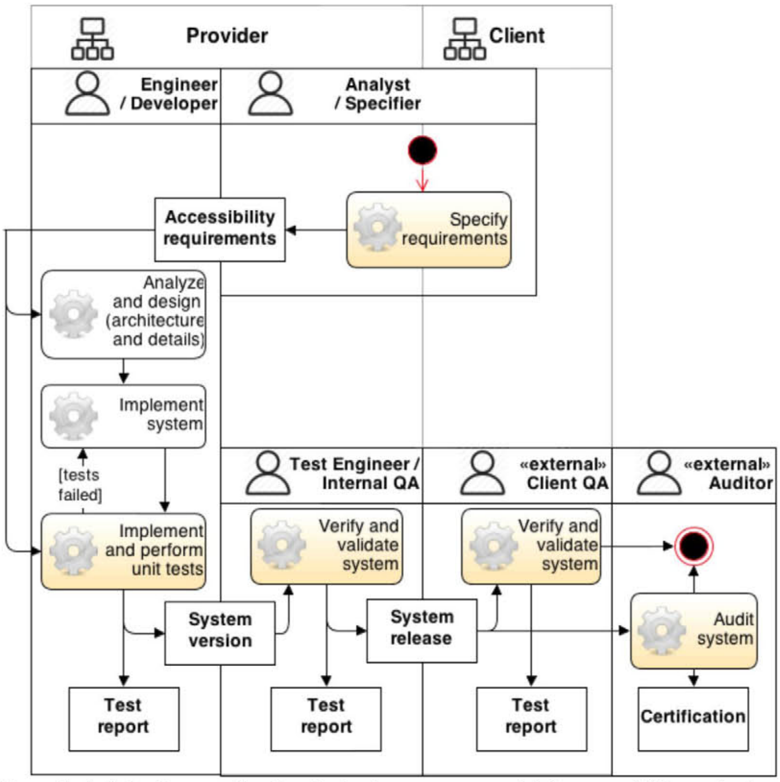

Figure 1. Activity diagram showing the business process model for accessibility evaluation.

4. Domain model

A domain model captures the entities appearing in the context of the system and the relationships between one another. Although each business context will contribute its own entities to the domain model, we can distil a subset that will appear in all accessibility testing tools. For that, we have departed from the Working Draft of the Evaluation and Report Language (EARL) [5] which defines an RDF vocabulary to be used for accessibility test reporting purposes, though it synthesizes cross-cutting concepts common to the whole accessibility testing process. Stripping out representation format details, we get the UML view of the core domain model of accessibility evaluation tools shown on Figure 2.

The core element of the accessibility evaluation domain is the **assertion**. It conceptualizes any result that may be output by the evaluation tool. A tool may state (assert) that e.g. "*the website at the domain example.com conforms to WCAG 2.0 at level AAA*", "*the content of the document index.html does not validate against the HTML5 formal grammar because the element <p id="1"> is not closed*", "*this Android application cannot be accessed using TalkBack*", "*the contrast of this image does not conform to the success criteria 1.4.3 of WCAG 2.0*", "*button b42 does not contain an instance of an AutomationProperties.Name*", or "*the font color used in the captions to render the dialogue of the main character conforms to UNE 153010:2012*". All these statements are examples of assertions, which contain the results of applying at least one test to some piece of data. Within an assertion, we may distinguish different elements.

A **Test Subject** is the product, service or (piece of) content to which the test is applied. The tool puts some specific subject into evaluation, and the assertion captures the result of applying the tests to that subject. It can be as fine- or coarse-grained as needed. Following the previous examples, a whole website, a document, an application, an image, a user interface input control, or an object attribute can all be regarded as test subjects.

A **Test Criterion** specifies the test that was applied. Again, test criteria may range from high-level abstract goals to tailored techniques only applicable in specific contexts. Several terms are usually applied to different kinds

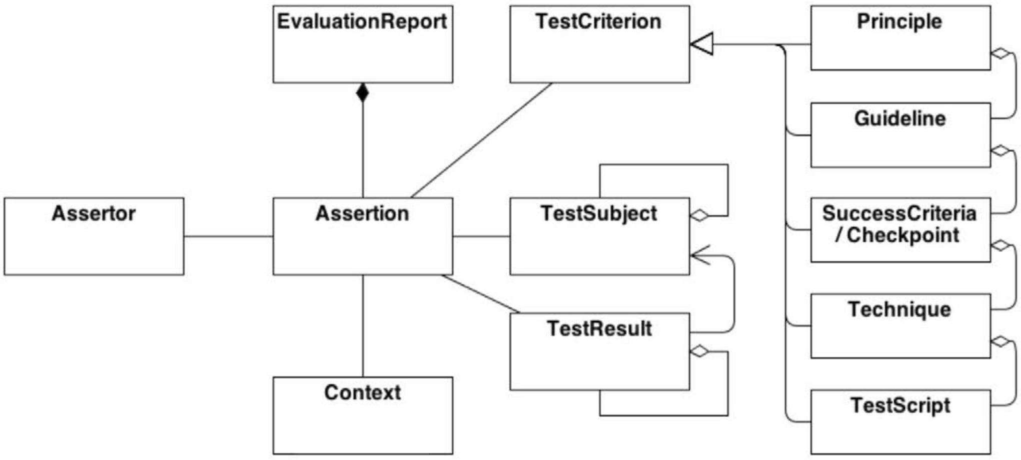

Figure 2. Class diagram showing the domain model for accessibility evaluation.

of criteria. ***Principles*** are abstract foundations which must be satisfied by any accessible product of some type, e.g. Principles of Universal Design, WCAG's (Web Content Accessibility Guidelines) POUR principles (perceivable, operable, understandable and robust). ***Guidelines*** define coarse-grained criteria. ***Success criteria*** or ***checkpoints*** refine them into assessable requirements against which an objective judgment can be issued, yet without binding to any specific technology. ***Techniques*** are applied to a specific content type or in a specific situation, and define detailed features the contents must (or should, or cannot) include in order to be accessible. Indeed, a tool will only support the techniques for the content types it deals with. ***Tests*** or ***test scripts*** refer to a specific procedure (set of instructions) to be executed when checking the criterion. They are sometimes called "test cases" as well, although this strictly refers to a set of conditions or variables under which a determined behavior is expected. Test criteria can be aggregated into families, which are usually compiled together and may be subject to standardization processes. Different families apply to different technologies (e.g. WCAG for web contents and applications) and can be considered test criteria per se as well.

A **Test Result** determines the outcome of applying the test criterion. If we draw a parallel between test criteria and logic propositions, test results will be usually expressed as boolean ***truth values*** (e.g. pass / fail). Results may be extended to other values, such as 'unknown', 'not applicable' or a certainty value continuously ranged between 0 and 1. These values would respectively appear in other logical systems such as Kleene's three-valued logic, description logics in e.g. the Semantic Web Stack, and fuzzy or subjective logics. Test Results may be enriched with additional information, such as ***pointers*** to the region of the Test Subject where the accessibility criterion was applied (and succeeded or failed), human-readable ***help*** to explain why the criterion has succeeded or failed, and suggested ***remedies or repair actions***, etc.

An **assertor** represents the issuer of the assertion: it can refer to the software tool, or the identity of the person or organization on whose behalf the tool is working.

An assertion can be enriched with further information detailing the **context** where it was obtained: a description of tool features, the mode by which the results were obtained (manual, automatic, semi-automatic), a timestamp…

5. Use case model

USDP employs use cases as a tool to define the functionality of a system. Each use case defines a sequence of actions between an external actor and the system, which provides some value to the actor or obtains value from it. Different evaluation tools may define their own actors and use cases, but at least some of them will extend the abstract use cases we define herein. Apart from the experience with evaluation tools, we have derived these use cases from generic software testing standards[6]. The three main uses cases of an evaluation tool deal with specification, processing and presentation of the evaluation; extensions include revision of the results.

The main goal of an accessibility evaluation tool is to automate (at least partially) the process of evaluating the accessibility of a product, service or content. Thus its core use case is the processing of the evaluation upon request of a Test Requester. In the simplest scenarios, this use case includes as well the other two (specification and presentation), while in more advanced workflows, different actors will independently invoke each use case at different moments. These actors may show different features depending on the specific scenario where the evaluation tool will be used. Actors may also represent other software systems, e.g. a business rule engine automatically invoking a tool, or an auditing system into which results are fed. The evaluation results may also be later used for other purposes, which may include repairing the errors detected. Analysts will thus to account for all these aspects, and tightly cooperate with the client or business units in order to define the specific requirements of their evaluation tool, defining and prioritizing a required subset of the tool features and use cases.

5.1. Specification use case

In order for an accessibility evaluation to be run, the test designer first needs to specify what is exactly meant with evaluation. Observing that assertions associate test subjects and test criteria, accessibility evaluation tools will at least need to allow test designers to specify (configure) which subjects and criteria will be used in the evaluation.

Test Subjects may be specified either individually or in bulk. In the first case, the Test Designer may in turn define the individual Test Subject in several ways. The most precise consists in directly providing the tool with a binary or textual ***serialization*** of the content or product under evaluation. In that case, the evaluation tool must provide the Test Designer with interfaces to input that serialization (e.g. an editable field, an http-post endpoint). It is up to him to get the content serialized, for which he may first resort to external tools if needed.

In the sake of convenience for the Test Designer, the evaluation tool may integrate with the ***APIs of the user agent*** the Test Designers employs to retrieve contents, so that they just need to instruct the tool to get the "currently displayed content" as Test Subject. User agent APIs can also be used to upload locally hosted files to remote tools.

Alternatively, the Test Designer may specify a ***unique identifier*** of the Test Subject, so that the tool may retrieve it when (or before) processing the evaluation. In order for that identifier to work, it must be part of a namespace shared between the Test Designer and the tool and accessible by both. For instance, paths to the local file system can be used if both are working on the same local node, and have equivalent access permissions. URLs are also good candidates, but there is no guarantee that a server will serve the same contents to different clients requesting the same URL. A detailed specification may account for ***context attributes*** which may determine the result provided by the server, e.g. user agent properties (user agent version and platform, preferred content types and languages, ...), user credentials, and browser state (cookies, cached versions, …). Some context attributes are difficult to emulate (e.g. request time) or even impossible (IP address). Moreover, the content of the Test Subject

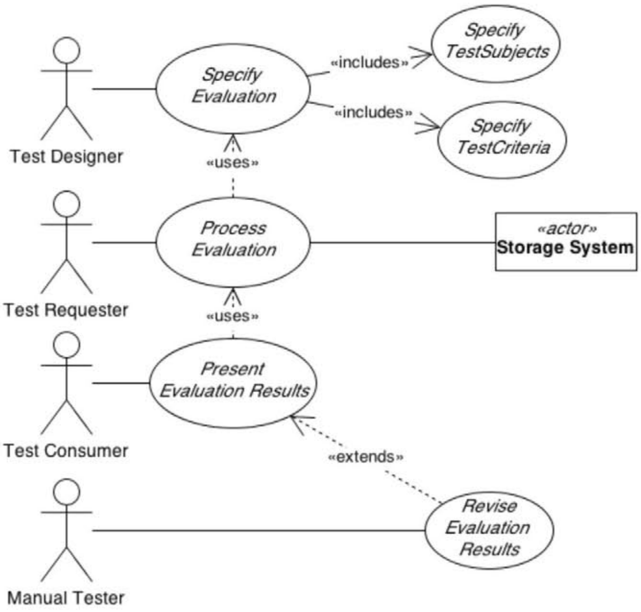

Figure 3. Accessibility evaluation use cases.

may change since the test is specified until it is executed; hence a potential trade-off may exist between the precision and the simplicity of the specification. To test some dynamic behavior or stateful products, more complex ***test fixtures*** can be defined, which may include a set of actions (retrieve several pieces of content in a specific order, generate user interface input events, interpret software scripts), before the test execution. This may even require manual user intervention to lead the evaluated system to the desired status.

It is usual that Test Subjects are not specified in isolation, but aggregated as part of a larger subject. Two typical ways to specify Test Subjects in bulk are ***link traversal*** and ***pattern-defined*** identifiers (both may be combined).

Many Test Subjects include (hyper-)links to other pieces of data: web documents that link to one another other, source or object code files which import other software modules, software controllers described in imperative languages which load declarative models or view descriptions, etc. It is customary that Test Designers will be interested in evaluating not only a Test Subject, but also all those linked from it. This process may be applied recursively, which requires in turn specifying termination conditions that prevent an unbounded recursion: e.g. select only those Test Subjects hosted in a specific realm (e.g. a local host, a specific web host, an organization DNS domain), traverse only those links to inline or embedded contents, follow links just to a depth of n levels, etc.

Pattern-based specification allows Test Designers to select at once all those Test Subjects whose identifiers match some kind of pattern (e.g. a regular expression). Patterns may be used to select contents of a specific type (based on its file extension), on a specific directory, etc. Pattern application requires an underlying query mechanism to be available which guarantees that all contents matching a pattern may be retrieved. Distributed systems such as the Web do not usually provide these functionalities, so pattern matching is usually combined with link traversal, effectively crawling web sites as a heuristic way to retrieve all their contents (same as what search engines do when indexing web sites). Even after the pattern is applied, the tool may come up with a large amount of potential Test Subjects. ***Filtering*** and ***sampling*** functionalities, either automatic or assisted by the Test Designers may help in creating a manageable set of subjects.

Test Criteria may as well be specified in bulk or on a case-by-case basis. Evaluation tools must offer Test Designers a set of pre-configured Test Criteria that reflect ***industry-wide accepted criteria families***. On the one hand, this simplifies the activity of specification; on the other, tools can foster the adoption of these widely recognized principles that distil the acquired knowledge of the community, rather than ad hoc criteria the Test Designers may decide on their own. Some families define different ***levels of conformance***, which account for the fulfillment of larger (stricter) or smaller (looser) sets of criteria. Tools should enable Test Designers to select the appropriate level of conformance against which the tool will evaluate the Test Subjects.

Test Designers may have a legitimate interest to only test the compliance of ***specific Test Criteria***. Tools may provide them with the possibility to choose Test Criteria one by one, or just those applicable in a specific context of use (e.g. for a user with a specific disability profile), but they must avoid misleading designers by clarifying that meeting single accessibility criteria does not imply the product being accessible at all. Test Designers may also wish to provide their own ***Test Script implementations***, which may be supported by tools that define a ***Test Criteria provider interface*** that may be extended by third-party developed implementations that act as plug-ins for the main tool. This may be the case to e.g. support new technologies or account for specific deployment contexts or corporate policies. New tests may also be specified following a ***declarative format***, providing, e.g. query expressions and conditions their results must abide by to meet accessibility requirements. Nonetheless, tools should encourage that any newly defined criteria preferably fits within existent success criteria, guidelines and principles, in that order, so as to discourage the definition of “unusual” approaches to accessibility.

Specification may also configure ***other properties***: events triggering the evaluation process, evaluation schedule, routing of the results, etc. The specification might also be split between ***two phases***: first a system analyst would provide a partial specification for test subjects (e.g. a template defining content types under evaluation), and then a developer would specify the specific software artifacts to be evaluated after they have been developed.

5.2. Processing use case

Once the test criteria and subjects have been specified, the proper evaluation process may start. The execution use case may be split into several steps: triggering, retrieval, execution, and reporting.

Triggering: Evaluation may be triggered either directly by the Test Requester or by an authoring tool he or she is using (e.g. a Content Management System, an integrated development editor).

Retrieval: Test Subjects must be retrieved from an external storage system, according to the previously configured specification. If identification was provided for test subjects, a transport mechanism must be used to retrieve them. In order to retrieve subject contents, all the context and test fixtures defined during configuration must be first set up. If link traversal had been specified, contents must first be analyzed to find (and resolve) links.

Execution: Each test criterion is applied to all the test subjects where it is pertinent. For each test subject, a typical structure is: 1) applicability rules define which points of the subject may validate / break a criterion; 2) a test procedure is applied to each point; 3) the results are compared with expected values that determine the potential test results. Semi-automated criteria may need (sometimes) Test Requester help regarding the points, procedures and/or conditions. Their involvement may include interactive interventions, postponed until the evaluation has ended, or be simply ignored and an "unknown" value be emitted. Heuristic procedures may be applied which provide a somehow trustworthy yet not definite answer.

Reporting: The results of each evaluation process execution may be represented as assertions, and stored following standardized formats (e.g. EARL). Although the initial Test Subject specification may be quite generic, detailed information about the ***effectively evaluated subjects*** may be included in the report. For instance, the specification might only establish that all the HTML documents in the web site www.example.com will be evaluated, while the report would provide a detailed list of the HTML files that were effectively evaluated, and potentially the exact HTTP requests that were issued to retrieve them. Reports may include ***aggregate measures***, such as conformance to larger families, or comprehensive numeric accessibility measurements. Numeric scores should account for potential barriers[7] that are correctly tackled (i.e. did not materialize), rather than simply focusing on the number of criteria that pass or fail. Care should be taken to avoid presenting a 50% success rate as a "decent" accessibility (while it truly means that many users will encounter one or another accessibility barrier). Nonlinear and weighted scoring algorithms can be applied to avoid that. Although scores do not offer a total order metric, they should be monotonous regarding the (dis-)appearance of potential barriers. Different scores can also be provided for different user ability profiles, kinds of criteria, content types, groups or families, etc. Reports may also include information about ***causes, symptoms and remedies***[8]. Going beyond evaluation, the tool output may automatically modify the contents so as to correct accessibility barriers, becoming what is known as a ***repair tool***.

5.3. Presentation use case

After an evaluation has been performed, results will be presented to a Test Consumer, who may or may not be the same person who ran the evaluation. For instance, test engineers may design and run evaluation tests, and then return the results to the development team so that they can fix any issues. Or an accessibility consultant may present the results of an accessibility evaluation to their clients. Or a company may be presented with the results of the accessibility evaluation of a corporate site developed by a contractor. Or a user association could get the report of a third-party site created by an external auditor.

Results are reported and stored following standardized formats. But their presentation can be arranged in different ways depending on the consumer needs. In any case, the result format must match with the consumer's context of use, e.g. HTML or PDF will be used for human consumers, standardized languages for software tools.

A ***detailed presentation*** includes all the results of the evaluation in a human-consumable format. Apart from a linear view of the results (displaying one assertion after another), navigability mechanisms should be provided for human consumption. Tabular structures may ease the orientation through the different results. Sorting, grouping, querying and filtering mechanisms ease locating the assertions with specific criteria, conformance levels or result values. Collapsible trees hide any unwanted details while allowing users go deeper into results of their interest, etc.

Summarized results can also be provided: numeric scores, aggregated tables per criteria, priority or document; infographics such as radar charts, etc.

Test Results can also be rendered together with ***the original content***. For instance, an icon representing a Test Result can be superimposed wherever a Test was applied, so that consumers may easily locate the points that have yielded failed or inconclusive results. These icons must comply themselves with accessibility criteria and may link to detailed explanations. Likewise, annotations may decorate the source code on a development environment, so that accessibility violations are treated in the same way as other software construction errors or warnings.

5.4. Revision use case

Finally, Test Results can also be modified by human testers (evaluators), which may provide additional data for test criteria that require manual intervention (as indicated by the test results), or even to rectify wrongly assessed criteria. The tool must keep ***provenance*** data that help tracking the responsible for each assertion. The tool output (e.g. reports, scores) must be recomputed after modified Test Results are saved.

6. Analysis class model

Analysis activities provide a preliminary model of the internal classes that will implement the functionalities of the different use cases. It is richer than the domain model, including classes that reflect technical concepts and architectural features; but is not yet as detailed as the design model. Our domain model already anticipated some analysis classes: a real client would have usually preferred terms from their domain (e.g. multimedia content) to deal with the problem realm, while accessibility specific terms (e.g. test subject) would only arise later.

Figure 4 below presents our analysis class model for evaluation tools, using three class stereotypes: entity, boundary and control (akin to the model-view-controller pattern). We may note how the tool must provide different user interfaces (**boundary classes**) for each of the supported use cases. Two **entities** model the configuration parts (test subject and test criteria). While the configured criteria are directly fed to the test executor, the test subject configuration needs first to pass through a retrieval task that obtains the contents of all the resources to be evaluated (according to the test subject configuration). **Control classes** are defined for each of the main activities defined in the use cases: configuration, content retrieval, evaluation execution, reporting and presentation.

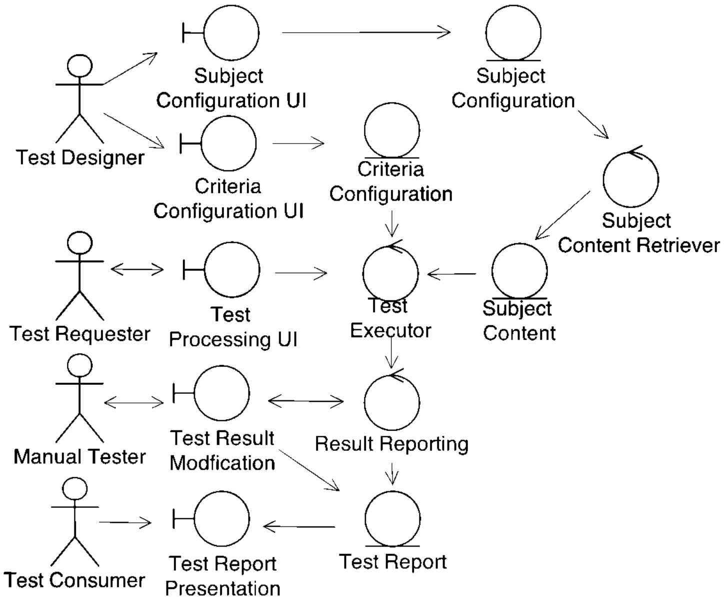

Figure 4. Analysis class model for accessibility evaluation tool.

7. Deployment models

USDP deals with deployment planning as part of the design activities. A deployment model defines the physical architecture, identifying the active classes (processes) that will be created in the system, the physical nodes (hosts) where they will be running and the communications mechanisms among them. Evaluation tools may follow varied deployment alternatives, thus a particular reference model is of little use. Nonetheless, most can be subsumed under two major deployment alternatives: as either local applications or remote/distributed services. The chosen deployment model may also impact other aspects such as exploitation, scalability, etc.

Local applications are stored and run on the same device the Test Requestor directly operates. Although they may access external services to retrieve remote data (e.g. Test Subject contents), all the operations directly related to the evaluation process are performed locally, including the presentation of the results. Local applications need to abide by the platform requirements imposed the target environment (operating system, component dependencies, user permissions). They may either be executed as ***standalone software*** or require a ***previous installation***. A special case for local applications is that of **browser add-ons**: their target environment is a sandboxed platform provided by the browser. Browser add-ons have their control user interface tightly integrated within the browser, and may delegate on it some functions such as content retrieval or result rendering.

Remote services, on the contrary, expose a functionality to be accessed from different hosts, with which they interact to pass input and output data back and forth. For instance, following a Service Oriented Architecture (SOA), the tool may provide a Web Services or a RESTful interface to offer its functionalities. Input data (test criteria and subject configuration) are passed through this interface and output data (test results) are returned through it as well. Remote services may also include a ***presentation layer*** that retrieves the configuration from an HTML form and returns the results as web pages to be rendered by the user's browser (making what is usually known as a web application). Conversely, invocations to remote services may be triggered from controls added to the user's browser (e.g. *scriptlets*, *favelets*, menu add-ons), which dynamically generate the necessary configuration based on the current ***browsing context***.

An **intermediate case** is that of evaluation tools which run on the **server side,** but do not interact with the client. For instance, a tool may be a module of a Content Management System that validates the accessibility of the contents it stores. The tool runs on an application server, but is only invoked from within.

8. Further considerations

Other USDP features may be leveraged to implement the large amount of fine-grained techniques generic evaluation tools should deal with. Packages may be defined for related techniques (e.g. dealing with the same content type, or implementing the same criterion), successive iterations may implement different techniques, etc.

The USDP defines many other artifacts and activities that will need to be considered when developing an accessibility evaluation tool. The specification activities will need to count with: a broad glossary that goes beyond the domain model; user interface prototypes that help communicate the user perspective to the client since the beginning; and the specification of non-functional requirements. Regarding the last point, it is of paramount relevance to make the evaluation tool accessible itself, although other aspects should not be dismissed, such as the performance of tools which provide developers with real-time feedback, or the localization of presentations. In any case, workers will be needed during specification who can grasp the knowledge of the specific business and users.

More specific material exists which applies these principles to different realms. Regarding web accessibility evaluation tools, supplementary online material is available with a detailed list of useful documents.

References

1. Jacobson I, Booch G, Rumbaugh JE (1999). *The unified software development process.* Addison-Wesley Professional; 1999.
2. Magnabosco P (ed.) Resolutions adopted during the 2011 Warsaw meeting. ISO/IEC JTC 1/SC 35. La Plaine Saint-Denis: AFNOR; 2011.
3. Jacobson I, Bylund S. Building your own process by specializing a process framework. In: *The Road to the Unified Software Development Process*, rev. up. ed. Cambridge: Cambridge University Press; 2000.
4. ISO/IEC 25010:2011. Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models. Geneva: International Organization for Standardization; 2011.
5. Abou-Zahra S (ed.), Henry SL (ed.). Evaluation and Report Language (EARL) Overview. World Wide Web Consortium, 2011. www.w3.org/WAI/intro/earl
6. IEEE Standard for Software Test Documentation. IEEE Std 829-1998
7. Freire AP, Fortes RPM, Turine MAS, Paiva DMB. An evaluation of web accessibility metrics based on their attributes. In: *Proc. of the 26th annual ACM international conference on Design of communication (SIGDOC '08),* 73-80. New York: ACM; 2008.

8. Brajnik J. Beyond Conformance: The Role of Accessibility Evaluation Methods. In Hartmann S, Zhou X, and Kirchberg M (Eds.) *Proc. of the 2008 international workshops on Web Information Systems Engineering (WISE '08)*, 63-80. Berlin, Heidelberg: Springer-Verlag, 2008.

Appendix A. Relation with World Wide Web Consortium Web Accessibility Initiative (W3C WAI) documents

The process above described is agnostic regarding the technology of the test subjects the tool will evaluate. Nonetheless, specific guidance material exists which targets web contents, mainly provided by W3C's Web Accessibility Initiative. In particular, the Evaluation and Repair Tools Working Group (ERT WG) is creating some work-in-progress material that may become quite useful for the purposes herein described, quite especially the ***Guidance on the development of web accessibility evaluation tools*** [1]. Although the authors actively contribute to ERT WG, the positions here presented solely express their views, and do not imply any endorsement by W3C.

Table 1 lists different documents from W3C WAI and others, which can be useful to apply the suggested guidance to web contents. Much of them can also be applied to mobile and desktop operating systems natively using web technologies in their applications, such as Windows 8 or Firefox OS.

Table 1. Reference documents applicable as input to the different artifacts of the USDP for accessibility evaluation tools.

Whole process and use case model	Guidance on the development of web accessibility evaluation tools [1] [In progress] Selecting Web Accessibility Evaluation Tools [2] [Draft] Web Accessibility Evaluation Tools [3]
Business process model	Requirements for "Techniques for Automated and Semi-Automated Evaluation Tools" [4] [In progress] Evaluation Approaches for Specific Contexts [5] D-WAB4 Unified Web Evaluation Methodology (UWEM 1.2 Core) [6] [Sections 2, 3] Website Accessibility Conformance Evaluation Methodology (WCAG-EM) 1.0 [7] Selecting Web Accessibility Evaluation Tools [8] [Draft]
Domain model and glossary	Evaluation and Report Language (EARL) 1.0 Schema [9] [In progress] Developer Guide for Evaluation and Report Language (EARL) 1.0 [10] [In progress] Pointer Methods in RDF [11] [In progress] Media Fragments URI 1.0 (basic) [12] [W3C Recommendation] Template for Accessibility Evaluation Reports [13] D-WAB4 Unified Web Evaluation Methodology (UWEM 1.2 Core) [6] [Section 6]
Specification use case	HTTP Vocabulary in RDF 1.0 [14] [In progress] Representing Content in RDF 1.0 [15] [In progress] Website Accessibility Conformance Evaluation Methodology (WCAG-EM) 1.0 [7] [Steps 1-3] D-WAB4 Unified Web Evaluation Methodology (UWEM 1.2 Core) [6] [Section 4]
Evaluation processing use case: Execution step	D-WAB4 Unified Web Evaluation Methodology [16] (UWEM 1.2 Tests) Techniques For Accessibility Evaluation And Repair Tools [17] [Abandoned draft] [Both obsolete, as they refer to outdated WCAG 1.0, yet useful as references] Website Accessibility Conformance Evaluation Methodology (WCAG-EM) 1.0 [7] [Step 4] Test Case Description Language 2.0: Specification and Guide [18]
Evaluation processing use case: Reporting step	Developer Guide for Evaluation and Report Language (EARL) 1.0 [10] [In progress] Evaluation and Report Language (EARL) 1.0 Schema [9] [In progress] Pointer Methods in RDF [11] [In progress] Media Fragments URI 1.0 (basic) [12] [W3C Recommendation] HTTP Vocabulary in RDF 1.0 [14] [In progress] Representing Content in RDF 1.0 [15] [In progress] Template for Accessibility Evaluation Reports [13] D-WAB4 Unified Web Evaluation Methodology (UWEM 1.2 Core) [6] [Sections 5, 9, 10] Website Accessibility Conformance Evaluation Methodology (WCAG-EM) 1.0 [7] [Steps 4-5] Research Report on Web Accessibility Metrics [19] [Advanced]
Presentation use case	D-WAB4 Unified Web Evaluation Methodology [6] (UWEM 1.2 Core)
Testing plan (of the tool)	WCAG 2.0 Test Samples [20] [In progress]

Appendix References

1. Velasco CA (ed.), Ackermann P (ed.), Vlachogiannis E (ed.) Guidance on the development of web accessibility evaluation tools. World Wide Web Consortium, 2013. [In progress] www.w3.org/WAI/ER/WD-AERT/ED-AERT
2. Abou-Zahra S (ed.) Selecting Web Accessibility Evaluation Tools. World Wide Web Consortium, 2005 [Draft] www.w3.org/WAI/eval/selectingtools.html
3. Abou-Zahra S (ed.) Web Accessibility Evaluation Tools. World Wide Web Consortium, 2006 [Draft] www.w3.org/WAI/RC/tools/Overview
4. Velasco CA (ed.) Requirements for "Techniques for Automated and Semi-Automated Evaluation Tools". World Wide Web Consortium, 2013. [In progress] www.w3.org/WAI/ER/WD-AERT/ED-requirements
5. Abou-Zahra S (ed.) Evaluation Approaches for Specific Contexts. World Wide Web Consortium, 2005 [Draft] www.w3.org/WAI/eval/considerations.html
6. Velleman E, Nietzio A, Velasco CA. D-WAB4 Unified Web Evaluation Methodology (UWEM 1.2 Core). Web Accessibility Benchmarking Cluster, 2007. www.wabcluster.org/uwem1_2/UWEM_1_2_CORE.pdf
7. Velleman E (ed.), Abou-Zahra S (ed.) Website Accessibility Conformance Evaluation Methodology (WCAG-EM) 1.0. World Wide Web Consortium Working Draft, 2013 [In progress] www.w3.org/TR/WCAG-EM/
8. Abou-Zahra S (ed.) Selecting Web Accessibility Evaluation Tools. World Wide Web Consortium, 2005 [Draft] www.w3.org/WAI/impl/software
9. Abou-Zahra S (ed.) Evaluation and Report Language (EARL) 1.0 Schema. World Wide Web Consortium Working Draft, 2011 [In progress] www.w3.org/TR/EARL10-Schema/
10. Velasco CA (ed.), Abou-Zahra S (ed.) www.w3.org/TR/EARL10-Guide
11. Iglesias C (ed.), Squillace M (ed.) Pointer Methods in RDF 1.0. World Wide Web Consortium Working Draft, 2011. [In progress] www.w3.org/TR/Pointers-in-RDF10/
12. Troncy R (ed.), Mannens E (ed.), Pfeiffer S (ed.), Van Deursen D (ed.), Hausenblas M, Jägenstedt P, Jansen J, Lafon Y, Parker C, Steiner T. Media Fragments URI 1.0 (basic). World Wide Web Consortium Recommendation, 2013 [In progress] www.w3.org/TR/media-frags/
13. Brewer J (ed.), Template for Accessibility Evaluation Reports. World Wide Web Consortium, 2002. www.w3.org/WAI/eval/template.html
14. Kock J (ed.), Velasco CA (ed.), Ackermann P(ed.) HTTP Vocabulary in RDF 1.0. World Wide Web Consortium Working Draft, 2011 [In progress] www.w3.org/TR/HTTP-in-RDF10/
15. Kock J (ed.), Velasco CA (ed.), Ackermann P(ed.) Representing Content in RDF 1.0. World Wide Web Consortium Working Draft, 2011 [In progress] www.w3.org/TR/Content-in-RDF10/
16. Velleman E, Nietzio A, Velasco CA. D-WAB4 Unified Web Evaluation Methodology (UWEM 1.2 Tests). Web Accessibility Benchmarking Cluster, 2007. www.wabcluster.org/uwem1_2/UWEM_1_2_TESTS.pdf
17. Ridpath C, Chisholm W. Techniques For Accessibility Evaluation And Repair Tools. World Wide Web Consortium Working Draft, 2000 [Unfinished, discontinued work] www.w3.org/TR/AERT
18. Strobbe C. Test Case Description Language 2.0: Specification and Guide. Benchmarking Tools and Methods for the Web, 2009. www.bentoweb.org/refs/TCDL2.0.html
19. Vigo M, Brajnik G, O Connor J. Research Report on Web Accessibility Metrics. World Wide Web Consortium Working Draft, 2012. www.w3.org/TR/accessibility-metrics-report/
20. WCAG 2.0 Test Samples. World Wide Web Consortium Draft, 2009 [In progress] www.w3.org/WAI/ER/tests/