JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 38, 375-395 (2022)
DOI: 10.6688/JISE.202203_38(2).0006

Identification and Validation of Web Themes: A DOM-Structure Matching Approach*

BAO-AN NGUYEN[1,2], HSI-MIN CHEN[1], CHYI-REN DOW[1], YAN-TING CHEN[1] AND HOANG-THANH DUONG[1]
[1]*Department of Information Engineering and Computer Science*
Feng Chia University
Taichung, 40724 Taiwan
[2]*Department of Information Technology*
Tra Vinh University
Tra Vinh, 940000 Vietnam
E-mail: annb@tvu.edu.vn; {hmchen;crdow; m0705965; dhthanh}@mail.fcu.edu.tw

In modern large-scale websites, front-end web development is constructed by the teamwork of developers. To be a professional website, all web pages should strictly follow predefined theme specifications in order to maintain the common look and feel of the web layout. However, this requires a lot of testing effort to ensure that web pages developed by individual developers are compliant with the web theme. This research proposes a system to enable the automatic identification of web theme templates from existing web pages and to examine the compliance of new web pages based on the identified web theme template. By using our system, violations of web themes can be detected as soon as developers commit changes to the version control repository. In this way, the apparent consistency of websites is not only carefully maintained, but the effort for manual validation can also be drastically reduced.

Keywords: web theme template, web programming, software testing, web quality, software quality

1. INTRODUCTION

At a first glance, web development tasks can be categorized into front-end and back-end tasks [1]. While back-end development refers to server-side operations providing functional communication between the database and the web browser, front-end development refers to user experience via visual functionalities. Front-end developers are engaged in designing web user interfaces and ensuring user interaction at the client-side using front-end languages such as HTML (HyperText Markup Language), CSS (Cascading Style Sheets), and JavaScript [2]. Recently, many front-end frameworks have been proposed to reduce developers' workload and boosting user satisfaction as well as web applications' quality such as BootStrap, React, Angular, Vue. On the contrary, there are few studies in web testing to support web developers to validate their designs.

Received November 16, 2020; revised December 27, 2020 & February 21, 2021; accepted March 1, 2021.
Communicated by Chu-Ti Lin.
* This research was supported in part by Ministry of Science and Technology, Taiwan, under Grants No. 109-2221-E-035 -055 -MY3.

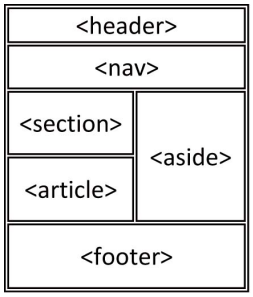


Fig. 1. Basic template of web theme.

Since it is related to users' look and feel, the web layout should be always consistent among all element web pages of a website. A web theme template is a pre-defined layout scheme that facilitates developers to ensure the consistency of the website interface. Fig. 1 shows a commonly used web theme template in several websites, according to the semantic elements of HTML 5 [3]. The identification signs of the web site are presented in the header block and the footer block. The aside block is placed in a fixed position for showing dynamic announcement or advertisement contents. The theme template is always embedded to provide the same layout but different contents for all web pages of the website.

In large-scale websites, it is hard to maintain this consistency, due to the variations in design habits of various front-end developers who contribute to the project [4]. In the early days of web projects developed from scratch, the web theme template and/or design specifications were not carefully considered. Since at that time, the website is only on small scale and easy to maintain with a few developers, or because the customers required a short time to release [5]. Nevertheless, as the website scale grows incrementally, web templates become an essential concern that not only guarantees a professional look and feel presentations for users but also ensures that developers adhere to a consistent layout style during their works. According to the survey by [6], the volume of web templates on the World Wide Web was about nearly 50% and contains around 30% hyperlinks and visible terms. In such a context, how to identify a theme template from a series of web pages already developed becomes an essential issue that needs to be addressed. Moreover, after committed by a developer, how to confirm that a web page fully adheres to the pre-defined web theme templates? Web development teams have to spend a long time verifying the compliance between the newly added web page and the templates if no support tools have been used. This raises a further motivation for us, where the new web pages have to be validated automatically immediately after they have been committed.

Based on the above aspects, we developed a strategy to identify theme templates from existing web pages and to validate the compliance of newly introduced web pages towards the detected theme. Through our approach, web developers can easily define a template from a selected web page without designing it from scratch and can get immediate feedback after the validation scheme has been applied. The proposed automated system for identification and validation of web theme templates can dramatically reduce web development costs with consistent look-and-feel themes.

In addition, our proposed approach shows the potential to be applied in web content

mining tasks such as classification, clustering of web documents and even in search engines. Using our tool, the web template blocks are quickly identified and detached from the main content of web pages, and thus, the execution time of information retrieval tasks can be reduced significantly.

The rest of the paper is organized as follows. Section 2 describes related works. The proposed approach is presented in Section 3. The experimental results are carried out in Section 4 and finally, we conclude our work in Section 5.

2. RELATED WORK

The high impact of web applications generates high demand for web testing techniques to ensure the quality of web projects [7]. In web testing, the input models can be classified into one of such categories: navigation models, control or data flow models, document object models (DOMs), and others [8]. While the two former types of input models are used to analyze the dynamic operation of websites, DOM models are a representation of the rendered layout of web pages, and hence, they are used as the main input of GUI related test cases. In such test cases, due to the tree structure of DOM models, tree-based algorithms are used to implement testing procedures.

The problems of tree matching and tree similarity measuring are popularly used in HTML/XML comparison, natural language processing, compiler design, computer vision, molecular biology, and many other fields [9–12]. In web testing, the problem of determining template nodes from DOM trees can be reduced to the problem of finding common subtrees, in which a common subtree represents the hierarchical structure of the extracted template. Two common approaches for determining common subtree are using the suffix tree [13] and using the concept of common longest subsequence (LCS). The maximum common subtree problem is reducible to the LCS problem with input are ordered trees [14]. Tree LCS can be computed using various *Tree Edit Distance-based* algorithms [15]. The authors of [16] measured the structural similarity of HTML pages by using *Tree Edit Distance* and proposed a clustering algorithm for web pages. For the same purpose, Gupta and Chhabra use *Cityblock Distance* to compare the similarity between web pages [17]. Yin *et al*. used *Text Edit Distance* between tags sequences extracted from the DOM trees as the main similarity measurement for template extraction and clustering web pages [18]. The authors of [19] used *Tree Edit Distance* in a restricted top-down mapping algorithm for web template detection using the DOM tree. High time complexity $O(N^3)$ of the *Tree Edit Distanc*e is one limitation of this conventional similarity measurement [20]. Besides, using *replace, delete* and *insert* functions in *Tree Edit Distance* based matching algorithms may lead to wrong level templates when level information of nodes is not maintained when edit functions are executed.

Recently, to address the drawbacks of conventional methods, researchers proposed new DOM tree similarity measurements for web template extraction. The article [21] proposed a new similarity metric based on DOM's path with shingles technique according to the ratio of the intersection and union of the shingles paths between two DOMs; this metric allows either web pages clustering or structural mining on web documents. The study [22] adopted *minimum description length* to manage the unknown number of web templates to improve the performance of template identifying and extraction algorithms.

Joshi *et al*. [23] proposed the *bag of tree-paths model* for measuring structural similarity in web documents which only capture parent/child relationships to reduce the computational complexity of structural information comparison tasks. The study [24] leveraged dynamic programming to produce the maximum matching between two DOM trees by recursively comparing every possible pair of subtrees at each corresponding level and capture the maximum matched. Similarly, in the web template extraction plugin *TexMe* [25] implemented its comparison function using the graph theory formalism called *Equal Top-Down Mapping (ETDM)* to inspect the relation between DOM trees. The study [26] used the combination of *hyperlink distance* and *DOM distance* to identify templates in web pages belonging to a website. To enable a fast and no-restricted comparison of web pages, the authors of [27] proposed the method *Similarity-based Tree Matching* which can match real-life documents in practical time (less than a second for matching the DOM of Youtube). The aforementioned studies introduced multiple efficient similarity metrics for web page matching. However, those approaches may not suitable for web template comparison since they were designed for information extraction or eliminating the template, not for template validation. Some additional works should be done to make these approaches suitable for web template validation.

In dynamic web, the appearance of web pages can vary in different web browsers. Webdiff [28] analyzed both the structure of the DOM structure and the visual representation of the layout of the web page to detect rendering errors in cross-browser tests. Since user actions via AJAX can change the content of DOM trees. The study [29] inspected the variants of DOM trees to detect Ajax-specific faults after user events. These matching methods test variants of DOM trees of only one page, thus they are not suitable to be merged into template identification tasks, which always take place on multiple web pages.

Besides those web documents analysis methods based on DOM trees, layout analysis based on rendered contents of web documents were also studied. Roudaki *et al*. [30] extracted structured data from web documents by leverage a converting technique called visual engineering for web patterns in which two-dimension graph grammar of web page is transformed into one-dimension of string induction. Moran *et al*. [31] adopted a computer vision based approach to identify errors in mobile application GUIs by leveraging the design diagram. However, visual-based methods might not applicable when dealing with complicated nested structures of layout elements.

In this paper, we add into the literature two web template identification methods and their corresponding web template validation procedures. The first method, called *Rigid Template*, concerns original structural information of the DOM trees with both top-down and bottom-up approaches. The second method, called *Soft Template*, performs matching on the flattened version of the DOM trees. We enhance the second method by using a post-processing step to avoid wrong level matching on the flattened trees. Our proposed approach can be used as a useful testing tool for developers from the very first stage of the web development process. In the following sections, we present our novel methods for identifying web theme templates from HTML documents as well as methods for validating web theme compliance of newly developed web pages. The analysis algorithms not only analyze the content of nodes but also consider the structure of DOM trees and the relative position of nodes in the trees.

Table 1. Notations for web template identification algorithms.

Notation	Description
T	A set of n DOM trees $T = \{T^1, T^2, \ldots, T^n\}$.
T^P	The DOM Tree selected as the base tree.
T^C	The currently considered DOM Tree
$root^P$	Root node of the DOM tree P.
n_i^P	A i^{th} child node of root node $root^P$ *(level-1)*.
$n_{i,j}^P$	A j^{th} child node of the node n_i^P *(level-2)*.
	With index of nodes denoted by i, j. Indexing rules are as follows:
	if $index > 0$: indexing starts from left.
	if $index < 0$: indexing starts from right.
N^P	The set of nodes $\{n_1^P, n_2^P, \ldots\}$ which are children of $root^P$.
N_i^P	The set of nodes $\{n_{i,1}^P, n_{i,2}^P, \ldots\}$ which are children of node n_i^P.
$PreOrder(T)$	A sequence of tags generated by traversing the tree T in *node-left-right* order.
$SubTree(n_i^P)$	A subtree of the tree P formed by node n_i^C and its descendants.
$LCS(n_i^P, n_j^C)$	Longest common subsequent betwen the i^{th} subtree of DOM tree P and j^{th} subtree of DOM tree C.
$V_{i,j}(P,C)$	Length of Longest Common Subsequence (LCS) between the i^{th} subtree of DOM tree P and j^{th} subtree of DOM tree C.
	Example: Length of LCS between $\{div1, li2, a3\}$ and $\{div1, li2\}$ is 2

3. PROPOSED APPROACH

This section introduces our algorithms for identifying and validating a website's theme templates. DOM trees are adopted as input data models for the analysis algorithms. Concerning the identification of web templates, the algorithms try to find common nodes between the input DOM trees, starting from a base tree given by a specific HTML document. In order to cover possible matching cases between DOM trees, this study focuses on two types of web templates called Rigid Template and Soft Template. Whereas the Rigid Template inspects both the order and the position of nodes at each tree level in its matching procedure, the Soft Template takes into account only the order of common nodes at each level of input trees. We introduce the necessary notations for the proposed algorithms in Table 1. The definition of two types of templates, identification algorithms and validation algorithms for web templates will be presented in the next subsection.

3.1 Theme Template Definitions

Given a DOM tree T^P as a base web page and a set of DOM trees T which contains all inspected pages, we define the web theme template as follows:

3.1.1 Rigid template

A rigid template is defined as a structure of web layout in which each GUI element appears in a fixed position among all web pages. Formally, a web template is a structure of common nodes that appear at fixed positions with the same orders at each level of all DOM trees. Since the template nodes can be found with a high possibility from the

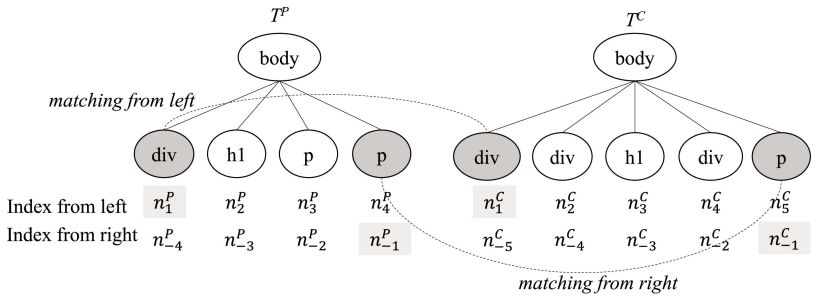


Fig. 2. Example of the rigid template.

top and the bottom of web pages, it is feasible for searching for template nodes in both two fashions, top-down and bottom-up. In tree structures, top-down searching is done by traversing nodes of a level from left to right, and vice versa for bottom-up searching. Formally, given a two DOM trees T^P and T^C , in which T^P is the base tree and T^C is the currently considered tree, we can identify rigid templates by following procedures:

Matching from the left: Two lists of nodes at each level of the two DOM trees are traversed concurrently from left to right. If n_i^P of tree T^P and n_i^C of tree T^C are the same element, they are template nodes. As shown in Fig. 2, the first nodes from the left of both two trees are div nodes with the index $i = 1$. Hence, n_1^P and n_1^C should be selected as a template node.

Matching from the right: Two lists of nodes at each level of the two DOM trees are traversed concurrently from right to left (hence, the nodes have negative indexes). If n_{-i}^P of tree T^P and n_{-i}^C of tree T^C are the same elements, they are template nodes. Take Fig. 2 as an example, the rightmost nodes two trees, n_{-1}^P and n_{-1}^C, have the same value p with index $i = -1$, so the p element of this position is detected as a template node.

Generally, template UI elements positioned at the bottom of a UI block are usually linked continuously, hence, any break in the searching from the right indicates that the template block is interrupted at that position. As shown in Fig. 2, the second node from the right of two trees T^P and T^C are p and div, respectively. Due to this difference, the following nodes will not be counted as template nodes and the search step will be terminated before it reaches n_{-3}^P and n_{-3}^C. A general example of a rigid template is illustrated in Fig. 2.

3.1.2 Soft template

Rigid templates are useful for managing structural information of web themes because the orders and indexes of a template node are fixed. To support a flexible identification scheme, we suggest another type of template, called soft template, in which the orders of the template nodes must be maintained, but the corresponding indexes can vary between DOM trees. In other words, the soft template is an extension for identifying template structures of such websites that have been developed in the style of freedom, with additional content nodes being added between template nodes.

As shown in Fig. 3, the three nodes $\{div, h1, p\}$ appearing in three pairs $\{n_1^P, n_1^C\}$,

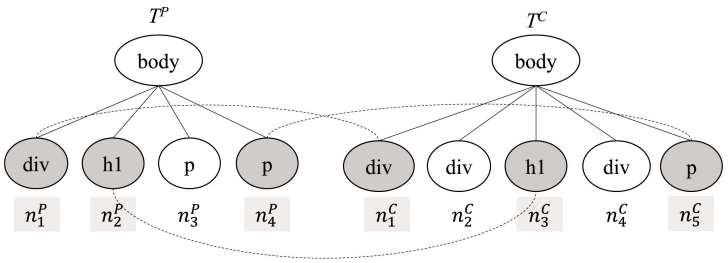


Fig. 3. Example of the soft template.

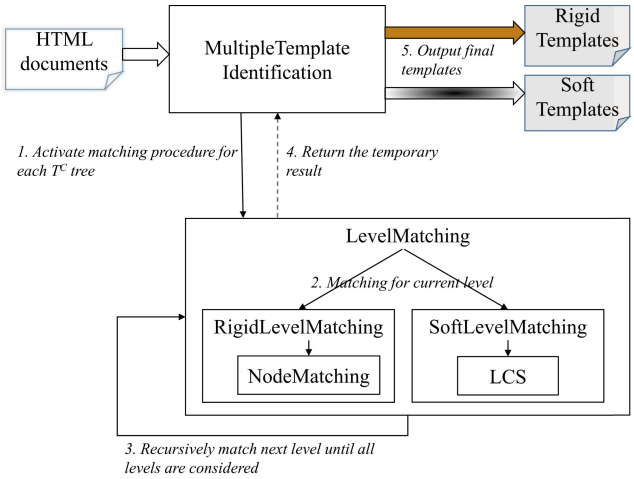


Fig. 4. The workflow of web template identification.

$\{n_2^P, n_3^C\}$, and $\{n_4^P, n_5^C\}$, respectively, in the two DOM trees can be identified as template nodes in the soft template, since they share common elements and orders but vary in their indexes. To identify such cases of pattern matching, we leverage LCS (Longest Common Subsequence) [32] technique on the input trees, in which the tree structures are flattened to be string vectors and inputted to the matching procedure. All template identification algorithms will be discussed in the next section.

3.2 Template Identification Algorithms

Given a set $T = T^1, T^2, \ldots, T^n$ of DOM trees and a base tree T^P, the web template is identified in T by the matching procedures between T^P and each $T^C \in T$. Both rigid template and soft template should be outputted by the algorithms.

The Selenium tool [33] was used to render dynamic web pages and compare resulting DOM trees of HTML documents to determine which UI elements belong to the template. Fig. 4 shows the flowchart of the template identification process. Let's start with the algorithm *MultipleTemplateIdentification* with the base tree T^P and a set T of DOM trees, the identification process of web template can be explained as follows:

Algorithm 1: MultipleTemplateIdentification

Input: A set of n DOM trees $T = \{T^1, \ldots, T^n\}$
Output: Web templates
begin
  $loss \leftarrow \{\}$;
  $T^P \leftarrow$ base tree for matching;
  //Comparing all web pages in T to identify the template
  for *each* $T^C \in T$ **do**
    if *templateType==”rigid”* **then**
      $T^P \leftarrow LevelMatching(T^P, T^C)$;
    else
      //templateType==”soft”
      $sameNodes \leftarrow LevelMatching(T^P, T^C)$;
      $loss \leftarrow loss \cup \{T^P - sameNodes\}$;
    end
  end
  //get the template by deleting all non-template node
  $webTemplate \leftarrow \{T^P – loss\}$;
end

Algorithm 2: LevelMatching

Input: n_i^P, n_i^C
Output: Templates structure of current level
begin
  $tN^P \leftarrow N_i^P$;
  $tN^C \leftarrow N_i^C$;
  if *templateType=”rigid”* **then**
    $n_i^P \leftarrow RigidLevelMatching(tN^P, tN^C)$;
  else
    //templateType=”soft”
    $n_i^P \leftarrow SoftLevelMatching(tN^P, tN^C)$;
  end
  //recursively get the child nodes of the next levels
  for *each* $n_{i,j}^P \in N_i^P$ **do**
    if $N_{i,j}^P \neq \emptyset$ **then**
      $tempN_{ij}^P \leftarrow SubTree\left(n_{i,j}^P\right)$;
      $tempN_{ij}^C \leftarrow SubTree\left(n_{i,j}^C\right)$;
      $LevelMatching(tempN_{ij}^P, tempN_{ij}^C)$;
    end
  end
end

Step 1: Select an input $T^C \in T$ for inspecting. The algorithm *LevelCompare* is called to perform matching on the level-1 nodes of two DOM trees T^P and T^C.

Step 2: Perform both two types of template matching for rigid template and soft template for the current level. *NodeMatching* function is used for matching two nodes in the rigid template; LCS function is used for matching two subtrees in the soft template.

Step 3: After the current level is matched, the algorithm *LevelMatching* is recursively performed on all pairs of corresponding subtrees of current levels until all levels of T^C are matched.

Step 4: Temporary result for the current DOM tree T^C is returned to the overall set of templates. The algorithm *MultipleTemplateIdentification* repeatedly performs identification steps (beginning from step 1) for the next DOM tree in T until all DOM trees in T are considered.

Step 5: Two kinds of web templates are outputted as the identified web templates.

The algorithm 1, *MultipleTemplateIdentification*, presents the workflow of the web template identification procedure. The rigid template can easily be obtained after a series of operations *LevelMatching* have been performed recursively. On the contrary, with the soft template, there are non-template nodes obtained after after each execution of *LevelMatching*, we need to run a post-processing step to eliminate these nodes. Finally, we receive two template files, one for the rigid template and one for the soft template, as results of the web template identification algorithm.

At each level of the DOM tree, we leverage the *LevelMatching* method to discover template nodes at that level. Just like commonly used approaches in tree matching, we use a recursive strategy to do the comparison at all levels of the DOM trees. In this function, if the selected method is *rigid*, the *RigidLevelMatching* method is activated to the matching steps for two directions of approaching, from left and right, as in Fig. 2. Otherwise, the *SoftLevelMatching* with LCS method is called to determine matching nodes between corresponding levels of the two trees based on the idea of the LCS algorithm. *LevelMatching* method is described in detail in Algorithm 2.

3.2.1 Rigid template detection

As discussed in the definition, the rigid template requires two corresponding nodes in two trees should reside in the same position (denoted by index numbers), hence, the two input trees are scanned concurrently to extract all matched pairs of nodes. Since template nodes are usually located at the top *(eg. header, banner, navigation bar, etc.)* and the bottom *(eg. footer)* of web pages, the matching algorithm should perform in two directions, top-down and bottom-up. First, the algorithm scans the level from the right to pick up template nodes from the bottom. The scanning steps from the right are terminated right after a pair of nodes cannot be matched. The last matching point is stored as end_R as a reference for further steps, as shown in Fig. 5 (a).

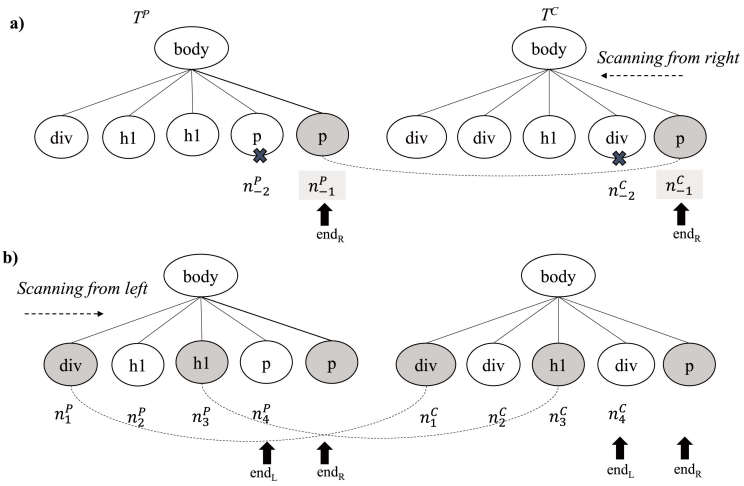


Fig. 5. Rigid template identification from right (a) and from left (b).

Algorithm 3: RigidLevelMatching

Input: N^P, N^C
Output: *templateNodes*: List of rigid template nodes of current level
begin
  $K \leftarrow ||N^P||$;
  $L \leftarrow ||N^C||$;
  $j = 0$;
  $templateNodes \leftarrow \emptyset$;
  //matching from right
  while $(j < K$ *and* $j < L)$ **do**
    if $NodeMaching(n^P_{i,K-j}, n^C_{i,L-j})$ **then**
      $templateNodes.add(n^P_{i,K-j})$;
      if $K < L$ **then**
        $end_R \leftarrow K - j$;
      else
        $end_R \leftarrow L - j$;
      end
    else
      break;
    end
    $j++$;
  end
  //matching from left
  $j \leftarrow 1$;
  for $j = 1$ *to* end_R **do**
    if $NodeMaching(n^P_{i,j}, n^C_{i,j})$ **then**
      $templateNodes.add(n^P_{i,j})$;
    end
  end
  return *templateNodes*;
end

Then the matching process is restarted from the leftmost node to detect template nodes from the top. The matching procedure is repeated until the node end_R is reached, as shown in Fig. 5 (b). All matching nodes output of the template nodes of the current level. As the algorithm only needs to traverse the DOM trees twice, one for top-down and one for bottom-up, in the worse case, the time complexity and space complexity of the rigid template matching algorithm is $O(n)$ and $O(\log(n))$, respectively. Steps of *RigidLevelMatching* are described in Algorithm 3.

3.2.2 Soft template detection

Soft templates can be identified using matching methods based on the concept of LCS. Given two DOM trees, a soft template is resulted by finding LCSs between them. To find out the LCSs, first, the DOM trees should be flattened to be sequences of strings *(HMTL tags)* by traversing in *Node-Left-Right* order (called *PreOrder* in short); then the matching procedures are run on these sequences. With each level, we use the array $V(P,C)$ to store the length of LCSs obtained after each comparison, in which $V_{i,j}(P,C)$ is the length of LCS between the i^{th} subtree of DOM tree P and j^{th} subtree of DOM tree Q. As shown in Fig. 6, when matching the two $[body]$ nodes of two trees, the comparison between n_1^P and n_1^P gave us $V_{1,1}(P,C) = 1$, and the comparison between n_2^P and n_1^P gave us $V_{2,1}(P,C) = 5$. The maximum value in $V(P,Q)$ lets us determine the length and the position of longest sequence of matched nodes at the current level.

In general, we can easily identify soft templates by matching all possible pairs of nodes of the current level using LCSs. However, after the DOM trees have been flattened by *PreOrder* traversal, the hierarchical information of nodes is lost and error cases arouse. In practice, we found and addressed two error cases as follows:

Error case 1: When two different tree structures produce the same tag sequences, the two trees can be mismatched. For example, the tag sequences $[div, li, a]$ generated by both two DOM trees in Fig. 7 lead to a mismatch. To address this wrong matching, we add level numbers into the tags when traversing the tree. For instance, after level numbers

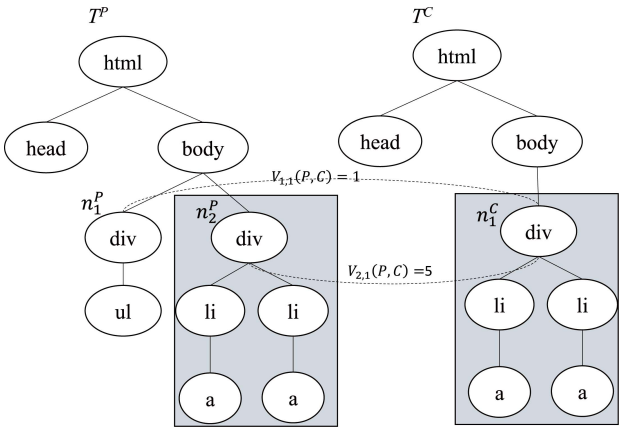


Fig. 6. Soft template identified by using LCS.

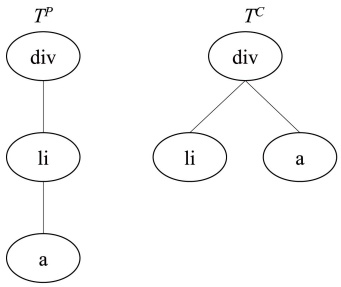


Fig. 7. Soft template error matching when different tree structures generate the same tag sequences.

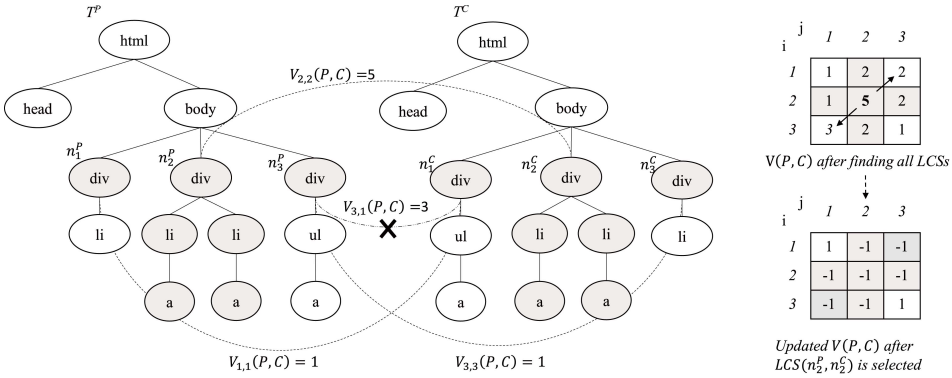


Fig. 8. Soft template wrong order matching: after n_2^P and n_2^C were matched, matching between n_3^P and n_1^C lead to wrong order matching.

were added, the tag sequence of the tree T^P becomes $[div1, li2, a3]$ and that of the tree T^C becomes $[div1, li2, a2]$, so that the error case of mismatching on $a2$ and $a3$ is avoided.

Error case 2: Since soft template requires the order of template nodes must be maintained, but the LCS based matching procedure may encounter errors when the matched nodes have wrong orders in the corresponding DOM trees. As shown in Fig. 8, after matching all possible pairs of descendants of $[body]$ node in two trees, we have a total nine LCSs whose lengths are stored in the matrix $V(P,C)$. To identify the soft template, we iteratively select the maximum number in the matrix and add corresponding nodes (denoted by the index of the selected number) to the template, until all nodes were examined. In the example, we first select the longest matched sequence given by $LCS(n_2^P, n_2^C)$ with $V_{2,2}(P,C) = 5$ into template nodes. If we then select the second-longest sequence given by $LCS(n_3^P, n_1^C)$ with length $V_{3,1}(P,C) = 3$, the wrong order matching would occur because the order of newly added template nodes is conflict to the previous ones.

To avoid this, after selecting an LCS as template nodes, we must eliminate such pairs that can lead to wrong order matching by utilizing the matrix $V(P,C)$. As shown in Fig. 8, after $LCS(n_2^P, n_2^C)$ was selected, certainly we must eliminate all pairs containing n_2^P and n_2^P. In $V(P,C)$ we mark all elements on 2^{th} row and 2^{th} column of $V(P,C)$ to be -1. In addition, such pairs (n_i^P, n_j^C) which have indexes *($i < 2$ and $j > 2$)* or *($i > 2$ and $j < 2$)* also

need to be marked as -1 in the matrix $V(P,C)$, since they lead to wrong order matching. The resulted $V(P,C)$ matrix after this post-processing step is shown in Fig. 8.

The algorithm $SoftLevelMatching$ describes steps of the soft template identification procedure. First, all LCSs of possible pairs of nodes in the two DOM trees are discovered and stored in the 2-D array *LCSs*. Their lengths are also stored in the 2-D array V, respectively. Then, the current longest LCS will be added to the list of $templateNodes$ based on the current maximum value of $V(P,C)$. After an LCS is selected, the post-processing step is executed to remove such LCSs that can generate wrong order templates. Concurrently, corresponding values of those LCSs in $V(P,C)$ are also set to -1. The selection step is repeated until all elements of V equals -1. The resulting $templateNodes$ after all LCSs examined is the soft template of the current level. The time and space complexities of

Algorithm 4: SoftLevelMatching

Input: N^P, N^C
Output: $templateNodes$: List of soft template nodes of current level
begin
  $LCSs \leftarrow [\,][\,]$; //2-D arrays for LCSs
  $V \leftarrow [\,][\,]$; //2-D arrays for length of LCSs
  $templateNodes \leftarrow \emptyset$
  //Determining all possible LCSs
  foreach $n_i^P \in N^P$ **do**
    $seq_i \leftarrow PreOrder(n_i^P)$;
    foreach $n_j^C \in N^C$ **do**
      $seq_j \leftarrow PreOrder(n_j^C)$;
      $lcs_{ij} \leftarrow LCS(seq_i, seq_j)$;
      $LCSs[i,j] \leftarrow lcs_{ij}$;
      $V[i,j] \leftarrow ||lcs_{ij}||$;
    end
  end
  //Select soft template using longest LCS
  while $max(V) > 0$ **do**
    $[k,l] \leftarrow argmax(V)$;
    $templateNodes.add(LSC[k,l])$;
    // Post processing to avoid wrong order template
    for $i = 1$ *to* $||N^P||$ **do**
      for $j = 1$ *to* $||N^C||$ **do**
        if $((i <= k)$ *and* $(j >= l))$ *or* $((i >= k)$ *and* $(j <= l))$ **then**
          $V[i,j] = -1$;
        end
      end
    end
  end
  return $templateNodes$;
end

SoftLevelMatching algorithm are $O(mn)$, where m and n are the numbers of nodes in the two DOM trees. This estimation is inferred from those of the LCS problem when it is implemented by using dynamic programming.

3.3 Template Validation

After web templates were identified, the *TemplateValidation* algorithm was proposed to inspect if newly added web pages conform to the detected templates. With the same workflow and procedures as template identification, the validation steps also try matching the DOM tree of the input HTML document with templates extracted and output a set of missing nodes called *loss*. According to two kinds of web templates, which are rigid template and soft template, two kinds of*loss templates* can be outputted by the algorithm, as shown in Fig. 9. Processing steps of web template validation are described in Algorithm 5.

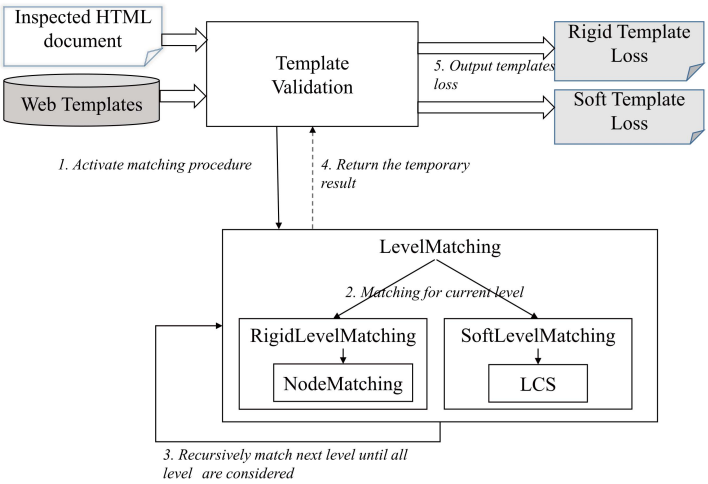


Fig. 9. The workflow of Web template validation.

Algorithm 5: Template validation.

Input: A DOM trees T^V, web template T^P
Output: template loss containing missing nodes
begin
  $lossNodes \leftarrow \{\}$;
  $matchedNodes \leftarrow LevelMatching(T^P, T^V)$;
  //get the missing nodes
  $lossNodes \leftarrow \{T^P – matchedNodes\}$;
end

4. EXPERIMENTAL RESULTS

In this section, we present the results of experiments conducted by using proposed identification and validation methods on two sets of input web pages obtained from StackOverflow [34] and W3Schools [3]. To set up the baseline for the comparison, we first identified the web templates from input web pages manually. We use the Chrome DevTools [35] to manually inspect and identify elements that belong to the templates, such as header, footer, banner, sidebars, navigation bars, *etc*. Then we did the same procedure using the tool Difflib [36]. The rigid template and soft template identification algorithms were executed in the same manner. With the soft template, we investigate both two tree matching methods using LCS and Suffix Tree [13]. The number of nodes in the DOM trees of resulting template pages is counted and compared. Overall, we have five result-sets for each experiment as discussed in the following sub-sections.

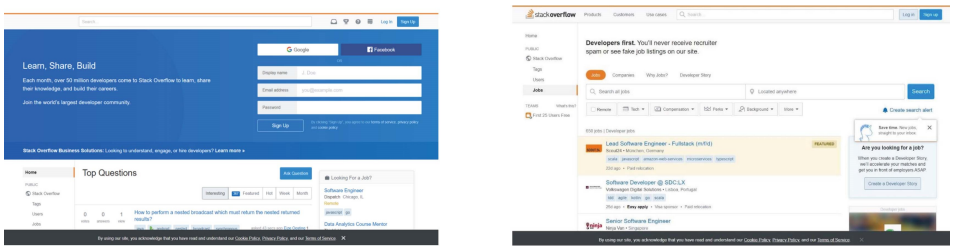
4.1 Experiment 1: StackOverflow Template

With input web pages from StackOverflow, we select two pages which are a typical question-answer page in Fig. 10 (a) and a member registration page in Fig. 10 (b). The number of extracted template nodes by different identification methods is shown in Table 2.

Table 2. The experimental results with StackOverflow website.

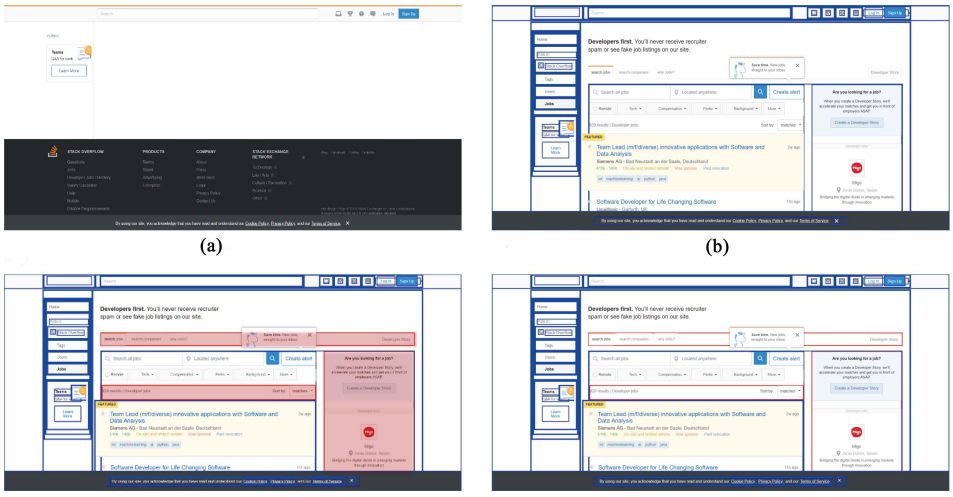
Method	Number of nodes identified in the template
Manual identification	427
Rigid template	420
Soft template with LCS	431
Soft template with Suffix Tree	427
Difflib tool	346

The numbers of template nodes extracted by proposed methods are comparable with that of manual identification. In Fig. 11, the blue boxes are GUI elements that are identified as web template nodes, and the red boxes mark elements that were missed in the rigid template but were well identified in the soft template. Since the registration form in Fig. 10 (a) appears at the top and shift the index of the following nodes, some blocks were



(a) (b)

Fig. 10. Input web pages from StackOverflow.



(a) (b) (c) (d)

Fig. 11. Identified web templates from StackOverflow: (a) Manual template; (b) Rigid template; (c) Soft template with LCS; (d) Soft template with SuffixTree.

missing the identified rigid template in Fig. 11 (b) in comparison to the soft templates shown in Figs. 11 (c) and (d). The soft template with LCS was quite different from the soft template with Suffix Tree. Since the matching procedure with Suffix Tree tries to find longest matching subtrees, the tag sequences should be continuous. If there are extra nodes inserted between template nodes, the matching methods with Suffix Tree cannot be matched, like the central text in Fig. 11 (d). Overall, the soft template with LCS produced outperformed other methods in template identification results, due to its flexibility in the matching method.

4.2 Experiment 2: W3Schools Template

In the second experiment, we conducted the same procedure as in Experiment 1 with input pages from the W3Schools website (Fig. 13). The number of discovered template nodes in Table 3 shows the advancement of the soft template with LCS in comparison with other methods. The template nodes in the central block also better recognized by the soft template with LCS than the soft template with Prefix Tree, as shown in Fig. 13 (c).

Table 3. The experimental results with W3Schools website.

Method	Number of nodes identified in the template
Manual identification	656
Rigid template	656
Soft template with LCS	664
Soft template with Suffix Tree	656
Difflib tool	591

(a) (b)

Fig. 12. Input web pages from W3Schools.

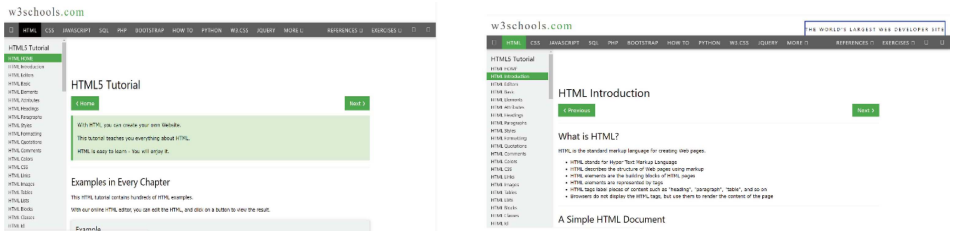
(a) (b)

(c) (d)

Fig. 13. Identified web templates from W3Schools: (a) Manual template; (b) Rigid template; (c) Soft template with LCS; (d) Soft template with SuffixTree.

Finally, we tested the validation function of our approach with the input web pages from W3School in Fig. 14 (a) accordingly to the template shown in Fig. 14 (b). It can be seen that the blue mark on Fig. 14 (b) is the template nodes missed in Fig. 14 (a).

The above experimental results demonstrate the performance of our method for identification and validation of web templates. Our method surpasses the Difflib [36] tool in all test cases. The rigid template identification method has a comparable performance to the manual identification. With the flexible mechanism in DOM tree matching, the soft method with LCS-based matching provides the most acceptable results, which were significantly better than themselves with the Suffix Tree-based matching. With the fruitful results obtained, our approaches are certainly suitable for integration into web GUI testing systems.



(a) (b)

Fig. 14. Validation web templates from W3Schools: a) The input web page; b) Result of template validation.

5. CONCLUSION

In this paper, we come up with an automatic system for identifying and verifying web theme templates using DOM tree analysis and matching algorithms. Two types of web templates and their identification algorithms were presented. The rigid template is suitable for websites with fixed layouts, where the positions of GUI elements are often pinned. The soft template can be applied in web development in a more flexible manner, where GUI elements can be occasionally added to web pages under the same ordered framework. Our proposed system can drastically reduce the labor costs of web developers in verifying the conformity of newly developed web pages with pre-defined web themes. With our system, developers no longer have to manually check the web template to ensure the consistency of the website. Besides, our template identification algorithms can be applied in web mining or information retrieval tasks, as it can easily extract web content by eliminating the web templates from the DOM trees. Nevertheless, our approach is limited in the condition that a base tree T^P is required as input for template identification. Currently, the web templates cannot identify without a base tree. In future work, we will study the fully automatic identification of web templates directly from a set of input pages. In addition, we will integrate continuous integration techniques into our system to facilitate automatic validation of website quality when new changes are committed.

REFERENCES

1. M. Stal, "Web services: beyond component-based computing," *Communications of the ACM*, Vol. 45, 2002, pp. 71-76.
2. S. Souders, "High performance web sites," *Queue*, Vol. 6, 2008, pp. 30-37.
3. W3Schools, "Html5 semantic elements," https://www.w3schools.com/html/html5_semantic_elements.asp, 2020.
4. A. A. Ozok and G. Salvendy, "How consistent is your web design?" *Behaviour & Information Technology*, Vol. 20, 2001, pp. 433-447.
5. M. J. Escalona, M. Urbieta, G. Rossi, J. A. Garcia-Garcia, and E. R. Luna, "Detecting web requirements conflicts and inconsistencies under a model-based perspective," *Journal of Systems and Software*, Vol. 86, 2013, pp. 3024-3038.

6. D. Gibson, K. Punera, and A. Tomkins, "The volume and evolution of web page templates," in *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, 2005, pp. 830-839.
7. A. Arora and M. Sinha, "Web application testing: A review on techniques, tools and state of art," *International Journal of Scientific & Engineering Research*, Vol. 3, 2012, p. 1.
8. S. Doğan, A. Betin-Can, and V. Garousi, "Web application testing: A systematic literature review," *Journal of Systems and Software*, Vol. 91, 2014, pp. 174-201.
9. P. Bille, "A survey on tree edit distance and related problems," *Theoretical Computer Science*, Vol. 337, 2005, pp. 217-239.
10. P. Klein, S. Tirthapura, D. Sharvit, and B. Kimia, "A tree-edit-distance algorithm for comparing simple, closed shapes," in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2000, pp. 696-704.
11. S. S. Chawathe *et al.*, "Comparing hierarchical data in external memory," in *Proceedings of the 25th Very Large Database Conference*, Vol. 99, 1999, pp. 90-101.
12. K.-C. Tai, "The tree-to-tree correction problem," *Journal of the ACM*, Vol. 26, 1979, pp. 422-433.
13. R. Grossi, "On finding common subtrees," *Theoretical Computer Science*, Vol. 108, 1993, pp. 345-356.
14. A. Lozano and G. Valiente, "On the maximum common embedded subtree problem for ordered trees," *String Algorithmics*, 2004, pp. 155-170.
15. S. Mozes, D. Tsur, O. Weimann, and M. Ziv-Ukelson, "Fast algorithms for computing tree lcs," *Theoretical Computer Science*, Vol. 410, 2009, pp. 4303-4314.
16. T. Gowda and C. A. Mattmann, "Clustering web pages based on structure and style similarity," in *Proceedings of IEEE 17th International Conference on Information Reuse and Integration*, 2016, pp. 175-180.
17. G. Gupta and I. Chhabra, "Optimized template detection and extraction algorithm for web scraping of dynamic web pages," *Global Journal of Pure and Applied Mathematics*, Vol. 13, 2017, pp. 973-1768.
18. G.-S. Yin, G.-D. Guo, and J.-J. Sun, "A template-based method for theme information extraction from web pages," in *Proceedings of IEEE International Conference on Computer Application and System Modeling*, Vol. 3, 2010, p. V3-721.
19. K. Vieira, A. S. Da Silva, N. Pinto, E. S. De Moura, J. M. Cavalcanti, and J. Freire, "A fast and robust method for web page template detection and removal," in *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, 2006, pp. 258-267.
20. K. Bringmann, P. Gawrychowski, S. Mozes, and O. Weimann, "Tree edit distance cannot be computed in strongly subcubic time (unless apsp can)," *ACM Transactions on Algorithms*, Vol. 16, 2020, No. 48.
21. D. Buttler, "A short survey of document structure similarity algorithms," in *Proceedings of International Conference on Internet Computing*, Vol. 1, 2004, pp. 3-9.
22. S. Pushpa and D. Kanagalatchumy, "A study on template extraction," in *Proceedings of IEEE International Conference on Information Communication and Embedded Systems*, 2013, pp. 109-115.
23. S. Joshi, N. Agrawal, R. Krishnapuram, and S. Negi, "A bag of paths model for measuring structural similarity in web documents," in *Proceedings of the 9th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, pp. 577-582.
24. H. Wang and Y. Zhang, "Web data extraction based on simple tree matching," in *Proceedings of IEEE WASE International Conference on Information Engineering*, Vol. 2, 2010, pp. 15-18.
25. J. Alarte, D. Insa, J. Silva, and S. Tamarit, "Temex: The web template extractor," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 155-158.
26. J. Alarte, D. Insa, J. Silva, and S. Tamarit, "Site-level web template extraction based on dom analysis," in *Perspectives of System Informatics*, 2016, pp. 36-49.
27. S. Brisset, R. Rouvoy, R. Pawlak, and L. Seinturier, "Sftm: Fast comparison of web documents using similarity-based flexible tree matching," arXiv Preprint, 2020, arXiv:2004.12821.
28. S. R. Choudhary, H. Versee, and A. Orso, "Webdiff: Automated identification of cross-browser issues in web applications," in *Proceedings of IEEE International Conference on Software Maintenance*, 2010, pp. 1-10.
29. A. Mesbah, A. Van Deursen, and D. Roest, "Invariant-based automatic testing of modern web applications," *IEEE Transactions on Software Engineering*, Vol. 38, 2011, pp. 35-53.
30. A. Roudaki, J. Kong, and K. Zhang, "Specification and discovery of web patterns: a graph grammar approach," *Information Sciences*, Vol. 328, 2016, pp. 528-545.
31. K. Moran, B. Li, C. Bernal-Cárdenas, D. Jelf, and D. Poshyvanyk, "Automated reporting of gui design violations for mobile apps," in *Proceedings of the 40th ACM International Conference on Software Engineering*, 2018, pp. 165-175.
32. V. Chvatal and D. Sankoff, "Longest common subsequences of two random sequences," *Journal of Applied Probability*, Vol. 12, 1975, pp. 306-315.
33. Selenium, https://www.seleniumhq.org, 2020.
34. StackOverflow, https://www.stackoverflow.com, 2020.
35. Chrome DevTools, https://developer.chrome.com, 2020.
36. Difflib, https://docs.python.org/3/library/difflib.html, 2020.

Bao-An Nguyen received B.S. in Hanoi University of Science and Technology in 2005 and the M.S. and the Ph.D. degrees in Information Engineering and Computer Science from Feng Chia University, Taiwan, in 2011 and 2021, respectively. He is currently a faculty of Department of Information Technology, Tra Vinh University, Vietnam. His research interests include data mining, software engineering and education technology.

Hsi-Min Chen received the B.S. and Ph.D. degrees in computer science and information engineering from National Central University, Taiwan, in 2000 and 2010, respectively. He is currently an Associate Professor with the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include software engineering, programming education, object-oriented technology, service computing, and distributed computing.

Chyi-Ren Dow was born in 1962. He received the B.S. and M.S. degrees in information engineering from National Chiao Tung University, Taiwan, in 1984 and 1988, respectively, and the M.S. and Ph.D. degrees in computer science from the University of Pittsburgh, PA, in 1992 and 1994, respectively. He is currently a Distinguished Professor with the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include mobile computing, ad-hoc wireless networks, agent techniques, fault tolerance, and learning technology.

Yan-Ting Chen received his M.S. degree in the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include software engineering, software quality assurance and web technologies.

Hoang-Thanh Duong received his B.S. degree in the Department of Information Technology, Tra Vinh University, Vietnam in 2018. He is currently a master student in the the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include software engineering, web technologies, IoT and mobile applications.