(An Association Unifying the Sciences, Engineering, and Applied Research)
ISSN (Online): 2279-0071
International Journal of Software and Web Sciences (IJSWS)
www.iasir.net

Reverse Engineering: Methodologies for Web Applications

Tapan Nayak[1], Dr. M. Hanumanthappa[2]
[1,2]Department of Computer Science and Applications,
Bangalore University, Bangalore, INDIA
tapan.nayak39@gmail.com, hanu6572@bub.ernet.in

Abstract: *The Reverse Engineering of Web Applications is a complex problem, due to the variety of languages and technologies that are contemporary used to realize them. Indeed, the benefits that can be obtained are remarkable: the presence of documentation at different abstraction levels will help the execution of maintenance interventions, migration and reengineering processes, reducing their costs and risks and improving their effectiveness. Moreover, the assessment of the maintainability factor of a Web Application is an important support to decision making processes. Business processes are often implemented by mean of software systems which expose them to the user as an externally accessible Web application. This paper describes a methodologies for recovering business processes by dynamic analysis of the Web applications which ex-pose them.*

Keywords: Reverse Engineering, Web Application, Business Processes, Dynamic Analysis

I. Introduction

Reverse engineering is the process of discovering the technological principles of a device, object, or system through analysis of its structure, function, and operation. It often involves taking something (a mechanical device, electronic component, computer program, or biological, chemical, or organic matter) apart and analyzing its workings in detail to be used in maintenance, or to try to make a new device or program that does the same thing without using or simply duplicating (without understanding) the original.
Reverse engineering [1] as "the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction". Their pioneering work on this subject area suggests that the main reason that reverse engineering is frequently adopted is because it enables engineers to recover inadequate or non-existent information about a system by conducting a "backward" analysis of the existing legacy system in order to gain an overall perspective of its functionality. This in turn, facilitates the production of accurate documentation describing the original implementation, design and requirements stages of the lifecycle. For further information on "standard" reverse engineering the authors would recommend [2].

Although now somewhat dated, present a good review of reverse engineering activities. A more recent, high level, overview of the aims and objectives of traditional reverse engineering and the associated issues can be found [3]. An interesting discussion of early work on traditional reverse engineering approaches can be found [4]. Many WWW reverse engineering methodologies/tools are built on established (general) reverse engineering approaches. For example [5] describe a (general) reverse engineering paradigm called GMT (Goals, Models, Tools) which was later used to support the WARE [6]. WA reverse engineering methodology [7] [8] describe how they adapted the Portable Bookshelf Environment (PBS) for "stand alone" reverse engineering to derive the architecture of WAs. Boldyreff and Kewish propose a system that exploits traditional reverse engineering ideas to extract duplicate content and style from WWW pages [9] with the aim of restructuring the WWW pages so as to improve their maintainability.

Even though the concept of reverse engineering has become widely accepted, this subject area in general has also brought about some controversy in terms of legal concerns regarding disassembling of a third parties' software application, especially in the context of copyright and Intellectual property rights [10]. However, exclusive rights for organizational applications developed by third-party developers are, in most cases, held by the organizations themselves and they therefore act as the owner of the software. In such cases, [11] determined that any manipulative activities performed on the system are not regarded as illegal acts as long as ownership rights and regulations are established at an early stage.

A. Motivation

Reasons for reverse engineering

- **Interfacing**. RE can be used when a system is required to interface to another system and how both systems would negotiate is to be established. Such requirements typically exist for interoperability.
- **Military or commercial espionage**. Learning about an enemy's or competitor's latest research by stealing or capturing a prototype and dismantling it. It may result in development of similar product.
- **Improve documentation shortcomings** Reverse engineering can be done when documentation of a system for its design, production, operation or maintenance have shortcomings and original designers are not available to improve it. RE of software can provide the most current documentation necessary for understanding the most current state of a software system
- **Obsolescence** Integrated circuits often seem to have been designed on obsolete, proprietary systems, which means that the only way to incorporate the functionality into new technology is to reverse-engineer the existing chip and then re-design it.
- **Software Modernization**. RE is generally needed in order to understand the 'as is' state of existing or legacy software in order to properly estimate the effort required to migrate system knowledge into a 'to be' state. Much of this may be driven by changing functional, compliance or security requirements.
- **Product Security Analysis.**. To examine how a product works, what are specifications of its components, estimate costs and identify potential patent infringement. Acquiring sensitive data by disassembling and analysing the design of a system component. Another intent may be to remove copy protection, circumvention of access restrictions.
- Creation of unlicensed/unapproved duplicates.
- **Academic/learning purposes.** RE for learning purposes may be understand the key issues of an unsuccessful design and subsequently improve the design.
- Competitive technical intelligence (understand what your competitor is actually doing, versus what they say they are doing).

II. Reverse engineering for Web applications

Web Applications are complex software systems providing to users access to Internet contents and services. In the last years they are having a large diffusion due to the growing of the diffusion of World Wide Web: nowadays the quantity of information and services available on the Internet is very remarkable.

Consequently, the diffusion of Web Applications in various different contexts is growing more and more, and the way business processes are carried out is changing accordingly. In the new scenario, Web Applications are becoming the underlying engine of any e-business, including commerce, e-government, service and data access providers. The complexity of the functions provided by a Web Application has also increased since, from the simple facility of browsing information offered by the first Web sites, a last generation Web Application offers its users a variety of functions for manipulating data, accessing databases, and carrying out a number of productive processes.

The increased complexity of the functions implemented by Web Applications is now achieved with the support of several different technologies. Web Applications generally present a complex
Structure consisting of heterogeneous components, including traditional and non-traditional software, interpreted scripting languages, HTML files, databases, images and other multimedia objects. A Web Application may include both 'static' and 'dynamic' software components. 'Static' components are stored in files, whereas 'dynamic' components are generated at run time on the basis of the user inputs. The Web Application components may reside on distinct computers according to a client-server or multi-tier architecture, and may be integrated by different mechanisms that generate various levels of coupling and flow of information between the components.

The high pressure of a very short time-to-market often forces the developers of a Web Application to implement the code directly, using no disciplined development process, and this may have disastrous effects on the quality and documentation of the delivered Web Application. This situation cannot be considered different from the one occurring for traditional software produced using no disciplined development process, and without respecting software engineering principles. Poor quality and inadequate documentation have to be considered the main factors underlying ineffective and expensive maintenance tasks, burdened by the impossibility of applying more structured and documentation-based approaches.

Reverse Engineering methods, techniques and tools have proved useful to support the post delivery life-cycle activities of traditional software systems, such as maintenance, evolution, and
Migration. The software community is now seriously addressing the problem of defining and validating similar approaches for Web Applications. Reverse Engineering allows recovering and abstracting documentation from an existing Web Application, to achieve comprehension, to assess quality factors, and so on

III. An Approach For Reverse Engineering Of Web Applications

Reverse Engineering (RE) processes are characterized by **goals**, **models** and **tools**. While tools aim to support the recovering process, goals and models specify the core of any RE process. Goals focus on RE motivations, they help to define a set of abstract views representing the reverse engineered application; models deal with the definition of the information to extract; models very often are complemented by intermediate representations upon which views are built
[12][13]. As in traditional software applications, WA behavior stems from static and dynamic elements, thus WA RE processes must recover:

- The static architecture;
- The dynamic interactions;
- The behavior.
-

Goals and models are therefore instantiated according to the above elements and consequently we propose a RE process (see Figure 1) encompassing the following phases:

1. Static Analysis;
2. Dynamic Analysis;
3. Behavioral Analysis.

The aforementioned phases recover views that can be adequately represented by extended UML diagrams. In particular, in this approach the following diagrams have been adopted: Class diagrams to represent the architecture of a WA;

- Sequence and collaboration diagrams to represent the dynamic model;
- Use Case diagrams to represent the WA behavior.

UML diagrams recovery requires as preliminary step the localization and identification of *elements* such as pages, frames, forms, scripts, (*elements* composing the application) and the relations among them. This is per-se a challenging activity in that it requires the parsing of Multilanguage files comprising a mixed of HTML, scripting languages, Java code etc. Moreover, to obtain independence from environments and tools, and to ensure a higher flexibility, recovered information is mapped into an intermediate representation stored in a repository.

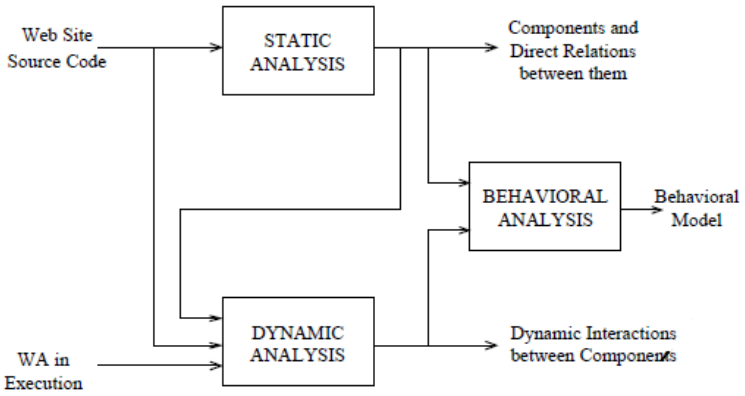

Figure 1. Reverse Engineering process of a Web Application.

a. Static Analysis

Static analysis does not execute the application. It recovers WA architecture components and the static relations among them. HTML files, directory structure, scripting language sources as well as any other static information (e.g., database structure, applet/servlet code) are processed. HTML pages and page sub-elements (frames, forms, widgets) composing the given page are localized, classified and recorded in an intermediate representation. Central to the RE process is the mapping between WA elements and object oriented entities, according to Conallen proposals [14], we made the assumption that HTML pages and relevant sub-elements (e.g., database connections) are mapped into classes, while link are mapped into relations. In other words, each identified component is a candidate class, while the links between pages or page elements are translated into candidate relations. *Parameters* of each component are considered as class attributes or as new classes. *Elementary* parameters may be modeled as attributes, while other, such as database connections, may require the introduction of new classes, see [15][16] for details.

The static analysis phase may be decomposed in the following activities:

- **Inventory** (e.g., WA files, databases and more generally components);
- **Component localization;**
- **Relation recovery;**
- **Intermediate representation generation.**

Recovered intermediate representation populates a repository upon which queries and views are constructed. It is worth noting that at the end of the static analysis phase a first approximation of the WA class diagram is available. Unparsed COTS or databases may serve to generate HTML documents that are not discovered by the static analysis phase.

b. Dynamic Analysis

The dynamic analysis phase rely on the static analysis results. The WA is executed and dynamic interactions among the components described in the class diagram are recorded. Dynamic analysis is performed *observing* the execution of the WA, *tracing* to source code (and, consequently, to the classes represented in the class diagram) any event or action. Traced events are those *observed* by the user or related to components external to the WA (e.g., third party databases or WEB sites). Events are the HTML pages/frames/forms visualization, the submission of forms, the processing of data, a link traversal, or a database query, etc. All elements responsible of these actions (typically links, scripts applets) are localized and the actions given to the method of related classes. The sequences of actions fired by an event, deriving from WA code control flow (e.g., access to a database following a user form submission) or from user actions (e.g., clicking on a link or submitting a form) are associated to sequences of messages exchanged between the objects of the WA. These sequences can be represented by **sequence diagrams** (or by **collaboration diagrams**). Notice that dynamic information is also used to verify, validate and, eventually, complete the class diagram extracted by the previous phase. In other words, repository information is complemented, augmented and assessed by dynamic analysis phase. The dynamic analysis phase may be decomposed in the following activities:

- **WA execution:** the WA is executed tracing the execution to the source code and to the class diagram.
- **Verification & Validation:** for each page displayed the class diagram has to include: a class corresponding to the page itself; a class for each component included into the page; appropriate relationships linking these classes. Hypertextual links are represented by associations among classes.
- **Detection of Interactions:** elements interacting by messages (caused by events fired from the control flow of the WA or from user actions) are detected and traced to the classes.
- **Abstraction of sequence/collaboration diagrams:** these diagrams are recovered to describe the operating scenarios. It is worth noting that the *Detection of Interactions* step localizes those 'active' elements responsible of WA 'actions'. These components may be scripts, applets, hypertextual links, etc. Each action performed is associated with a service in the class who is responsible for that action. While static analysis may be automatically performed, dynamic analysis requires human intervention. WEB server log files may be used to extract sequence of events, that played back mimic the user interaction while the loading of a page in a browser may be recognized as an observable user event. However, at the present level of tool implementation, the WEB server was not hacked (e.g., values corresponding to form inputs are not saved) thus, for example, a form filling requires the human intervention.

c. Behavioral Analysis

The behavioral or functional analysis essentially consists in abstraction processes oriented to detect the behavior of the WA from the user point of view. The recovered behavior is described by use case diagrams. This phase may be decomposed in the following tasks:

- **Analysis of sequence/collaboration diagrams:** all interaction diagrams are analyzed to abstract functional behaviors grouped into use cases;
- **Use cases definition:** use cases, actors, uses and extends relations are defined on the basis of the functional behaviors;
- **Use case diagrams abstraction:** the use cases defined in the previous step are represented in diagrams, at different levels of details.

IV. A Conceptual Model for WA

A WA conceptual model has to specify abstractions representing the application, its components and the relations between components. At first, and coarse grain, level, WA could be thought of
as composed by HTML pages. A page or a group of pages is/are responsible of a defined behavior of the WA. Pages are deployed on a WEB server; the WEB server processes client requests sending back HTML code, client scripts, applets, images, etc. What the WEB server sends to the client may or may not correspond to a physical file stored on the server: CGI bin, Servlet, server-side includes ASP and related technologies and tools may generate pages on-the-fly. More precisely the following preliminary taxonomy concerning pages was considered:

- ***server pages*** (i.e., pages that reside on the server) as opposed to *client pages* (i.e., the pages that are actually sent to a client);
- ***static pages*** as opposed to *dynamic pages*: a static page content is fixed, while a dynamic page content varies over time;
- ***simple pages*** as opposed to *framed pages*: a page may be divided into *frames* using a particular page, a *frameset*. A page appears in a frame specified by a *target*.
- ***unlinking pages*** as opposed to *linking pages*: a page, or a page component, may have *hypertextual links* to itself or to other pages. Links, in turn, may be both static and dynamic, i. e. the linked component is always the same or it is defined at run time.

A page will always be an aggregation/composition of finer grained components, such as *text, images, input/output form, text box, multimedia objects (sounds, movies), anchors, scripts, applets*, and so on. Page components (e.g., scripts or applets) may be *active components*. An active component is a component performing some processing action, for example, it may exchange data with other pages. A page, usually a server page, may be linked to other objects allowing the connections of the WA to a DBMS, managing the data of the WA, or to other systems. To obtain WA abstractions, it is necessary to extract, from the *source code*, all the information to identify pages, their components, the relations (both static and dynamic) existing among pages and components. WA abstractions must represent:

- The WA static architecture;
- The component dynamical interactions;
- The behaviors of the WA, assigning components to any given behavior.

The proposed RE approach focuses on a recover of components that may affect the behavior (and, thus the comprehension) of the WA; i.e. components like pages, scripts, applets, input/output forms, frames, links are recovered, while

V. CONCLUSION AND FUTURE WORK

This paper presents an Introduction of Reverse Engineering Process and also describes tool architecture to extract from existing WA UML diagrams dealing not only with static content, but also with the more challenging dynamic content. This will be very useful for economic relevance of the RE process needs to be validated by further studies: the paper assumes that recovering documentation, reflecting the *as-is* structure of a WA may cut down the costs related to the post-delivery WA life.

VI. References

[1] Chikofsky, E. J., Cross, J. H. (1990) Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software, Vol. 7 (10), pp13-17

[2] Muller, H.A., Jahnke, J.H., Smith , D.B., Storey, M.A., Tilley, S.R. and Wong, K.(2000). Reverse Eng

[3] Van Deursen, A. and Burd, L. (2005). Software Reverse Engineering. Guest Editorial, Jo of System and Software, Vol 77(3), Special issue on Reverse Engineering, pp209-212.

[4] Nelson, L. M. (1996) A Survey of Reverse Engineering and Program Comprehension. ODUCS 551 – Software Engineering Survey.

[5] Benedusi, P., Cimitile, A., De Carlini, U. (1992). Reverse Engineering Process, Document Production and Structure Charts. Journal of Systems and Software, Vol. 19, pp225-245.

[6] Di Lucca, G.A., Di Penta, M., Antonniol, G. and Casazza, G. (2001). An Approach for Reverse Engineering of Web-Based Applications. Proc. 8th Working Conference on Reverse Engineering, WCRE'01, IEEE, pp231-240.

[7] Hassan, A.E. and Holt, R.C. (2002). Architecture Recovery of Web Applications. Proc. Int. Conf. on Software Engineering (ICSE'02), pp349-359.

[8] Hassan, A.E. and Holt, R.C. (2003). A Visual Architectural Approach to Maintaining Web Applications. Anals of Software, Vol 16.

[9] Boldyreff, C. and Kewish, R. (2001). Reverse Engineering to Achieve Maintainable WWW Sites. Proc. 8th Working Conference on Reverse Engineering, WCRE'01, IEEE, pp249-257.

[10] Samuelson, P. (2002). Reverse Engineering under Sie ge. Communications of the ACM, Vol. 45, Issue 10, pp15-20.

[11] Kienle, M. H., German, D., Muller, H. (2004). Legal Concerns of Web Site Reverse Engineering. Sixth IEEE International Workshop on Web Site Evolution (WSE'04), pp41-50

[12] P. Benedusi, A. Cimitile, and U. De Carlini. A reverse engineering methodology to reconstruct hierarchical data flow diagrams for software maintenance. In *Proceedings of IEEE International Conference on Software Maintenance*, pages 180–191, Miami, FL, 1989.

[13] P. Benedusi, A. Cimitile, and U. De Carlini. Reverse engineering process, document production and structure charts. *Journal of Systems and Software*, 19:225–245, 1992.

[14] J. Conallen. *Building Web Applications with UML*. Addison- Wesley Publishing Company, Reading, MA, 1999.

[15] J. Conallen. Modeling web application architectures with uml. *Communications of the Association for Computing Machinery*, 42(10), October 1999.

[16] J. Conallen. Modeling web applications with uml. White paper, Conallen Inc., http://www.conallen.com/whitepapers/ webapps/ModelingWebApplications.htm, March 1999.]