Towards semi-automatic size measurement of user interfaces in web applications with IFPUG SNAP

Hassan Mansoor and Mirosław Ochodek
Poznań University of Technology
Faculty of Computing
Institute of Computing Science
ul. Piotrowo 2, 60–965 Poznań, Poland
Email: Hassan_mansoor32@yahoo.com,
miroslaw.ochodek@cs.put.poznan.pl

***Abstract*—Software Non-functional Assessment Process is a non-functional size measurement method proposed by the International Function Point Users Group. It can be used to measure the size of non-functional requirements related to usability of graphical user interfaces (UI). Unfortunately, measuring such requirements seems time-consuming because it requires identifying all UI elements and graphical properties that were configured to meet these requirements.**

In this paper, we propose a semi-automated approach to measure size of non-functional related to user interfaces of web applications. The method takes as an input a set of exemplary screens of application (HTML and CSS) and rules describing the mapping between HTML elements and UI Elements. The method provides a list of UI Elements and graphical properties that were configured as an output. We preliminarily validated the proposed method using a prototype tool that we had developed.

I. Introduction

Software size measures are probably the most frequently used measures in software development projects. They are, for instance, used as independent variables in many models of effort estimation, benchmarking and pricing. They also constitute a natural normalizing factor for many indirect measures.

The International Function Point Users Group (IFPUG) has developed two complementary software measurement methods. The first one is called Function Point Analysis (FPA) [1] and is probably the most recognized functional size measurement (FSM) method [2]. The second one is a non-functional size measurement method called Software Non-functional Assessment Process (SNAP) [3]. Although, the latter one is an emerging method, its initial beta tests performed in August and early September 2012 gave promising results [4]. Both of the methods seem complementary and could be used standalone or conjointly [5].

The process of measuring a size of non-functional requirements in SNAP requires associating non-functional user requirements (NFR) with categories and sub-categories defined in the method. To perform such mapping one has to be aware of how a given requirement is going to be implemented. The measurement is performed for the atomic units called SNAP Counting Units (SCU). In most of the SNAP sub-categories, SCU is *elementary processes*, which is the smallest, self-contained and consistent units of activity that are meaningful to the user.

In this paper, we focus on the SNAP's sub-category 2.1: User Interfaces (UI). The sub-category regards "unique, user identifiable, independent graphical user interface elements added or configured on the user interface that do not change the functionality of the system but affect non-functional characteristics (such as usability, ease of learning, attractiveness, accessibility)." [3]

To measure the size of an NFR related to usability one has to identify all unique UI Elements and their properties that have to be configured to meet that requirement. Unfortunately, it could be a cumbersome task taking into account the complexity of modern user interfaces. The real problem emerges when SNAP is used for pricing software development projects. From the experience of the authors who took part in such a project, it is difficult to achieve transparency in tracking which UI Elements were configured to meet certain non-functional requirements. Therefore, in this context, the following problem seems worth of investigating:

Problem 1: Given a set of screens defined by an elementary process, provide a list of the properties of UI Elements that were configured to meet non-functional requirements.

We are going to consider Problem 1 in the context of web application development. According to Forrester Research [6] web technologies are becoming increasingly critical to organizations' web and mobile application strategies. They reported that 27% of the surveyed organizations plan to move away from native development in favor of web technology (HTML5, JavaScript, and CSS) .

In this paper, we propose a semi-automated approach to measure size of non-functional requirements associated with the SNAP sub-category 2.1. The method accepts as an input a set of exemplary screens of an application (HTML and CSS) and rules for mapping HTML elements into UI Elements

recognized by the users. It returns a list of UI Elements and properties that were configured. Based on the provided output, the user of the method can decide which of the properties should be included in the SNAP measurement. We have developed a prototype to preliminary validated the proposed approach.

The paper is organized as follows. In Section II we briefly discuss the SNAP's sub-category 2.1. We present the proposed approach in Section III. In Section IV, we present the results of a preliminary validation of the proposed approach. The paper is concluded in Section V.

II. Measurement of User Interfaces with SNAP

A SNAP Counting Unit of the sub-category 2.1 is a set of screens defined by an elementary process. A measurer needs to identify UI Elements that has been configured to meet non-functional requirements. UI Element is defined as "a unique user identifiable structural element which makes up the user interface." [3] SNAP APM provides an exemplary list of UI Elements, e.g., windows, menus, icons, mouse pointers, hyperlinks, drop-down lists. This list can be extended by custom UI Elements.

Each UI Element has a set of properties that can be configured to achieve certain "look and feel" or behavior of the element. For instance, a button can have properties like button caption, background color, width, height.

The number of SNAP Points (SP) is calculated by determining UI Type Complexity based on the number of properties added or configured. Each complexity level has a corresponding weighting factor. The number of SP is obtained by multiplying the number of unique UI Elements by the weighting factor.

The sub-category can be used conjointly with IFPUG FPA. For instance, if a user requests a function allowing to generate a report about incomes, the functional size measured with FPA would be the same no matter if the report is delivered as a plain text or as visually attractive HTML page. The SNAP sub-category 2.1 could be used to capture this difference [7].

III. Proposed Appraoch

We propose a method that allows identifying UI Elements and properties that were configured based on analysis of HTML pages and corresponding CSS files. The overview of the proposed procedure is presented in Figure 1.

The method expects the following input (1):

- a set of HTML pages generated by an application or screen designs or prototypes in the form of HTML pages;
- for each HTML page, a set of XPath expressions defining fragments of the page that are treated as screens defined by the considered elementary process (e.g., a single HTML page might contain information that is relevant to different elementary processes).
- CSS files created to meet non-functional requirements;
- (optionally) a set of custom rules mapping between HTML tags and UI Elements and a set of custom UI-properties names.

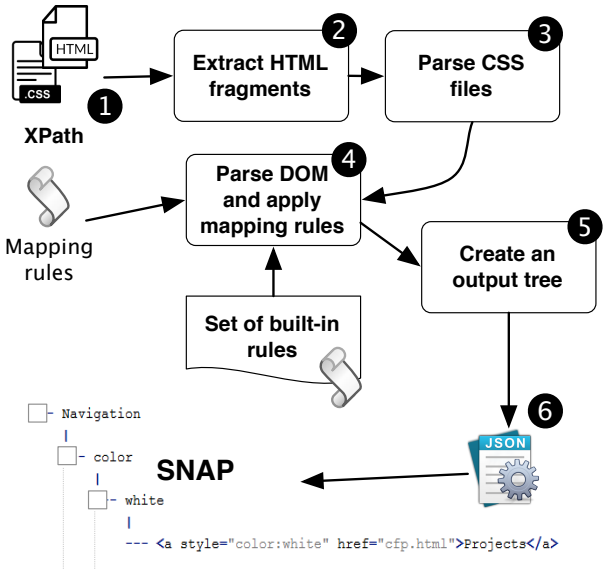

Fig. 1. The overview of the steps of the proposed method.

We extract the specified fragments of screens relevant to the considered elementary process from the HTML pages (2). In the following step (3), we process the given CSS files to extract all CSS properties that are relevant to the HTML tags in the extracted fragments of HTML pages, and then attach these properties to the HTML tags. In the nex step, we traverse through the DOM tree of the extracted HTML fragments and apply the rules of mapping between HTML tags and UI Elements (4). Finally, we collect all the graphical properties that were configured for each of the identified UI Elements (5). We return a tree-like structure as an output (6). The tree consists of four levels: UI Elements, UI properties that were configured for a given UI Element, the values of each property that were configured, and finally HTML tags that configured a given property. The output is consumed by an interactive front-end tool allowing to include/exclude certain UI Elements or UI properties from the measurement (7).

A. Mapping rules

We identified three types of mapping rules between HTML tags and UI Elements. While traversing through the DOM tree, the rules are applied for each HTML tag one by one until any of them is satisfied. Therefore, the rules should be ordered from the most specific to more general ones.

R1: *if a HTML tag is Y then UI Element is Z*

The rule allows straightforward mapping between HTML tags and UI Element. For instance, the rule "*if a HTML tag is A then UI Element is Hyperlink*" would allow treating all occurrences of tag A as UI Element Hyperlink.

R2: *if a HTML tag is Y AND has attributes* $[attr_1{=}value_1$ *AND* $\ldots$ *AND* $attr_n{=}value_n]$ *then UI Element is Z*

In many cases, the name of HTML tag is insufficient to map it to a UI Element. For instance, to identify HTML tags that

TABLE I
RESULTS OF THE PRELIMINARY VALIDATION (VALUES IN BRACKETS REPRESENT THE NUMBER OF DISTINCT VALUES SET TO A UI PROPERTY).

	yahoo.com	hitecmobile.com.sg	github.com
Elementary process	Login to the system	View product information	Signup for a GitHub account
Input HTML	Yahoo – login.html	huawei mate 8 price.html	Join GitHub – GitHub.html
Input CSS	combo.css	style.css	github-33dd01e39588....css
Screen (XPath)	//fieldset[@class='mbr-login-fieldset pure-group ']	//div[@class='product-details']	//div[@class='setup-wrapper']

Configured UI Elements	• **Form**
	  ○ margin (1)
	  ○ padding (1)
	  ○ border (1)
	• **Hyperlink**
	  ○ tabindex (1)
	  ○ text-decoration (1)
	  ○ color (1)
	  ○ font-weight (1)
	  ○ font-size (1)
	• **Drop-down list**
	  ○ margin (1)
	  ○ padding (1)
	  ○ drop-down-type (1)
	• **Label**
	  ○ for (1)
	  ○ display (1)
	  ○ margin (1)
	  ○ cursor (1)
	  ○ font (1)
	  ○ color (1)
	• **Button**
	  ○ tabindex (1)
		• **Hyperlink**
		  ○ color (1)
		• **Image**
		  ○ width (1)
		  ○ max-width (1)
		  ○ height (1)
		• **Drop-down list**
		  ○ height (1)
		  ○ float (1)
		  ○ clear (1)
		  ○ margin-bottom (1)
		  ○ width (1)
		  ○ background (3)
		  ○ border (1)
		  ○ display (1)
		• **Button**
		  ○ title (1)
		• **Label**
		  ○ for (4)
		• **Text box**
		  ○ maxlength (1)
		  ○ cols (1)
		  ○ rows (1)
			• **Header**
			  ○ margin-top (1)
			  ○ margin-bottom (2)
			  ○ font-size (2)
			  ○ font-weight (1)
			  ○ line-height (1)
			  ○ letter-spacing (1)
			  ○ padding (1)
			  ○ overflow (1)
			  ○ border-bottom (1)
			• **Drop-down list**
			  ○ padding (3)
			  ○ margin (3)
			  ○ font-size (1)
			  ○ list-style (1)
			  ○ margin-top (2)
			  ○ border-top (1)
			• **Label**
			  ○ autocapitalize (1)
			  ○ autofocus (1)
			  ○ for (3)
			• **Text box**
			  ○ autocapitalize (1)
			  ○ autofocus (1)
			  ○ size (1)
			  ○ width(1)
			• **Password box**
			  ○ size (1)
			  ○ width (1)

# Configured UI Elements	5	6	5
# UI Properties	18	17	24
SNAP Points (SP)	20	24	20

represent UI Element Button we would have to define at least two rules:

- R1-type rule: "*if a HTML tag is BUTTON then UI Element is Button*"
- R2-type rule: "*if a HTML tag is INPUT AND has attributes [type=submit] then UI Element is Button*"

We might also extend the set of rules with custom rules. For instance, let assume that we implemented a custom JavaScript code and defined CSS class called "button" that makes a DIV tag appearing and working like a Button. To treat such a DIV as Button, we could formulate the following rule: "*if a HTML tag is DIV AND has attributes [class=button] then UI Element is Button*."

R3: *if a HTML tag is Y AND is inside a UI Element X OR HTML tag W then UI Element is Z*

In some cases, a single UI Element is implemented as a subtree of DOM. For instance, a UI Element Drop-down could be implemented as a tag SELECT and its children OPTION tags. We assume that by configuring any property of the SELECT tag or OPTION tags it encloses we configure the same UI Element.

IV. PRELIMINARY VALIDATION

To preliminary validate the proposed method we developed a prototype tool and selected three web pages from yahoo.com, hitecmobile.sg and github.com as an object of study. We decided to select simple pages of the considered sites to be able to verify the results manually. We downloaded the pages and corresponding CSS files (we included only a base CSS files, treating rest of them as externally provided and excluded from the measurement). We also specified XPath expressions to extract fragments of pages that seem important from the perspective of the considered elementary processes. The results are summarized in Table I.

We observed that the prototype tool was capable of processing the selected pages and calculating the number of UI Elements and properties that were configured. During the validation, we made the following observations regarding the current version of the method and prototype tool:

- Sensitivity to HTML and CSS errors — the proposed approach is sensitive to errors in CSS and HTML. The positive side effect is that the tool might help to improve the quality of the HTML code of the application being measured.
- JavaScript modifying HTML — modern rich internet applications often rely on JavaScript to generate and modify HTML. In extreme cases, downloaded sources of web pages might not contain any HTML tags corresponding to UI Elements. To overcome this issue we recommend using web browser plugins, such as Firebug (http://getfirebug.com) that allow investigating and downloading source of HTML page as it is at runtime. On the other hand, we found a drop-down UI Element in the source of Yahoo's sing-in HTML page that was not visible to us at the moment we had visited the page (it would be in a different state of the page).
- Processing output HTML — we process HTML pages that are generated by an application. These are snapshots that capture screens in certain states. Consequently, the HTML page we are processing might not reveal all the properties that were configured. For instance, if we consider a form that uses some visual effects to show validation errors, we might not identify these properties if an empty form was downloaded and processed. To overcome this issue, we should provide many instances of the same page in different states to obtain a union of configured properties.
- Importance of custom mapping rules — the set of mapping rules is an essential component of the tool. For instance, our built-in set of mappings rules included HTML tags such as INPUT type="hidden" which did not seem important from the perspective of the processed UIs. Consequently, it caused identification of few UI Elements that were false positively classified as configured. It is also important to define custom rules to be able to distinguish different UI Elements that are represented by the same HTML tag. For example, without a set of custom mapping rules, we would not be able to distinguish between images and icons. However, in some cases, distinguishing such elements might be impossible (image and icon are distinguished based on their content and not based on HTML / CSS properties).

V. Conclusions

In this paper, we presented an approach to measure non-functional size related to user interfaces of web applications with the use of the IFPUG SNAP method.

We developed a prototype tool that processes HTML pages and CSS files to identify UI Elements and UI properties that were configured. The proposed approach can be customized by providing rules mapping between HTML tags and UI Elements.

We preliminary validated the prototype tool by processing three web pages. Although the prototype tool seems sensitive to errors in HTML code, we were able to obtain valid results.

As a future work, we plan to work on extending the set of built-in rules and providing a possibility of comparing different version of HTML pages to identify changes in the configuration of UI Elements resulted from enhancement projects. Moreover, we would like to validate the proposed approach and the prototype tool on a larger dataset.

References

[1] IFPUG, "Function Point Counting Practices Manual, Release 4.3.1, ISBN 978-0-9753783-4-2," 2010.

[2] ISO/IEC, *Information technology — Software measurement — Functional size measurement — Part 1: Definition of concepts*, 2007.

[3] IFPUG, "Software Non-functional Assessment Process (SNAP) — Assessment Practices Manual, Release 2.3, ISBN 978-0-9903007-3-1," 2015.

[4] C. Tichenor, "A New Metric to Complement Function Points," *CrossTalk—The Journal of Defense Software Engineering*, vol. 26, no. 4, pp. 21–26, July/August 2013.

[5] M. Ochodek and B. Ozgok, "Functional and Non-functional Size Measurement with IFPUG FPA and SNAP—Case Study," in *Software Engineering in Intelligent Systems*. Springer, 2015, pp. 19–33.

[6] Forrester Consulting, "The Rise Of Web Technology: Addressing Development Complexity And Productivity Demands," 2015.

[7] A. Sahoo, "Using SNAP for GUI Creation and Enhancement—An Experience," *MetricsViews*, vol. 8, no. 1, pp. 6–7, February 2014.