Environment for Automating the Evaluation of Web Application Interfaces against Guidelines

R. J. Pereira, K. K. Gunewardane, C. J. Wickramarathne

Abstract—The increasing demand to check web application interfaces for design issues has provoked interest in tools to support the evaluation process including and not limited to assessing the design, consistency, and suitability of colour combinations. Based on literature gathered in the field of web usability and results of a survey conducted with web design professionals, it was revealed that the most emphasized issues are concerned with the limitation of detecting issues in user interfaces during the time of development itself. The paper introduces an environment which automates the evaluation of web interfaces against recommended guidelines. The Human Computer Interaction (HCI) guidelines should be in a formal specification in contrast to its natural language description in order to uniformly process them. Accordingly, the paper describes mechanisms to detect and automatically fix these issues. To achieve the abovementioned outcome, a set of algorithms and data structures were developed for which in-depth analysis has been performed in the paper. In order to assess the success of the proposals, the environment has been applied on an add-in for the Microsoft’s Visual Studio IDE, to evaluate web interface issues in websites under development of the ASP.NET framework.

Keywords—ASP.NET - Active Server Pages of.NET Framework, CSS - Cascading Style Sheet, HCI - Human Computer Interaction, UI - User Interface, XML - Extensible Mark-up Language

I. INTRODUCTION

Since the recent past, the use of websites and web applications has consistently and continuously elevating, especially with the dawn and expansive use of the Internet. Consequently, web applications have been provoked into numerous contexts ranging from commercial to educational websites. The most fundamental goal of any website is to ease the user experience for its intended audience.

An initial survey was conducted with a sample size of 45. Participants included web developers, web designers and undergraduate students in the Information Technology of Software Engineering faculties. The survey was focused on proving an existence in the deficiency of a proper means of design evaluation. Following are the results:

- 40.70% of the sample has stated that the most difficult aspect of designing a website is to maintain consistency while another 33.33% have stated that the alignment and layout of web controls are the most difficult aspects.
- Although 96.30% of the participants agree that usability of the interfaces are important, a significant 37.00% are under the impression that a usable website is one that simply meets the requirements of the client.
- 30.80% of the participants are not familiar regarding with the existence of guidelines to be adhered to and 46.20% simply ignore guideline compliance evaluation simply because it is an optional process, which can be quite tedious to be done manually.

R. J. Pereira is with Sri Lanka Institute of Information Technology, New Kandy Road, Malabe, Sri Lanka (phone: 00 94 2802 473; e-mail: julian_p008@hotmail.com).
K. K. Gunewardane is with Sri Lanka Institute of Information Technology, New Kandy Road, Malabe, Sri Lanka (phone: 00 94 2637 696; e-mail: kishankalhari@gmail.com).
C. J. Wickramarathne is with Sri Lanka Institute of Information Technology, New Kandy Road, Malabe, Sri Lanka (phone: 00 94 241 3900; e-mail: jagath.w@shit.lk).

- One of the most outstanding results of the analysis is 85.20% believe that user interface issues need to be considered during the time of designing the website. However, as discussed in Chapter II, most available tools support for web interface evaluation after development of same. Further to this, 92.60% of the sample has mentioned that they do not use any tool during the designing of their websites to evaluate user interface issues.

The survey statistics proved the neediness of a tool which evaluates the usability issues of user interfaces at the designing phase. The environment proposed in this paper has been adopted in the tool named twUider (Tool for Web User Interface Design Evaluation and Resolution) which is an add-in to Microsoft Visual Studio 2010 Integrated Development Environment (IDE). This tool will make use of this environment to evaluate the web designs at the time of design and it will evaluate the designs according to a common set of web standard guidelines which have been published by well-reputed organizations such as HHS (U.S. Department of Health and Human Services) [1], ISO (International Standards Organization) [2], W3C (World Wide Web Consortium) [3] and IEEE (Institute of Electrical and Electronics Engineers) [4].

This paper proposes a structure to represent a guideline from a natural language description in a formal specification without hardcoding the logic of the guideline in the detection process. Further to this, the UI Specialist can customize which guidelines the designers of the organization need to adhere to in their web designs. Once the guidelines have been set up, the paper proposes a generic algorithm to detect the deviations from the customized guidelines specified in the previous structure. In addition to detecting the deviations, the paper suggests a mechanism to fix the detected deviations automatically, whenever possible. Detecting and suggesting
fixes are performed on web interface items including files with extensions “.aspx”, “.ascx”, “.master” and “.css”. The aforesaid tool implements these structures and algorithms, and before applying the suggested fixes, using the tool the designer can preview the side-by-side view of the design as to how it looks is rendered currently and how it will be rendered if the automatic fixes were applied.

Fig. 1. shows the overall design of this environment and how each component is connected to the other.

II. RELATED WORK

To support designers in their drive to achieve more usable web interfaces, web usability guidelines were proposed by usability and web experts and organizations, and many evaluation tools were developed to enable designers and evaluators verify the compliance of their designs. Examples of such tools include Bobby [5], A-Prompt [6], WebSat [7] and 508 Accessibility Suite [8]. The common foundation for all these tools is that they are used to detect user interface issues by analyzing the Hypertext Mark-up Language (HTML) elements in the web page against some pre-defined conditions that form a guideline.

Jakob Nielson, Ph. D, who is addressed as the king of usability by the Internet Magazine [11], has established the —discount usability engineering [12] movement for swift improvement of user interfaces. Thus, it is evident that individuals may adopt certain guidelines through experience.

Most usually, the logic of evaluating each guideline is configured directly within the source code. According to Abdo Beirekdar et al [10], the major concerns of this hardcoding approach include, first, that when these guideline evolve to cater to the technological demands, adding a guideline or modifying an existing one will require the program’s source code to be modified, re-built and re-deployed. Second, this mechanism does not provide an opportunity for designers to define their own guidelines. This is specifically vital for web designing organizations that inculcate private standards and guidelines for the designs created by them.

In addition, many of these tools do not offer possibilities of controlling the evaluation process like choosing which guideline to evaluate, the level of evaluation at evaluation time, or the level of priority. For example, Bobby only provides the choice of the guidelines set to evaluate: W3C or Section508. [5]

Additionally, a tool named Magenta [9] was introduced. The Magenta tool, which is based on the Java platform, implements an Extendible Mark-up Language (XML) solution, allowing users to check their websites either with the acclaimed Web Content Accessibility Guidelines (WCAG) 1.0 or even their very own guidelines. These guidelines can further be enabled or ignored when checking. The aforementioned paper limits its scope to basic HTML tags. The tool also supports, at an elementary level, repair of guideline mismatches. Leporini et al. suggest an XML-based storage mechanism to define guidelines formally so that this formal specification can be used by the tool for comparison purposes [9]. Additionally, they propose that using XML allows for greater flexibility and interoperability [15] allowing the user to easily enhance the guideline to suit custom policies.

Scott Henninger discusses in his paper of architecture to define and customize guidelines. He proposes a general architecture consisting of a guideline hierarchy, cases and rules [13]. Accordingly, each time a guideline is selected to be applicable to a project, cases are used to define how that guideline can be applied to that particular project alone.

In [10], a six-step approach is discussed to define a guideline. First is to determine interesting elements, then to classify them into evaluation sets from which each atomic set is defined. The fourth step is to define evaluation conditions and then, clean up these atomic sets. Finally, the guideline evaluation interpretations are defined. The paper states that structuring guidelines under an XML-compliant format enables realization of mathematical proof of properties (such as completeness and consistency) and to benefit from the existing XML audience and tools suite [10].

In general, this review of the state-of-the-art emphasizes on the lapse of tools that are able to provide integrated flexible support by defining custom guidelines, detection of issues during the time of design for an particular IDE, previewing and repairing. This proposed environment caters to common guidelines introduced by four major standard bodies and these guidelines mainly focus on the consistency and layout of the websites being designed in Microsoft Visual Studio 2010.

III. GUIDELINE DEFINITION AND CUSTOMIZATION

The proposed environment will allow the UI Specialist to define new web interface guidelines to cater to the individual or organizational policies. Additionally, the UI Specialist can select which of the available guidelines are applicable to their organization or project. The selected set of guidelines in this customization needs to be adhered to by the designers of the organization. This section describes the structures involved in this process.

A. Mapping Guidelines to Formal Definition

In order to avoid embedding every guideline’s logic within the code, it was required to develop a structure to define these guidelines in a standard method. This structure should be applicable to all guidelines so that there is a common way to store the guideline definition and to detect deviations in web interface designs in a uniform logic. Due to the requirement
for flexibility in this context, XML was selected to develop this structure.

This structure was derived by an incremental development considering a sample set of initial guidelines. Each guideline was taken into account and the XML structure was improvised to allow different guideline aspects to be defined.

A mechanism was built to take each guideline and convert it to an XML format using a common algorithm. Fig. 2, demonstrates a guideline published by HHS and W3C to use at least 12-point font [1][3]. Therefore, it is required to evaluate the source code of the web items whether they use fonts that are greater than twelve points.

![Fig. 2. Sample Guideline Definition](image)

The structure in Fig. 2. has the following main components in the specified line numbers:

- **Line #1** – The root node is the guideline which has an id and a name.
- **Line #2** – A description for the guideline.
- **Line #3** – The list of organizations that recommend this guideline. If the guideline is formalized by the designer, then this should be mentioned as a “User-Defined” guideline.
- **Line #8** – The categories to which this guideline is applicable for.
- **Line #9** – The file extensions that this guideline can be applied to when checking deviations.
- **Line #17** – The set of conditions that are being assessed in this guideline. Each guideline has at least one condition. Where there are more than one condition, each condition should have a relationship (AND/OR) with its preceding condition.

- **Line #19** – Each guideline condition has an object or a set of object associated with its assessment, which is defined here.
- **Line #21** – In this case, the Font-Size attribute is being checked.
- **Line #24** – In addition to the attribute, this guideline needs to be assessed in style sheets as well. Here, the css_property attribute specifies which CSS needs to be evaluated.
- **Line #27** – Each condition has a set of evaluators. That is, the objects defined previously need to be evaluated against the values in this section.
- **Line #28** – The first evaluator checks the Font-Size to be greater than or equal to (gte) the given values based on its measure.

**Fig. 3. Sample Guideline Customization**

This includes selecting the guidelines from the initially available guidelines from standard bodies and the guidelines, which were defined by the user.

Fig. 3. shows the XML structure to define a customization. This structure is relatively simple in contrast to the XML in Fig. 2. The guidelines node has a list of guideline IDs included in the customization. For example, the guideline ID of the guideline shown in Fig. 2. is 14 according to its guideline element in Line #1. This guideline is included in the customization shown in Fig. 3.

IV. IDENTIFYING ELEMENTS IN WEB INTERFACE SOURCE

A. Identifying ASP.NET Web Controls & HTML Controls

Prior to detecting deviations in the customized guideline set, the controls and elements on the web interface will have to be structured in a manner, which is more convenient to evaluate the properties of same. Another advantage of interpreting the source of web interfaces is that distributed source can be brought to a central location from where evaluation can be performed. For instance, if a particular .aspx has a link to an external style sheet, then part of the formatting for the controls on the web page will be on an external file. By performing this conversion, it is possible to have the control set as well as
the associated CSS in the same file. This will mean that detection will have to be performed only on this single file.

Fig. 4. shows a sample XML, which is the output of mapping the controls on a web page (.aspx) and its relevant external CSS file.

```
1 <SourcePage>
2  <WebsiteName>StudentRegistration</WebsiteName>
3  <FileName>StudentRegistration.aspx</FileName>
4  <FullFilePath>C:\StudentRegistration\StudentRegistration.aspx</FullFilePath>
5  <Controls>
6    ...
7    ...
8    <control>
9      <LineNumber>9</LineNumber>
10     <Type>TextBox</Type>
11     <MappedLine>10</MappedLine>
12     <ControlType>Text</ControlType>
13     <Column>1</Column>
14     <Row>1</Row>
15     <Id>txtName</Id>
16     <Value>Some Text</Value>
17     <Attributes>
18       <Attribute>Width</Attribute>=<Value>200</Value>
19     </Attributes>
20     <CSS>
21       <cssType>internal</cssType>
22       <css><![CDATA[<asp:TextBox ID="txtName" runat="server" Width="200">Some Text</asp:TextBox>]]></css>
23      ...
24    </control>
25    ...
26  </Controls>
27 </SourcePage>
```

Following information are included in this XML structure.
- Line #1 – The root node is the <SourcePage> element.
- Line #2 to Line #4 – The name of the website, the file and the absolute path of the file.
- Line #5 – The <controls> element maintains a list of controls in the web page.
- Line #8 – Each control has a corresponding line number in the source, a control type, the value between the opening and closing tags of the control, a set of attributes (Line #17) and the details of its CSS.
- Line #39 – The <css> section of a control captures its CSS. All inline, internal and external CSS are mapped to this section and are distinguished by the <cssType> element. The CSS itself is defined under <declarations> as <property> and <value> pairs.

Now, the detection can be more accessibly performed on this single XML file. In order to extract the information from the relevant files, certain regular expressions were developed.

- The following regular expression is used to match any tag in the source.
  `(?<tag><[^%/>][^/<]*[^/>]+[^/>])` For example:
  `<asp:TextBox ID="txtName" runat="server" Width="200">Some Text</asp:TextBox>`
- The second regular expression is used to retrieve the tag name and the value within the tag from the above result.
  `(?<name>[a-z][a-z1-9]*[>]<[^/>]+[^/>]+[^/>])` For example:
  `<asp:TextBox ID="txtName" runat="server" Width="200">Some Text</asp:TextBox>`

**B. Extracting CSS of Controls**

For the purpose of extracting CSS, an algorithm was designed to associate a control with its CSS whether it be inline, internal or external.

```
1 FUNCTION get_css_for_control(source_control)
2  ...
3  SET css_file = get_css_file(source_control)
4  SET external_css = get_external_css(css_file)
5  SET control_type = get_control_type(source_control)
6  SET control_id = get_control_id(source_control)
7  SET control_class = get_control_class(source_control)
8  ...
9  ADD TO css[control_type] internal.css
10  FOR EACH css_str IN css_file
11    ADD TO cssexternal.css
12  IF control_type = NOT NULL
13    ADD TO css[control_type] external.css
14  IF control_id = NOT NULL
15    ADD TO css[control_id] external.css
16  IF control_class = NOT NULL
17    ADD TO css[control_class] external.css
18  ...
19  GET inline_css
20  IF control Has style_attribute
21    ...
22    SET css_key_value = extract_css_key_value[style_attribute]
23    ADD TO css[css_key_value] external.css
24  ...
25  GET internal and external css
26  ...
27  ...
28  ...
29  ...
30  ...
31  ...
32  ...
33  ...
34  ...
35  ...
36  ...
37  ...
38  ...
39  ...
40  RETURN css_key_value_list
```

Fig. 5. shows the algorithm that is used to extract the CSS for a given control in the provided source of the web interface. Inline CSS can be easily obtained by extracting the “style” attribute of the control.
When approaching the internal and external CSS, both have the similar syntax except that the latter is located on an external file. In order to process both types simultaneously, all external CSS files attached to the source, are read into an array data structure. To this array, the internal CSS was also added. By iterating through this array, key-value pairs can be extracted. CSS defined in internal and external can be linked to the control in 3 primary possibilities. CSS possibilities exist for the control type, control ID and the “class” or “CssClass” attribute of the relevant control. Therefore, in each CSS file, it is required to again iterate for CSS for each of the 3 possibilities.

V. DETECTING DEVIATIONS FROM GUIDELINES

The sole purpose of developing a common structure was to avoid the hardcoding of the guideline logic and have a minimal number of algorithms (ideally one) to process these guidelines. As such, after incremental development, Fig. 6 shows the single pseudo code that was designed to achieve the aforementioned outcome.

There are five main levels of iteration in the algorithm, namely, guideline, condition, guideline object and control in source and evaluators. Each guideline object has a type, based on the definition in the guideline XML as follows:

- **Object Type-01** is where the object is simply the value of a tag. This will be represented in a sample guideline XML as:

  ```xml
  <object>
    <select type="tag" value="value">title</select>
  </object>
  ```

- **Object Type-02** is where the object is simply the value of an attribute of any tag. This will be represented in a sample guideline XML as:

  ```xml
  <object>
    <select type="attr" value="value">Width</select>
  </object>
  ```

![Fig. 6. Algorithm to Detect Deviations](image-url)
- Object Type-03 is where the object is the value of an attribute of a specific tag. This will be represented in a sample guideline XML as
  
  ```xml
  <object type="tag" name="TextBox">
    <select type="attr" value="value">Width</select>
  </object>
  ```

- Object Type-04 is for CSS. This will be represented in a sample guideline XML as
  
  ```xml
  <object type="tag" name="TextBox">
    <select type="attr" css_property="font-size" value="value">style</select>
  </object>
  ```

The algorithm will return a list of violations or deviations, which can be notified to the user. Additionally, this output is written to another XML file as shown in Fig. 7. This output will be used in the fixing process along with the source.

VI. AUTOMATIC FIXING OF DEVIATIONS

A. Fixing Web Controls

Once the deviations have been detected on evaluation, fixing a particular deviation simply requires backtracking from the deviation list to the source of the web interface. Each deviation entry records the control type, control ID, the filename and line number, where the control is located, and guideline, which has been deviated.

Iteratively, each control that deviates is located and values for its attributes are set according to the complying values in the guideline’s XML specification. The similar process is carried out to automatically fix CSS issues. The relevant CSS of the control is located (once again, if it is inline, the style tag is assessed and if it is internal or external, the relevant CSS is assessed,) and the appropriate value is modified in order to maintain compliance. Consequently, the automatic fixing process is, to a certain extent, straightforward one the detection process has successfully output the deviation list.

However, it is an acceptable fact that it is not practically feasible to automatically fix all deviations. For example, for a guideline that states to use a proper page title, the environment can detect is the value of the <title> tag is empty or whether it is equal to “untitled”. In such an instance, although the deviation can be successfully detected, the deviation requires user intervention to fix it completely. Therefore, in conditions that mainly check for inequality, an automatic fix cannot be implemented.

B. Previewing Interfaces

Another unique aspect of the proposed environment is that it allows the designer to assess a preview of how their web interface will appear like should it adhere to all guidelines. The benefit of this feature is that the designer can decide whether the preview is how the expected design should look like.

The previewing is achieved by creating dummy files of the actual design files. When a web interface, which has been submitted for evaluation of guideline compliance, is selected for the preview, a duplicate of same is created, along with copies of the resources it uses including “.master” and “.css” files that it refers to. The next step is to apply the fix on these dummy files, rather than the actual files. Now the preview files will comply with the selected guidelines. In order to perform the actual preview, the actual web interface and the dummy interface can be rendered in the browser for the designer to assess. Optionally, the designer should be able to compare the source of the actual file and the preview file. For this the two files can be submitted to a differ tool to highlight the differences between the content of the files. Once the designer is satisfied with the preview, he/she can proceed with fixing the actual design files, which will make the required change permanently.

VII. RESEARCH RESULTS AND EVIDENCE

The environment discussed in this paper has been adopted in the tool named twUider (Tool for Web User Interface Design Evaluation and Resolution), which automates the detection of issues in ASP.NET web application interfaces against the guidelines. The tool is an add-in for VS 2010.

At the very beginning, the HCI guidelines which are in natural language needed to be convert into a formal specification. As a solution to this, a common XML structure was generated to store any kind of guideline.

Once the detection process completes, the UI designer can fix the detected deviations automatically or manually. Before applying the fixes, the tool will allow the option to preview as shown in Fig. 7.

![Fig. 7. Deviation Listing for Previews and Fixing](image)

The environment was tested on a sample size of twenty four. The sample consisted of web developers, web designers and undergraduates of the Information Technology and Software Engineering faculties. The tool twUider was given to the participants and allowed them to use the tool with their own ASP.NET websites.

Certain significant results of the analysis are described in this section. 54.17% of the sample agreed that the overall usability of the tool was excellent with a quantitative score of 5 out of 5. 33.33% of participants scored the usability of the tool as 4 and a 12.50% of users gave a score of 3 out of 5. More than 60.00% of the participants admitted that their user interfaces improved by applying the tool, with a score of 5 out of 5. An extremely noteworthy result of the survey was that 100.00% of the participants decided that they will recommend this tool among their fellow colleagues. 79.17% of the sample
stated that they mostly prefer the automatic support of fixing
deveiations over a manual fix.

VIII. Conclusions

At present, numerous websites fail their user experience
test as proved by Forrester’s continuously updating survey
results [14]. An environment was developed to automate the
evaluation of Microsoft ASP.NET web application interfaces
against HCI guidelines. Designs are evaluated using
guidelines established by highly recommended organizations
including W3C, IEEE, ISO and HHS.

While achieving the target, separate algorithms were
developed and used for the formalization, customization,
detection and fixing processes. All these algorithms are
according to a general and standard format, which is the most
convenient method to manipulate the guidelines, which are of
dynamic nature. Therefore, an XML file was generated while
formalizing the guidelines and a set of XML files were
bundled while customizing. After customizing, the bundle is
transmitted by the UI Specialist to his/her designers, who
should apply it in order to evaluate their web designs
according to the guidelines in the customization bundle. In
addition to detection of the deviations, the environment allows
for previewing the deviations and automatically fixing them to
improve the web interface.

A tool named twUider adopted this environment and has
followed all these algorithms to automate the evaluation of the
Microsoft ASP.NET websites. The tool, which is an add-in to
the Microsoft’s Visual Studio IDE, provides functionality to
formalize guidelines, customize, perform detections, produce
previews and initiate automatic fixes.

The significant benefit of this environment is that it
encourages the web designers to consider the user interface
issues that exist in their designs as a design time support. As a
result of continuous use of the tool, designers eventually
become aware of common design mistakes and correct
themselves to refrain from repeating such mistakes. It
provides a background for designers to learn and evolve from
their mistakes, while ensuring the improvement of the web
interfaces. Additionally, since the logic of each guideline is
not hardcoded into the algorithm, the environment allows
users to define their own guidelines as per organizational
policies and ensure that the selected customization is adhered to.
Before performing a fix, the designer can view a preview of the
interface. This is a unique feature that provides a fail-
safe measure for the designer to ensure that the automatically
fixed design is exactly what is expected by the guideline.

The main limitation of this approach is that it is able to
detect the deviations using only the available source code of
the design files. This means that the dynamic content of a
website, which change during the runtime of the site, are
overlooked. Another limitation of the proposed design is that,
when fixing, it is not pragmatically possible to fix all the
deviations. In some situations, the tool cannot provide
suggestions for the web designers and for such cases, the
deviations should be fixed by the designer manually.

However, even in such an instance, since the detection
algorithm records the line number of the deviation, the tool
can provide functionality to allow the user to navigate to the
particular line in the source by simply double-clicking on the
relevant deviation in the deviation list. Therefore, the user
only needs to be concerned about changing the relevant line in
the source according to the description of the deviation.

The ultimate goal of the research focused on ASP.NET
website support for user interfaces. However, the algorithms
explained in this paper have been presented in a general
mechanism, which allows it to be customized for other
platforms with very little or no modifications at all. As such,
farther development, this environment can be applied for
IDEs that are used to design websites in platforms other than
ASP.NET, in order to provide development support for
counterpart designers in parallel platforms. Additionally, the
guideline formalization structure can be further improvised to
be made generic and applicable for a wider scope of
guidelines. This research requires an incremental basis
development where each increment improves and refines the
structures involved.

ACKNOWLEDGMENT

We would like to express profound gratitude to Mr.
Chathuranga Manamendra, for providing us with guidance
and support in the successful realization of this research.

Personnel, who took part in the research surveys and
provided their contribution at the initial literature gathering
and by testing the tool with real world applications, are also
thanked with highest gratitude.

Last, but not least, we thank Ms. Sherihan Anver and Mr.
Azam Ahmed for extending their support in helping with the
development of the environment and the tool to adopt this
environment.

REFERENCES

[1] U.S. Department of Health Human Services, Standards and Usability
Guidelines.[Online].Available: http://webstandards.hhs.gov/search/results?utf8=%E2%9C%93q=classifications%5B5D=guidelineclassifications%5D=standardcategories%5B%5D=standardcategorys%5B%5D=commitSearch, Jan 24, 2013.Abdou Beiredkarl, Jean
Vanderdonckt1, and Monique Noirhomme-Fraiture “A Framework and
a Language for Usability Automatic Evaluation of Web Sites by Static
Analysis of HTML Source Code,” Fourth International Conference on


http://www.w3.org/TR/2012/NOTE-WCAG20-TECHS
20120103/general.html, Jan 24, 2013.

webteam/stylesheet/web_presence_usability.html, Jan 27, 2013.

Technology (CAST), National Universal Design Laboratory, Peabody,

Center (University of Toronto) and Trade Center (University of


