Evaluating the Reverse Engineering Capabilities of Web Tools for Understanding Site Content and Structure: A Case Study

Scott Tilley
Department of Computer Science
University of California, Riverside
stilley@cs.ucr.edu

Shihong Huang
Department of Computer Science
University of California, Riverside
shihong@cs.ucr.edu

Abstract

This paper describes an evaluation of the reverse engineering capabilities of three Web tools for understanding site content and structure. The evaluation is based on partitioning Web sites into three classes (static, interactive, and dynamic), and is structured using an existing reverse engineering environment framework (REEF). This case study also represents an initial evaluation of the applicability of the REEF in the related but qualitatively different domain of Web sites. The case study highlights several shortcomings of current Web tools in the context of aiding understanding to support evolution. For example, most Web tools are geared towards new page design and development, not to understanding detailed page content or overall site structure. The evaluation also identified some aspects of the REEF that might benefit from refinement to better reflect Web tool capabilities that support common evolution tasks. For example, Web server log file analysis as a specialized form of data gathering and subsequent information presentation.

1. Introduction

Consider the following scenario. A new employee of a young and rapidly growing company is given the task of adding e-commerce capabilities to the company's Web site. The employee is a recent computer science graduate, but is unfamiliar with the structure of the Web site and has little experience with some of the technology involved. For example, the Web site uses a library of Perl scripts to implement CGI (Common Gateway Interface) applications, but the traffic on the site has increased sufficiently that the CGI interface is no longer meeting requirements due to performance problems. Upon further exploration of the site, the employee finds that the company was one of the early adopters of distributed object technologies, such as CORBA (Common Object Request Broker Architecture), but the vendor that the company selected to provide the CORBA services shipped a product that was flawed (it may in fact have been a beta release). Further complicating matters, the vendor has gone out of business. What can the fresh-faced employee do to gain a better understanding of the company's Web site, in order to carry out the modernization efforts?

In many ways, this scenario is similar to a traditional maintenance scenario. As software ages, the task of maintaining it becomes more complex and more expensive. Poor design, unstructured programming methods, and crisis-driven maintenance can contribute to poor code quality, which in turn affects understanding. Better understanding of a program aids in common activities such as performing corrective maintenance, system reengineering, and keeping documentation current. To minimize the likelihood of introducing errors during the change process, the software engineer must understand the system sufficiently well so that changes made to the source code have predictable consequences. But such understanding is difficult to recover from a legacy system after several years of operation. In the current atmosphere of accelerated schedules, years are reduced to months or even weeks.

The main difference in the scenario outlined above and a "traditional" maintenance task is the application undergoing maintenance: a large-scale Web site. It is a prime example of a modern system destined to be the legacy system of tomorrow. Indeed, several Web sites can already be classified as a legacy system, given their age, size, and constituent components. In the past, the subject system might have been a monolithic, mainframe-based payment processing system written in COBOL. Or it might have been a two-tier client/server system written in C/C++. For the Web application, there is a much wider range of implementation languages, heterogeneous networked applications, scripts on both the client (e.g., JavaScript) and on the server (e.g., Perl), and several other interpreted aspects of the system's architecture and implementation that make it more difficult to understand, and hence more difficult to evolve in a disciplined manner.

As with all complex software-intensive systems, to evolve a Web site one must first understand its current design, structure, and normal usage patterns. To gain such an understanding, reverse engineering techniques can be adapted to the application domain of large-scale Web sites.

0-7695-1050-7/01 $10.00 © 2001 IEEE

Through reverse engineering, one can identify artifacts, discover relationships, and generate abstractions that can be used to redocument the logical and physical Web site structure, thereby enabling its maintainers to gain a better understanding of the site's operational profile. However, at present it is unclear which particular reverse engineering activities are appropriate for understanding Web sites to better support their evolution.

As pointed out in [3], maintaining hypertext systems (of which the Web is the largest example) is one of the challenges facing the research community today. This paper describes an evaluation of the reverse engineering capabilities of three Web tools in the context of aiding understanding of site content and structure to support evolution. The evaluation is based on partitioning Web sites into three classes: static, interactive, and dynamic. The evaluation is structured using an existing reverse engineering environment framework (REEF). The REEF has previously been used to assess the capabilities of traditional software reverse engineering environments in support of program understanding. As such, this case study also represents an initial evaluation of the applicability of the REEF itself in the related but qualitatively different domain of Web sites.

The next section discusses the nature of Web site evolution, the underlying research area for this case study. Section 3 provides an overview of the reverse engineering environment framework and briefly discusses its uses in other case studies. Section 4 details the three Web tools examined as part of this case study. Finally, Section 5 summarizes our findings and outlines future work.

2. Web site evolution

Many organizations are faced with maintaining aging software systems that are constructed to run on a variety of hardware types, are programmed in obsolete languages, and suffer from the disorganization that results from haphazard maintenance. Evolving these legacy systems so that they continue to meet the organization's current requirements represents a significant challenge. Reverse engineering can be used to foster systematic evolution by providing a software engineer with a better understanding of the subject system's current design and overall structure. In the early 2000s, many of these legacy systems are in fact relatively young Web sites that have become exemplars of modern software-intensive systems.

Many Web sites are initially created to meet the relatively modest requirement of serving as a passive means of disseminating information. If the site is successful, it inevitably becomes a critical component of the organization's digital infrastructure, trying to meet the much more aggressive requirement of serving as an integration hub for a wide variety of activities, such as electronic commerce, streaming media, and online collaboration. The evolution from passive dissemination to active integration can be a complex process, one that is typically implemented in an ad-hoc and unpredictable manner. It therefore seems prudent to examine how they can evolve in a more discipline manner.

As they age, Web sites suffer from some of the same afflictions as any complex software system: their structure degrades, maintenance becomes increasingly problematic, and legacy applications and interfaces hinder evolution. But they are unique in several ways, not the least of which is their rate of change. Web sites are representative of applications that evolve in so-called "Internet time," an accelerated schedule which usually forces trade-offs, such as between engineering quality and time-to-market. The result is often a Web site that is developed with little regard to established software engineering principles, and is therefore difficult to evolve in response to changing requirements in a predictable manner.

One of the major attractions of Web sites over other forms of more permanent media, such as the printed page, is the ease with which content can be dynamically updated. This is similar to the advantageous flexibility that software offers, something not easily realized in hardware systems. However, this same flexibility is the root of the problems of both software-intensive traditional systems and Web sites: the very flexibility encourages sloppy and low-quality updates, typified by the edit-compile-debug cycle. This is particularly true for Web sites, which are often developed by people who may lack a formal computer science or software engineering background, thereby compounding the evolution problem. Reverse engineering technologies, suitably retargeted to support Web sites, may remedy this situation by uncovering hidden architectural patterns and design rationale.

3. The reverse engineering framework

The case study described in this paper is part of the initial requirements gathering phase of a larger research project which focuses on constructing a framework for evaluating Web site evolution mechanisms, and subsequently on developing new tools and techniques to address the problem. This Web site evolution framework is an extension to an existing framework developed for categorizing the capabilities of traditional software reverse engineering environments [14].

Reverse engineering is seen as an activity that does not change the subject system; it is a process of examination, not a process of alteration [5]. It directly supports the essence of program understanding: identifying artifacts, discovering relationships, and generating abstractions. This process depends on several factors, including one's cognitive abilities and preferences, one's familiarity with the application domain, and the set of support facilities provided by the reverse engineering

environment. Reverse engineering in the context of the overall software lifecycle is shown in Figure 1.

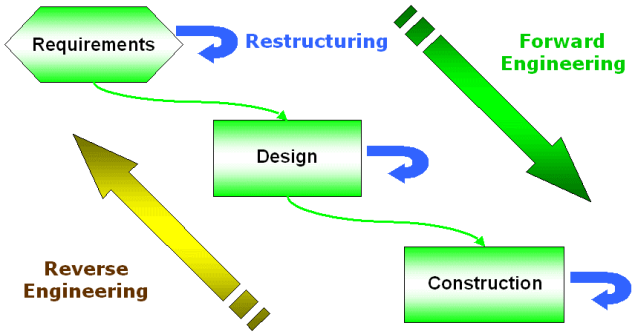

Figure 1. Reverse engineering in context

The REEF is based on a descriptive model that categorizes important support mechanism features based on a hierarchy of attributes, a set of which can then be compared using a common vocabulary. For example, when exploring a tool's data gathering capabilities, one looks for different forms of system examination, such as support of static analysis of HTML page structure. In this way, the model can be used to gain an understanding of the reverse-engineering tool's overall strengths, as described by the canonical activities, rather than its task-specific strengths.

At present, the model organizes attributes according to the broad categories of cognitive-model support, reverse-engineering tasks, canonical activities, quality attributes supported by the reverse-engineering environment, and miscellaneous characteristics. The canonical activities are data gathering, knowledge management, and information exploration. A detailed description of the canonical activities can be found in [13]; the rest of this section summarizes the overall framework itself.

3.1 Cognitive model support

A cognitive model describes the processes and knowledge structures used by the software engineer to form a mental representation of the program under study. Two common approaches to understanding often cited in the literature are a bottom-up functional approach that emphasizes cognition by *what* the system does, and a top-down behavioral approach that emphasizes *how* the system works. These two approaches are directly related to the level of domain expertise of the software engineer.

3.2 Reverse engineering tasks

There are many different reverse-engineering tasks. Five of the most important captured in the REEF are program analysis, plan recognition, concept assignment, redocumentation, and architecture recovery. The first three tasks can be viewed as pattern matching at different levels of abstraction. Program analysis is syntactic pattern matching in the programming-language domain, such as control-flow analysis and slicing. Plan recognition is semantic pattern matching in the programming-language domain, for example identifying common and stereotypical code fragments known as clichés. Concept assignment is semantic pattern matching in the application domain, enabling the engineer to search the underlying code base for program fragments that implement a concept from the application domain. Redocumentation is the process of retroactively providing documentation for an existing software system. Architecture recovery (also known as structural Redocumentation [19]) derives the overall gestalt of the subject system.

3.3 Canonical activities

The artifacts manipulated during reverse engineering can be classified into three categories: (1) data, the factual information used as the basis for study, reasoning, or discussion; (2) knowledge, the sum of what is known, which includes data and information such as relationships and rules progressively derived from the data; and (3) information, which is contextually and selectively communicated knowledge. The data artifacts are the raw bricks used as building blocks to support program understanding. They form the foundation for the higher level knowledge artifacts. The information artifacts can be created by abstracting up from the data artifacts and matching with expected results from knowledge artifacts.

Based on the description of these artifacts, three canonical reverse-engineering activities emerge: data gathering, knowledge management, and information exploration (which includes navigation, analysis, and presentation). All tasks carried out by a software engineer during a system understanding exercise can be mapped to a composition of one or more of these canonical activities supported by a reverse-engineering environment.

3.3.1 Data gathering

To identify the artifacts and relationships of a system and use them to later construct and explore higher level abstractions, raw data about the system must be gathered. Techniques used for data gathering include system examination (such as compiler-based static analysis), document scanning, and experience capture.

3.3.2 Knowledge management

Knowledge management refers to capturing, organizing, understanding, and extending past experiences, processes, and individual know-how. Understanding relies on knowledge-management techniques such as domain modeling to create, represent, and reason about the artifacts and relationships of interest.

3.3.3 Information exploration

The majority of system understanding takes place during information exploration; therefore, it is arguably the most important of the three canonical reverse-engineering activities. Data gathering is required to begin the reverse-engineering process. Knowledge management is needed to structure the data into a conceptual model of the application domain. But the key to increased comprehension is exploration because it facilitates the iterative refinement of hypotheses.

Information exploration makes use of the knowledge-discovery structures discussed above. Using a web metaphor, the software engineer navigates through the structured information space that represents the information related to the subject system. As part of the exploration, the information is analyzed and filtered with respect to domain-specific criteria. Various presentation mechanisms are used to clarify the resultant information. Exploration is a composite activity that includes navigation, analysis, and presentation.

3.4 Quality attributes

Barbacci *et al* define a quality attribute as a system requirement that is essentially non-functional in nature [1]. Examples of quality attributes include applicability, extensibility, and scalability.

3.5 Miscellaneous characteristics

There are a wide variety of system characteristics that do not properly fit into any of the other categories discussed above. However, when it comes to investigating a particular reverse-engineering environment, they can be equally important. Representative miscellaneous characteristics include the computing platform, ancillary requirements, and cost (both initial purchase and subsequent adoption and deployment of the tool).

3.6 Previous case studies using the REEF

The REEF has already been used to investigate the capabilities of several reverse-engineering tools that aid program understanding. The most comprehensive study published to date is documented in [12]; the study analyzed the DISCOVER development information system from Software Emancipation Technology (now Upspring Software) [16]. DISCOVER provided an excellent opportunity to look at the state-of-the-practice in program understanding tools. It also provided an opportunity to exercise the REEF itself.

A more recent case study using the REEF is documented in [15], where the stated goal was to understand the functional nature, high-level design, and implementation details of Gnut [10], a program that implements the Gnutella [18] peer-to-peer Internet file system protocol. Gnutella is representative of a new breed of net-centric applications that is both qualitatively and quantitatively different than the typical legacy systems that are usually the focus of program understanding exercises. The primary motivation for the analysis was an interest in evaluating the applicability of traditional reverse engineering tools to aid in the understanding of this type of software system.

4. Web tool evaluation

The evaluation of the reverse engineering capabilities of Web tools for understanding site content and structure is based on partitioning Web sites into three categories according to their level of complexity, interactivity, and functionality. Class 1 Web sites are those that are primarily static. These are also known as “brochure-ware” and are the easiest to develop and to maintain. They are the simplest of the three classes of Web sites, providing limited functionality with no user interactivity.

Class 2 Web sites are those that provide client-side interaction, possibly though the use of multimedia technologies such as Dynamic HTML [6] or Macromedia Flash [8]. Class 2 sites are more functionality advanced than Class 1 Web sites. They are more complicated to create and to analyze, due to the presence of embedded scripts and active components that provide the interaction.

Class 3 is the most complex of the three types of Web sites. They provide all the capabilities of Class 1 and Class 2 Web sites, plus the additional functionality of dynamic content supplied by a server-side database. Class 3 sites are the most complicated ones to design, develop, and deploy. As such, evolving Web site evolution in this context requires significantly more expertise, and correspondingly more powerful tools.

By partitioning Web sites into three classes, criteria for tool selection are clarified. For example, if the goal is to evolve a Class 2 Web site into a Class 3 Web site, then the capabilities provided by the Web tool should include analysis of client-side scripting and user interaction. Once a tool has been selected according to the class of Web site under examination and the perceived needs of the evolution task, the evaluation of its capabilities can begin.

The three tools examined in this case study are Microsoft FrontPage 2000 [9], Macromedia Dreamweaver UltraDev 4 [7], and Big Picture Technologies SmartSite 3 [2]. These tools were chosen because they are representative of commercial tools that target different market sectors and support different classes of Web sites. FrontPage is a very successful product for both HTML page development and site management that is now part of the Microsoft Office family. UltraDev is a new product from Macromedia that builds upon the capabilities of the Dreamweaver 3 HTML editor by adding server-side

database support. Smart Site is different from the other two tools in this case study in that it is focused solely on Web site analysis, not on content development.

The tools were assessed using the framework described in Section 3, according to plausible evolution scenarios for large-scale Web sites. For example, adding e-commerce capabilities to a Class 1 Web site, changing site structure and navigation aids to better reflect actual usage patterns and personalization issues to a Class 2 Web site, or changing database systems in a Class 3 Web site.

Due to space restrictions, this section only discusses selected aspects of the framework for each tool examined in this case study. The focus is on those aspects of each tool that are particularly noteworthy in a positive sense. As none of the tools examined support the notion of cognitive models as described in the REEF, the topic is not covered.

4.1 FrontPage 2000

Microsoft's FrontPage 2000 product is sometimes erroneously viewed by Web professionals as a "toy" HTML editor. However, it offers significantly more functionality than many of its competitors targeted at the same class of Web sites. In fact, it is one of the few tools that offer integrated page design and site management features in one application. The promotion of FrontPage to a full-fledged member of the Microsoft Office family has greatly increased its popularity and use, witnessed by the proliferation of Web presence providers who specialize in hosting FrontPage-enabled sites (see below).

4.1.1 Reverse engineering tasks

Redocumentation. FrontPage's capabilities in the area of team management, group collaboration, task views, controlled check-in/out of files, and workflow reports can be categorized as a type of reverse engineering task (redocumentation), since the end result is a record of the development of the site. Of course, this documentation cannot be easily captured after the fact, but if FrontPage was used in the site's initial development then a concise record of the tasks performed by the team is available. In fact, one of several views of the Web site is called "Tasks" and displays just this information. The task view functionality in FrontPage is unique to the tools examined in our case study.

4.1.2 Canonical activities

Data Gathering. In terms of canonical activities, FrontPage provides several unique features. Under the system examination category of data gathering, FrontPage provides the powerful ability to import an entire Web site from the Internet into a local workspace. This feature is very useful for upgrading existing sites that may not have been developed using FrontPage. It is also a useful learning feature, since one can peer inside a well-done site to examine its internal structure.

Knowledge Management. FrontPage provides a built-in mechanism for creating, and managing site-wide navigation aids. This type of organizational knowledge management aid is somewhat unique in Web site tools. It is sometimes difficult to use properly, but when it works it can be a great time-saver. It also codifies one of the most common needs for Web sites of all sizes: consistent navigation for the user.

To use the navigation features properly, the "Navigation" view is used. Pages are dragged from the window listing all files and directories in the site onto the navigation view. This action causes the pages to be entered into the FrontPage navigation records, so that they are managed automatically. This means if the user is visiting the page in question, a highlighted image (or text) is displayed, while other pages in the same level on the hierarchical navigation view are displayed in de-emphasized graphics. FrontPage even generates the JavaScript necessary to implement image rollovers to highlight the images in response to mouse movement. A snapshot of the navigation view from a small-scale Class 1 Web site is shown in Figure 2.

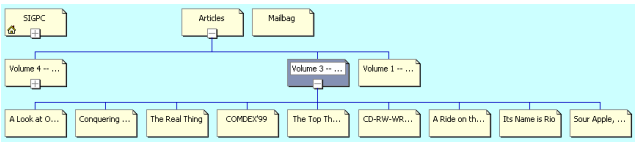

Figure 2.FrontPage navigation view

Information Exploration. Another important feature that is not unique to FrontPage but is well supported is multiple views of the Web site. This is critical to understanding the site's structure. Several views are available. The "Hyperlinks" provides an alternate perspective on the Web site, highlighting the macro-level HREF links between pages. The different types of artifacts references are shown using different icons.

4.1.3 Quality attributes

Applicability. FrontPage is clearly targeted at Class 1 Web sites, those that are primarily static and client-side intensive. However, Microsoft does make FrontPage applicable to Class 3 Web sites by providing the ability to develop sites with server-side database access. Although FrontPage works best with Access or SQL Server, any ODBC-compliant database will work. Database wizards are provided for using Active Server Pages and server-side scripting. However, this functionality is geared towards site development, not site evolution. Little traceability between the client and server is possible.

Extensibility. From an integration point of view, the 2000 version of FrontPage is now tightly coupled to the

Microsoft Office family of products. If the Web will be uploaded to a site that has the FrontPage server-side extensions installed (described below), then the user can exploit this feature in designing the site's content and structure. FrontPage provides a number of built-in "web bots" that implement common functionality for the Web, such as search engines or form processing.

4.1.4 Miscellaneous characteristics

Ancillary Requirements. One of the most interesting aspects of FrontPage is its support of (some might say reliance on) server-side extensions. When uploading a local copy of a Web site to its destination on the Internet, FrontPage interrogates the server; if it is finds it has been augmented with the server-side extensions, then it uploads the Web site using a much more rapid method than if it is forced to use standard FTP to transfer the files to their external destination.

Computing Platform, Cost. FrontPage is available only for the Microsoft Windows platform. At a retail price of $149 (but widely available for under $75), FrontPage is clearly targeted at the typical Microsoft Office user. However, its page design tools are more than acceptable. From our case study and personal experience, the site management capabilities of FrontPage, especially in terms of their mapping to equivalent functionality in the framework and their support of overall Web site evolution through importation and understanding, is significant.

4.2 Dreamweaver UltraDev 4

The UltraDev product from Macromedia is recent addition to their Web development product family. UltraDev is essentially a combination of the Dreamweaver HTML editor and the recently acquired DrumBeat e-commerce engine. Since it includes all of Dreamweaver, UltraDev can be used to create Class 1 and Class 2 Web sites. However, its strength is really in creating and managing Class 3 Web sites by adding server-side database connectivity and processing. UltraDev was built specifically for creating dynamic Web applications by combining client-side HTML authoring with server-side scripting support through ASP, JSP, and CFML.

4.2.1 Reverse engineering tasks

Plan Recognition. For maintaining consistency across the entire Web site, UltraDev provides a powerful search-and-replace facility. This is useful for changing Web content during upgrades or alteration of common page elements, such as headers or footers.

Redocumentation. One of the most important characteristics of UltraDev for editing HTML content is that it supports "round-trip" editing. This means that the editor does not alter the HTML code that you may have coded by hand. Such alteration of code was a problem for other editors in the past, notably the pre-2000 release of FrontPage.

Program Analysis. A common evolution task for Web sites is to incorporate new content from other tools. Although somewhat limited, Microsoft Word can (and often is) used to save documents into HTML format. However, the resulting HTML is quite "dirty," riddled with Internet Explorer-specific directives and unnecessary style formatting. UltraDev provides a feature specifically targeting this problem, called "Clean up Word HTML." Basic cleanup strips redundant and Word-specific tags; detailed cleanup removes more Word tags and unnecessary CSS elements. This functionality is directly analogous to a software reverse engineering tool that can pretty-print or reformat poorly structure code.

4.2.2 Canonical activities

Data Gathering. With UltraDev you can import an existing Web site – but only one from a local network disk or an FTP site to which you have access. You cannot load from a URL on the Web, as you can with FrontPage. Publishing is also with FTP. The site synchronization between local copy and remote can be counter-intuitive. UltraDev is also quite slow when gathering all the site's artifacts from the Internet.

Knowledge Management. The distinguishing feature of UltraDev is its strong support for server-side development. Since the Web is similar to a distributed heterogeneous software system, visibility into all aspects of the site is crucial to gaining a complete understanding of the project. The "live data preview" feature of UltraDev can be used to gain this type of visibility.

The live data preview lets you view and edit server-side data in the workspace and make edits on the fly. In other words, this is a real-time connection to the database, where the designer can make changes to the way a page is laid out and presented. The page is then updated both directly to the server and to the desktop where the information is dynamically updated. It saves time and avoids repetitive tasks, such as designing first, uploading later, and finally testing files after the fact.

4.2.3 Quality attributes

Applicability. Support for cross-server development is unique to UltraDev. If you are migrating your site from ColdFusion Markup Language to the more common Java Server Pages for example, UltraDev can handle the task.

Extensibility. UltraDev integrates well with other Macromedia products, such as FireWorks for graphics and Flash for vector-based animation. It also provides a JavaScript API for extending UltraDev's functionality.

4.2.4 Miscellaneous characteristics

Cost. UltraDev is the most expensive of the three tools examined in this case study. Dreamweaver 3 by itself costs

about $199, while UltraDev costs about $599. However, there are Enterprise editions of UltraDev that support high-end Web site evolution tasks such as load balancing and task partitioning that significantly more. Macromedia has segmented the UltraDev market, but the tool is clearly targeted towards higher-end Web professionals.

The learning curve for UltraDev is steeper than that of the other tools in this case study, but that can be attributed in part to the complexity of the task that UltraDev is used for. Creating and managing Class 3 Web sites is challenging, even with good tools.

4.3 SmartSite 3

Big Picture Technologies' SmartSite product is specifically geared towards site analysis and management. Unlike the other two tools examined in this case study, Smart Site does not provide the usual Web tools such as HTML editors or site publication aids. Instead, it focuses solely on providing insight into the nature of existing Web sites. As such, Smart Site is the tool most oriented towards Web site evolution and provides the strongest reverse engineering-like capabilities of the three tools discussed.

4.3.1 Reverse engineering tasks

Program Analysis. Analogous to some of the activities carried out during program analysis, Smart Site provides extensive markup validation. It can analyze HTML, SGML, XML (and associated DTDs) and report on the content's compliance to various W3C standards and accessibility initiatives [17].

SmartSite can also examine the use of ALT and META tags on the selected pages. Proper use of the ALT tags is an important feature in making Web pages accessible to those with visual disabilities, and will likely become a more common evolution activity. Hence, the reporting on the use (or misuse) of these tags is useful.

4.3.2 Canonical activities

Data Gathering. Focused as it is on complete site analysis, Smart Site supports processing of Webs that are local or already posted to the Internet. This type of "site grabbing" capability can be tailored by choosing to download or exclude graphics, components, and archives.

Site-wide data gathering through system examination is in fact richer than just static page processing. Smart Site also integrated processing of Web server log files into its analysis. This is an extremely powerful and valuable capability, as information provided by log files can be used to tailor site settings to better reflect actual usage patterns.

Knowledge Management. Smart Site provides many reports that can uncover and in essence (re)document the business rules inherent in the Web site's content and structure. A variety of automatically produced reports are available, such as the catalog of resources used in the site (and their inter-relationships).

Information Exploration. One of the main strengths of Smart Site are the many different types of views that it provides. One of the most useful is the spider view, shown in Figure 3. It displays the spatial relationship between artifacts of the Web site. This type of view is not new; it has been used in traditional reverse engineering tools as an aid in uncovering and visualizing hidden relationships between software artifacts. However, the inclusion of this type of view of the Web may be of great use in understanding the overall site's structure and organizational axes.

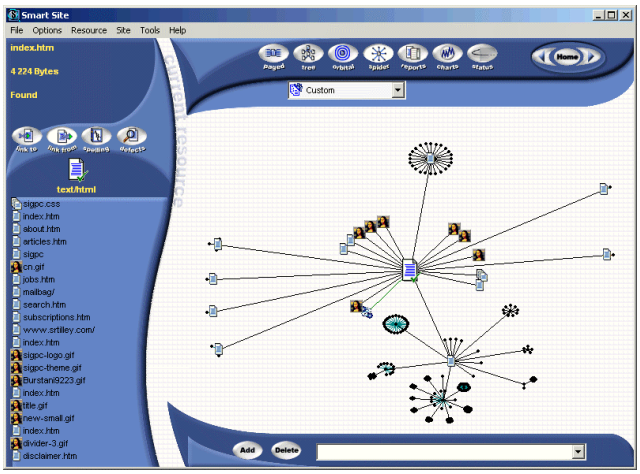

Figure 3. SmartSite spider view

5. Conclusions

This paper described an evaluation of the reverse engineering capabilities of three Web tools for understanding site content and structure. The evaluation was structured using an existing framework for categorizing the capabilities of software reverse engineering environments. An important part of the case study was an assessment of the framework itself, to ascertain its applicability to the related but qualitatively different domain of Web sites.

Section 4 discussed the evaluation of three tools: Microsoft FrontPage 2000, Macromedia UltraDev 4, and Big Picture Technologies SmartSite 3. A complete summary of this evaluation in the context of the REEF is shown in Tables 1 to 4 at the end of this paper. Based upon this preliminary case study, implications for researchers and developers, as well as implications for the REEF itself, have been identified.

A general conclusion regarding Web site tools is that they closely resemble the software CASE tools of several years ago [4]. As with software systems, the specialized tools for system maintenance and reverse engineering for understanding are not yet mature or even widely available.

5.1 Implications for researchers and developers

Based on this (admittedly limited) study, several shortcomings in current approaches to system evolution as applied to Web sites were identified. For example, the tools are primarily based on static analysis, and support Class 1 Web sites the most. Support for Class 2 Web sites is less developed, and support for Class 3 Web sites is rarer still. It is important that tools support Class 3 Web sites in the future: since Web sites are interactive in nature, their operational profile may not be accurately reflected by examining only HTML document structure. Web sites are distributed systems with a sometimes-blurred distinction between the roles of client and server. Moreover, given the nature of linking a hypermedia, Web sites are very loosely coupled; a complete picture of the Web site and its entire constituent parts—however ephemeral—is hard to create.

When they do support Class 2 or Class 3 Web sites, most tools usually work only on homogenous systems (e.g., system implemented on a single platform); Web sites are heterogeneous, using HTML and/or XML, embedded scripts in several languages and dialects, third-party plug-ins, applets, servlets, linked cgi-bin programs, and so on. This shortcoming is analogous to software reverse engineering tools that only support a single programming language.

To gain a complete understanding of a large-scale Web site, the tool should reverse engineer all related components. Most tools tend to focus on the micro-level of Web pages, or on the macro-level of Web sites, but not both levels at once. To properly understand the entire Web, an holistic approach that manages both perspectives and allows the maintainers to move freely from one view to the other is preferable.

5.2 Implications for the REEF

In evaluating the utility of the REEF, several questions must be addressed. Did the investigation into the tool's capabilities reveal any fundamental flaws in the framework? Is the framework complete? That is, are there features offered by the tool that could not be mapped into the canonical activities of the model? Conversely, are there areas of support described in the REEF that do not seem readily available in the tool in question? Finally, how easy was it to use the REEF to classify the capabilities of the tool?

As mentioned in Section 3, the cognitive modeling portion of the framework does not yet seem to be applicable to the domain of Web sites, mainly due to the lack of support from the tools evaluated. This is may change as sites mature and as tools become more powerful, but for the this study this criteria was not used.

The reverse engineering tasks do have some counterparts for the Web. For example, in several instances in Section 4, analogies to program analysis but for HTML pages were discussed. However, other reverse engineering tasks such as concept assignment seem less relevant. This section of the framework may need to be replaced by a more expansive section covering representative Web site evolution tasks, such as log file analysis. This is a prime area for future investigation.

The canonical activities section of the framework has some portions that seem directly applicable to Web sites, such as data gathering (in particular, static analysis). However, other parts, such as knowledge management, seem less relevant – or at least less well supported – in current tools. The most important activity is clearly information exploration, and it is this section that would also be interesting to further develop in future work.

The portions of the quality attributes section of the framework that seem most pertinent are the portions on extensibility and integration. There is a obvious need to incorporate end-user programming capabilities into the tools examined, for the same reason it was important to incorporate them into software reverse engineering tools: a third-party market for add-ons that will help spread the use of the tools.

5.3 Future work

The work described in this paper is part of the initial requirements gathering phase of a larger research project that focuses on constructing a framework for evaluating Web site evolution mechanisms. One of the next steps is to perform additional studies of representative Web sites and tools. For this next step in the research, the requirement for the tool to be commercial and widely available may be relaxed to include leading-edge research prototypes that represent state-of-the-art rather than state-of-the-practice. Other non-traditional tools, such as network monitors, may also be investigated with respect to their supporting role in understanding Web site characteristics. Through these studies, the REEF will continue to evolve.

As a complimentary effort, it might also prove fruitful to examine the applicability of traditional software reverse engineering tools to the problem of Web site evolution. There may be an opportunity for such tools to be retargeted to this new area. Tools such as Source Navigator [11], that provides a developer kit with a documented API, would be a first choice. Once both of these two phases of analysis have been completed, the next step would be to investigate integrating new functionality into existing products, so that software reverse engineering and Web site tools can be combined to create a more effective solution.

This work represents one small step towards creating an integrated approach to dealing with the challenges inherent in understanding large-scale Web sites by identifying areas for further research and by exercising an

assessment framework for structuring subsequent studies in the area. There are many exciting opportunities for both academic researchers and commercial tool developers to explore in the nascent field of Web site evolution.

References

[1] Barbacci, M.; Klein, M.; Longstaff, T.; and Weinstock, C. *Quality Attributes*. Technical Report CMU/SEI-95-TR-021. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995.

[2] Big Picture Technologies, Inc. *SmartSite 3*. Online at http://smartsite.bigpic.com.

[3] Brereton, P.; Budgen, D.; and Hamilton, G. "Hypertext: The Next Maintenance Mountain." *Computer* 31(12):49-55, December 1998.

[4] Brown, A.; Carney, D.; Morris, E.; Smith, D.; and Zarella, P. *Principles of CASE Tool Integration*. Oxford University Press, 1994.

[5] Chikofsy, E.; and Cross, J. "Reverse Engineering and Design Recovery: A Taxonomy." *IEEE Software* 7(1):13-17, January 1990.

[6] Goodman, D. *Dynamic HTML: The Definitive Guide*. O'Reilly & Associates, 1998.

[7] Macromedia Corp. *Dreamweaver UltraDev 4*. Online at http://www.macromedia.com/software/ultradev.

[8] Macromedia Corp. *Flash 5*. Online at http://www.macromedia.com/software/flash.

[9] Microsoft Corp. *FrontPage 2000*. Online at www.microsoft.com/frontpage.

[10] Munafo, R. *Gnut*. Online at http://www.mrob.com/gnut.

[11] Redhat Inc., *Source-Navigator*. Online at http://sources.redhat.com/sourcenav.

[12] Tilley, S. "Discovering DISCOVER." Technical Report CMU/SEI-97-TR-012. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997.

[13] Tilley, S. "The Canonical Activities of Reverse Engineering." *Annals of Software Engineering*, Volume 9 (May 2000): 249-271. Baltzer Scientific / Kluwer Academic.

[14] Tilley, S. *A Reverse-Engineering Environment Framework*. Technical Report CMU/SEI-98-TR-005. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998.

[15] Tilley, S.; and DeSouza, M. "Spreading Knowledge About Gnutella: A Case Study in Understanding Net-Centric Applications." To appear in the *Proceedings of the 9th International Workshop on Program Comprehension* (IWPC 2001: May 12-13, 2001; Toronto, Canada). Los Alamitos, CA: IEEE Computer Society Press, 2001.

[16] Upspring Software Inc. *DISCOVER*. Online at http://www.upspringsoftware.com/products/discover.

[17] W3C. *Web Accessibility Initiative*. Online at www.w3.org/WAI.

[18] Wego.com, Inc. *Gnutella*. Online at http://gnutella.wego.com.

[19] Wong, K.; Tilley, S.; Müller, H.; and Storey, M.-A. "Structural Redocumentation: A Case Study," *IEEE Software* 12(1):46-54, January 1995.

Table 1. Reverse engineering tasks

	FrontPage	UltraDev	SmartSite
Program Analysis	Reports, link checks	Clean HTML	Reports, Validation
Plan Recognition	No	Search/replace	No
Concept Assignment	No	No	No
Redocumentation	Task view	History palette, Design notes	Views
Architecture Recovery	Load site	Load site	Load site

Table 2. Miscellaneous characteristics

	FrontPage	UltraDev	SmartSite
Computing Platform	Windows	Windows, Mac	Windows
Ancillary Requirements	Server extensions	Server database	None
Cost	$149	$599 +	$40

Table 3. Quality attributes

		FrontPage	UltraDev	SmartSite
Applicability	Application Domain	Classes 1-2; partial 3	Classes 1-3	Class 1, 2
	Implementation Domain	HTML, JS, VB	HTML, ASP, JSP, CFML, …	HTML, SGML, XML
Extensibility	Integration Mechanisms	Server-side, MS Office	Macromedia, JavaScript API	No
	End-User Programmability	No	JavaScript	No
	Automatability	No	No	No
Scalability		Medium	High	Medium

Table 4. Canonical activities

			FrontPage	UltraDev	SmartSite
Data Gathering	System Examination	Static	HTML	HTML, scripts, SQL	HTML, SGML, XML
		Dynamic	DB wizard	Extensive DB	No
		Mixed	No	No	No
	Document Scanning		No	No	ALT, META tag analysis
	Experience Capture		Task view	History palette	No
Knowledge Management	Organization		Nav. bars	Server-side	No
	Discovery		Site loading	Import site	Reports
	Evolution		Nav. bars	DB visibility	No
Information Exploration	Navigation	Selection	Explorer views	Text find	Text find
		Editing	Nav. bars	HTML, server	No
		Traversal	Multiple views	Windows	Windows
	Analysis	Types	Links, load	Links, loads	Links, loads
		Levels	No	No	No
		Automation	No	Some	No
	Presentation	Multiple Views	Yes	Yes	Many
		Visualization Techniques	Several views	Yes	Many views
		User Interface	Fixed	Changeable	Fixed