A Component-Based Reverse Engineering Approach: Decomposing Web Pages to Facilitate Maintenance and Reusability

Thiam K. Chiew, Karen V. Renaud

Abstract — Web page creation has become even easier with the emergence of many different authoring tools. Unfortunately, these authoring tools do not guarantee that the design is effective or that the resulting web pages are usable and perform well. With more Web pages being designed and implemented by novices, Web page maintenance has become more difficult. Furthermore, pressure to deploy Web applications within a short time has led to pages being published without first undergoing thorough testing. Consequently, such pages are more likely to be modified and refined frequently after deployment. Moreover, the heterogeneous operational platform and environment of Web-based applications, as well as diversified user base of the applications, inevitably poses unpredicted new requirements after deployment. Therefore, the maintenance of Web pages needs to be carried out more frequently and rapidly as compared to traditional software maintenance. This research proposes a reverse engineering framework that can be used to facilitate and ease maintenance of Web pages. Existing Web pages are decomposed into different types of components to facilitate systematic examination. A proof-of-concept prototype tool was developed to assess the feasibility of the proposal. Initial evaluation has delivered encouraging results.

Index Terms — Maintenance, Reusability, Reverse engineering, Web page

1 Introduction

Software maintenance is the process of evolving a system after it has been delivered, in order to correct coding, design, or specification errors, or to accommodate new requirements, by means of modifying existing system components or adding new components to the system. However, maintenance is normally perceived as a second-class activity to development [1].

Hence maintenance of Web applications does not receive enough attention. Pressure to deploy Web applications within a short time has led to applications that still have actual development tasks outstanding, ostensibly to be carried out during the operation and maintenance phase after deployment [2].

Even with well planned and engineered Web applications, the heterogeneous operational platform and environment of the application, as well as the diversified user base, inevitably poses unprecedented new requirements after deployment. The requirements may come as immediate feedback soon after the application is launched. This is due to the fact that Web applications themselves can serve as a communication channel between the developer and end-users. Once an application has been launched, end-users can provide feedback about the application via emails, online feedback forms, or discussion forums. The feedback may well include suggestions for improvement or request additional functionality. Therefore, Web application maintenance needs to be carried out more frequently and rapidly as compared to traditional software maintenance. In fact, the distinction between development and maintenance of Web applications has, over time, become blurred.

Apart from maintaining Web sites in response to the users' feedback and to correct errors, Web sites also need to be maintained to keep the content up-to-date and accurate, by adding or updating database tables, for example. Technology improvement is another factor that drives the maintenance efforts. It is reasonable to conclude that maintenance of Web sites is driven and motivated by diverse factors which make it inevitable rather than optional.

This paper proposes a new approach to Web maintenance. Most existing Web

- *T.K. Chiew is with the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. E-mail: tkchiew@um.edu.my.*
- *K.V. Renaud is with the Department of Computing Science, University of Glasgow, Scotland. E-mail: karen@dcs.gla.ac.uk.*

maintenance approaches focus on maintaining of Web sites as a whole, here we argue for an approach that maintains Web pages as individual entities using a component-based reverse engineering approach. The approach views Web pages as being composed of distinct components that can be isolated and extracted from the Web pages to be scrutinised for maintenance purposes.

The main types of components that make up a Web page were identified by studying existing Web technologies and reviewing related literature. A prototype tool was then built to demonstrate the feasibility of the proposed maintenance approach.

Section 2 of this paper reviews existing approaches to Web maintenance. Section 3 outlines the rationale for the component-based approach. Section 4 presents the five identified Web page components. Section 5 describes a prototype tool that was built to facilitate examination of database queries in Web pages to support maintenance activities. This is followed by a section that presents and discusses the results obtained from using the prototype tool for examining database queries in five Web pages. Section 7 concludes.

2 EXISTING APPROACHES TO WEB MAINTENANCE

There are some existing tools based on established software engineering methods that help to ensure the correctness of the site structure. For example, Sciascio *et al.* [3] used a symbolic model checking technique, a formal method, to verify structure models of Web sites against their specifications expressed in a logical language [3].

Reverse software engineering is a widely used technique to help building maintenance tools for Web sites. Martin and Martin [4] describe a tool that parses HTML documents and Web server log files to construct and visualise the structure of a Web site as a graph. Ricca and Tonella [5] developed a tool called *ReWeb* which analyses a Web site and models its structure as a graph, as well as tracking the site's evolution to discover changes that may degrade its original structure. Chung and Lee [6] also reverse-engineered Web sites using the Unified Process and visual models with Unified Modelling Language (UML) to understand their navigation schemes and physical structures.

It is obvious that most existing tools adopt a macro approach and focus on the maintenance of Web sites as a whole, rather than focusing on maintenance of individual Web pages as entities. Maintenance of Web pages requires a more focused approach, something more intelligent than mere editorial assistance such as that which is provided by tools such as HTML parsers which check the correctness of the code, or human-computer interaction tools that examine the pages' graphical design, validity of hyperlinks in the pages, and other related usability aspects.

A component-based maintenance approach can support maintenance of Web pages, but also promotes reusability of components that make up a Web page. In this paper, a component is defined as: *"any Web page element that has a clear role and can be isolated, and can be replaced with a different component with equivalent functionality"*. This definition is adapted from [7: 311].

3 WHY A COMPONENT-BASED APPROACH?

Modifying and updating Web pages, as well as changing their graphical designs, are classic Web maintenance tasks. Capilla and Duenas [8] proposed a re-engineering approach that involves Web file comparison, code inspection, and manual examination of Web files' graphical views to identify common and variable aspects of existing Web sites. The common and variable aspects are then used to construct a product-line that reveals the site's architecture. This assists in the development of new Web sites and the maintenance of existing Web sites. Even though the study focused on the maintenance of Web *sites* more than Web *pages*, it proposes the decomposition of Web pages into different components. The components include different parts of HTML code including tables, JavaScript functions, database code including PHP and Active Server Pages (ASP) functions, and other modules including XML files.

Viewing a Web page as a *group of components* has the advantage of modularising the page for ease of understanding. It is common to use different technologies in the creation of a single Web page. These could include markup languages (HTML, XML, etc), client-side scripting languages (JavaScript, VBScript, etc), and server-side programming languages that support dynamic creation of the page (ASP, PHP, etc).

Each of these technologies serves different purposes. HTML describes the structure and layout of the page; client-side scripting language provides additional functionality to the page such as producing

pop-up or drop-down menus, and performs form validation before form data are sent to the server for processing; server-side programming languages are useful typically for reinforcing business logics and supporting database functions and keeping page content current.

As a result, the page comprises a mixture of different codes. Code of one type may be embedded within code of another type, making them difficult to comprehend. For a Web page comprising different codes, one of the ways to enhance comprehension is to isolate the code chunks into different components based on the technologies and/or functionality.

4 The Five Web Page Components

By studying existing Web technologies and reviewing related literature, five main types of Web page components had been identified:

1. Page structure and layout, described using HTML and cascading style sheets (CSS).
2. Page content, including text, images, multimedia elements, and hyperlinks.
3. Additional functionality provided by scripting languages, e.g. JavaScript, or other Web development techniques, e.g. Ajax.
4. Business and application logics reinforced by languages such as PHP, JSP, and ASP.
5. Database functionality supported by languages such as SQL, PHP, JSP, and ASP.

This categorisation corresponds to Padmanabharao's [9] view of a Web site as consisting of two main parts: "the front end that the user interacts with, and everything else that includes business components and data repositories that power the Web site". Page structure and layout, content, and additional functionality correspond to the first part, while business and application logics, and database functions correspond to the second part. The modularised component-based view of a Web page is illustrated in Fig. 1 using UML-like notations.

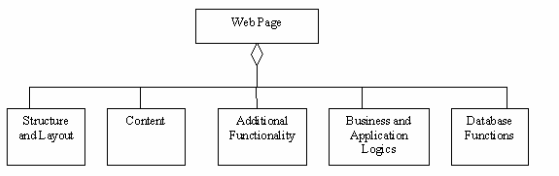

Fig. 1. Modularised Component-based View of a Web Page.

Each of the components delineates different aspects of a Web page. For example, a typical HTML Web form for user registration can be described by specifying its features from the perspective of the five components, as shown in Table 1.

By viewing a Web site as being composed of some or all of these components, it is possible to reverse-engineer an existing Web page. Each component can then be scrutinised individually for maintenance purposes or to identify possibilities for reusability.

Based on the study carried out by Linos et al. [10], "there is a worldwide tendency for Web sites to accumulate large numbers of stale (more than 6 months) documents (HTML, plain text, and image files) over long periods of time". This is a challenge to the maintenance of Web sites and the Web pages they host. Reverse engineering, supported by the component-based approach outlined above, provides a consistent yet comprehensive way of understanding existing but inadequately documented Web pages. As a result, Web pages with similar or overlapping content can be consolidated; outdated Web pages can be removed; orphan pages and broken links can be identified; user interfaces can be made consistent; business and application logic can be enhanced; undocumented database table structure can be unveiled; and old, but not obsolete, data can be retrieved. The steps are useful to deliver a better user experience when browsing the Web site.

TABLE 1

Specifying an HTML Web Form for User Registration as Components

Component	Specification
Structure and Layout	Top to bottom flow layout.
Content	Logo of the company (image) and instructions for filling in the form (text).
Additional Functionality	JavaScript to validate data entered to the form before submitting the form.
Business and Application Logics	Ensure no replication of user name.
Database Functions	Store user data to the database or populate particular fields of the form for the user.

5 A Prototype Tool

In order to assess the feasibility of the proposed component-based reverse engineering approach, a proof-of-concept prototype tool was developed using Java. The tool takes a Web page as input and then

extracts and analyses database functions within the page to determine whether the functions are formulated to behave efficiently. Similar tools can be built to support examination of the other page components if sets of best-practice guidelines are identified for the implementation of the components. The database component was chosen instead of others because of its well-defined purpose (management of data), and limited yet straightforward activities (select, insert, update, delete, etc), which allow a higher level of automation to be achieved more easily.

Another point worth mentioning is that application of such a tool depends on the language or technology that the Web page uses. The reason for this is that automated code examination can only be done on the basis of encoding the syntax and rules of the implementation code, which is language and technology dependent. The tool discussed here was developed for the examination of Web pages written in ColdFusion Markup Language (CFML)[1], a scripting language similar to HTML that uses tags. The use of tags in CFML, in turn, supports the automation of the tool.

The examination of Web pages focuses on the SQL (Structured Query Language) queries performed using SELECT statements, each of which retrieve data from databases. The purpose of the examination is to determine whether all the data retrieved are indeed used, in other words, that the retrieval of the data is necessary.

6 RESULTS AND DISCUSSION

Five CFML Web pages were taken from a Web site supporting the needs of a community church and examined using the prototype tool. After that, the results of examination were compared with results from a manual inspection to determine whether the tool was able to identify the following aspects:

1. Number of query statements, together with the query names and data sources.
2. Data fields retrieved.
3. For each data field retrieved, its number of occurrences and the location of each occurrence in the page.

Table 2 shows the overall comparison between inspections done manually and using the tool. LOC refers to “lines of code” while “queries and data fields’ occurrences” refers to the occurrences of query names with select … from SQL statements and data fields retrieved by those queries.

The comparison demonstrates that the maintenance tool was indeed able to identify the number of queries in the page examined and to determine how many among them contain `select … from` SQL statements. However, the tool is not yet able to track the occurrences of query names and data fields accurately for two reasons.

First of all, the tool only differentiates between comments and other code in the page. The page content and CFML code are treated equally. Therefore, if the page contains words or characters that are identical to the query names or data fields identified, those occurrences in the page content will be counted as well. As a result, the tool will show more occurrences of query names and data fields than manual inspection. This is exhibited in the inspections of Page A. The problem can be solved by extracting only CFML code from the page before the inspection is done.

TABLE 2
COMPARISON BETWEEN AUTOMATED AND MANUAL INSPECTIONS OF CFML WEB PAGES

Page	LOC	Attributes	Automated	Manual
Page A	146	No. of Query	12	12
		No. of Query with select … from	3	3
		Queries and Data Fields’ Occurrences	12	7
Page B	161	No. of Query	11	11
		No. of Query with select … from	4	4
		Queries and Data Fields’ Occurrences	8	8
Page C	224	No. of Query	3	3
		No. of Query with select … from	3	3
		Queries and Data Fields’ Occurrences	8	12
Page D	500	No. of Query	7	7
		No. of Query with select … from	7	7
		Queries and Data Fields’ Occurrences	19	37
Page E	571	No. of Query	17	17
		No. of Query with select … from	15	15
		Queries and Data Fields’ Occurrences	66	83

Secondly, the tool is not aware of the scope of the data fields. When a data field is

[1] http://www.adobe.com/products/coldfusion/

referenced without being qualified by the query name, i.e. the data field is preceded by the query name followed by a period (.), the tool does not recognise its occurrence. For example, when the data fields are referenced within a <CFLOOP> statement. The consequence is exhibited in the inspections of Pages C, D, and E. To solve the problem, a more complicated syntactic analyser is required to make the tool aware of the context of the CFML code and the scope of the variables used.

Regardless of the limitations, the tool still demonstrates a way of supporting the Web page maintenance tasks, particularly in inspecting database queries involved in creating the page. For example, inspection on Page E using the tool has discovered that the creation of the page involves 17 database queries, which will probably have a negative impact on the page's performance in terms of response time. Based on the report generated from the tool, a more in-depth manual inspection was carried out to examine the page. It was found that most data retrieved were either not used or retrieved more data than was used.

With the support of the tool, weaknesses in the design of database queries in Page E, which has 571 LOC, were discovered easily and quickly. Therefore the tool is a useful aid in facilitating the Web site maintenance process.

7 CONCLUSION

Maintenance is an essential part of the software life cycle. It is also challenging if the software to be maintained is not properly documented, which makes it difficult to understand. This is especially the case for Web sites because of the ease of changing individual pages to correct faults in the system, without the need for any documentary evidence. Understanding software is a task that needs to be carried out in the early stages of the maintenance process to ensure correctness and effectiveness of maintenance. This paper described a component-based approach that could help maintenance staff to understand Web pages in a systematic way. The approach decomposes a Web page into five possible components, which include structure and layout, content, additional functionality, application and business logics, and database functions. This approach also suggests the possibility of being able to reverse engineer existing Web pages into different components.

Tools can be built to examine and analyse each of these different components. A proof-of-concept prototype tool that examines database queries in ColdFusion Web pages was developed and presented in this paper. The tool checks whether the embedded queries are formulated and used effectively and efficiently. The tool was able to produce a result that is similar to that achieved by the more time-consuming manual inspection, but clearly takes less time and effort, especially when complex and lengthy pages need to be maintained.

REFERENCES

[1] I. Sommerville, *Software Engineering*, 8th Ed, Essex: Pearson Education, 2007.

[2] A. Ebner, B. Proll, and H. Werthner, Operation and "Maintenance of Web Applications," *Web Engineering*, G. Kappel, B. Proll, S. Reich & W. Retschitzegger, eds., Heidelberg: John Wiley & Sons, pp. 155-170, 2006.

[3] E. Sciascio, D. Donini, F. M. Mongiello, and G. Piscitelli, "Web Applications Design and Maintenance Using Symbolic Model Checking," *Proc. 7th European Conference on Software Maintenance*, pp. 63-72, 2003.

[4] J. Martin, J., and L. Martin, L., "Web Site Maintenance with Software-Engineering Tools," *Proc. 3rd International Workshop on Web Site Evolution*, 2001.

[5] F. Ricca, and P. Tonella, "Web Site Analysis: Structure and Evolution," *Proc. International Conference on Software Maintenance*, 2000.

[6] S. Chung, and Y.S. Lee, "Reverse Software Engineering with UML for Web Site Maintenance," *Proc. 1st International Conference on Web Information Systems*, 2000.

[7] T. C. Lethbridge, and R. Laganiere, *Object-Oriented Software Engineering: Practical Software Development using UML and Java*, 2nd Ed, Berkshire: McGraw Hill, 2004.

[8] Capilla, and J. C. Duenas, "Light-weight Product-Lines for Evolution and Maintenance of Web Sites," Proc. 7th European Conference on Software Maintenance and Reengineering, 2003.

[9] S. Padmanabharao, "Improving User Experience through Improved Web Design and Database Performance," *High-Performance Web Databases: Design, Development, and Deployment,* S. Purba, eds., Boca Raton: Auerbach Publications, pp. 623-630, 2001.

[10] P. K. Linos, E. T. Ososanya, and H. Natarajan, "Maintenance Support for Web Sites: A Case Study," *Proc. 3rd International Workshop on Web Site Evolution*, 2001.

Thiam K. Chiew Obtained both his bachelor and masters degrees in computer science from the University of Malaya in 1998 and 2000, respectively. He received his PhD degree in computing science from the University of Glasgow in 2009. He is now a lecturer at the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. He is also the review process coordinator and secretary for Malaysian Journal of Computer Science.

Karen V. Renaud obtained her Honours degree from the University of Pretoria and her Masters degree from the University of South Africa. She received her PhD degree from the University of Glasgow in 2000. She is currently a Senior lecturer in the Computing Science department of the University of Glasgow.