We investigated automatic and semiautomatic Web site analysis with our tool ReWeb, focusing on a site’s architecture and evolution. Our case study demonstrates how ReWeb addresses the need to support Web site maintenance and evolution while retaining and possibly improving quality.

Either directly or indirectly, many companies’ business depends on their Web presence. Industry and academia are developing technologies for sophisticated, advanced Web site development that will offer a variety of features and facilities for the different company needs. However, only a few studies have addressed the problem of Web site evolution and restructuring.1,2 (See the “Related Work” sidebar for more information.)

Similar to software system development, Web developers can produce high-quality and reliable sites by adapting proper methodologies and techniques. In particular, developers should anticipate the maintenance phase, designing sites for change and evolution. In fact, it’s likely that maintenance will absorb considerable effort (as it does with software) given a successful site’s high pressure for change.

Our contribution to Web site maintenance is in defining a set of analyses that extracts a description of a site’s main architectural features and of their evolution. Developers can use the proposed analyses when their Web sites need to evolve while retaining and possibly improving their quality. Even with consolidated design practices, analyzing an implementation remains important, because developers can contrast the recovered models with the initial architecture to highlight discrepancies and make traceability links explicit. We derived some of our analyses from the ones traditional software systems use. We divided them into two categories: analysis of the structure and the evolution.

To implement these ideas, we developed the tool ReWeb. It can download and analyze Web sites, and its graphical interface provides search and navigation facilities. It uses colors to display the history and pop-up windows for the outcome of structural analyses. The results we obtained by using the tool to restructure a real-world Web site illustrate its potential.

Web site analysis

In this article, we identify a Web site as all the information that users can access from a given Web server. We consider documents accessed through different servers external to the given site. The Web site’s structure is its page organization and links between pages.

We can model a Web site’s structure with a graph and then apply several known analyses (including flow analyses, traversal algorithms, and pattern matching). We can then represent a Web site’s evolution by using colors as time indicators. In this way, we can show users the history of individual pages and links in a compact and expressive form, highlighting updates that might degrade the original structure.

Because we’re supposed to conduct a Web site’s analysis in the development environment, which should facilitate understanding and maintenance activities, we consider some information that external users can’t normally access browsing the site available. In some cases, we can automatically extract the information from artifacts (such as CGI scripts) the Web server uses, while in other cases we’ll assume the site’s developers manually provide it.

Modeling the structure

We can represent the structure of a Web site without frames as a directed graph $W = (P, E)$, where each node $p \in P$ represents a single HTML page, and an edge $e = (p, q) \in E$ connects two nodes $p, q$ if an HTML link exists from the page associated to $p$ to the page associated to $q$. (We use the terms node and page interchangeably, as well as edge and link.)

We must extend the basic model for a Web site’s structure to account for the peculiarities of pages containing frames and of dynamically gen-
We horizontally joined frames \( f_1 \) and \( f_2 \) to intuitively suggest decomposition into frames and collapsed incoming edges into a single edge. The links between \( f_1 \) and \( p_2 \) and between \( f_2 \) and \( p_3 \) indicate that the \( p_2 \) and \( p_3 \) pages are initially loaded into \( f_1 \) and \( f_2 \), respectively. Finally, we used the dashed edge connecting \( p_2 \) to \( p_4 \), labeled \( f_2 \), to show that a link in \( p_2 \) doesn’t result in the navigation within \( f_1 \) toward a different page. It loads page \( p_4 \) into
Figure 2. Sample Web site organization including a dynamic page with outgoing dynamic links.
connected to a page in the $d_2$ directory.

Researchers have traditionally applied flow analyses—such as computing dominators, reaching definitions, reachable uses, available expressions, and copy-constant propagation—to solve problems related to the static analysis of computer programs. Given the graph representation of the Web site we propose here, it’s simple to extend the range of applicability of flow analyses to Web sites.

**Reaching frames**

Computing the reaching frames determines the set of frames in which a page can appear. A page can load into a frame as the initial page in it, or it can be reachable from the initial page. Moreover, a page can load into a frame because a link in another page forces it to. Pages reachable from it will appear in the same frame.

To compute the reaching frames of a Web site’s pages, we can use a specialization of the flow analysis framework. The propagation of flow information is the basic idea behind flow analysis. In the case of the reaching frames, the information we want to propagate is the frame’s identifier (name), and the generators of such information are the nodes of the type frame ($F$) and the edges forcing page loading into a specified frame ($E_2$).

The outcome of the reaching frames analysis helps us understand the assignment of pages to frames. This analysis makes the presence of undesirable reaching frames clear. Examples of undesirable reaching frames include the possibility of loading a page at the top level, while it was designed to always be loaded into a given frame or loading a page into a frame where it shouldn’t be.

**Dominator**

When users traverse a Web site’s pages to reach a document, they can choose several paths. Nevertheless, it’s impossible to reach a given page without traversing a set of other pages, called the page of interest’s dominators. The page initially provided by the site server—say index.html—is a dominator of any other page in the site. We consider it the root of a Web site’s graph representation.

More formally, given a graph rooted at node $r$, a node $m$ is a dominator of node $n$ if every path from $r$ to $n$ traverses $m$. We can compute the dominator set for each node of a rooted graph by exploiting a well-known specialization of the flow analysis framework.

Knowing a page’s dominators is useful to understanding a Web site’s navigation constraints. We expect a Web site designed to provide several alternative navigation paths to have only the root in the dominator set of its nodes. Sites in which traversing a given page is considered mandatory—for example, because it contains advertising material or important information—will have it in every node’s dominator set.

**Shortest path**

Reaching a page of interest in a Web site might require users to traverse several pages. The minimum number of pages that users must visit before reaching the target document is useful information about a Web site.

Given a graph rooted at node $r$, the shortest path from $r$ to each graph node $n$ is the path from $r$ to $n$ with the minimum total weight associated with the edges. In this article, we weight edges 1 when they represent real user selections (mouse clicks) and weight them 0 when page loading is automatic. In particular, edges going from a node $p \in N$ to a node $f \in F$ have weight 0, because they represent page decomposition into frames. We must weight edges connecting a frame $f \in F$ to its initial pages (from $N$) 0, because the loading is automatic.

Information about the shortest path to each page in a site indicates potential problems for users when they search for a given document if such paths are long. A well-designed Web site should provide short paths to the most relevant documents.

**Strongly connected components**

During Web site navigation, users might not be able to reach some pages from the current page without restarting from the beginning or from a previously visited page. Regions with fully circular navigation facilities often appear in Web sites, and in several cases, the site is one such region as a whole.

Given a Web’s site’s directed graph, its strongly connected components are the equivalence classes of nodes that are mutually reachable from each other. In other words, if nodes $n_1$ and $n_2$ belong to the same strongly connected component, the site’s developer has ensured that a path exists from $n_1$ to $n_2$ and from $n_2$ to $n_1$.

Strongly connected components containing several nodes suggests that the page organization ensures that the user never gets stuck. Circular paths leading to previously visited pages let the user explore alternative pages.
Pattern matching

Developers use recurrent patterns when designing Web sites. Matching a library of known patterns against a given Web site results in the identification of portions of the site that comply with a predefined structure from the pattern library. The algorithms we used for patterns matching are those detecting the isomorphism of a subgraph with respect to a set of model graphs. For specific patterns, we exploit ad-hoc solutions.

The tree, hierarchy, diamond, full connectivity, and indexed-sequence are examples of reference patterns that we can match against a given Web site (see Figure 3). A portion of a Web site has a tree organization if its nodes and edges form a tree—that is, its graph representation is acyclical and each node has exactly one parent, except for the tree root, which has no parent. A Web site region has a hierarchy pattern if it’s acyclical, but more than one parent exists. The diamond hierarchy is characterized by a single entry point—a top node (root) with no parents—and a single exit point—a bottom node with no children. A subgraph matches a full connectivity pattern if each node in the subgraph is connected to all other nodes in the subgraph. A full connectivity subgraph is also a strongly connected component, but the inverse isn’t true. Indexed sequence pages are arranged into a single- or double-linked list, so that each page makes the next one available, and optionally the previous one. In addition, an index page exists from which all other pages are directly accessible.

To determine a site’s pattern, it might be necessary to remove some navigation links aimed at facilitating operations, such as going to the home page or moving backward, which don’t contribute to the pattern structure.

Different patterns implemented within a Web site enforce different navigation styles. Tree and hierarchy patterns correspond to a search strategy based on the refinement of general notions into more detailed ones. At each step, the user makes some choice, thus successively restricting the area of interest. The diamond organization enforces a single exit point, associated, for example, with a final operation that users will perform after selecting the items of interest or to a final document that users will view before they leave that portion of the site. Fully connected regions provide full access to every document from each page, without any need for query refinement; all information is available from everywhere on the Web site. Pages are made available sequentially in a Web site organized according to the indexed sequence pattern, when there’s a natural order in which pages are expected to be traversed, as in book chapters. The index allows jumping in the middle.

History

Knowing a Web site’s history is useful for documenting the events leading to its current organization and for identifying the reasons for potential structural problems. Analyzing a Web site’s evolution requires comparing successive versions of its pages and graphically displaying the differences. Works on history visualization for software systems greatly influenced the approach we chose.

Given two versions of a Web site at different dates, a comparison should determine which pages were added, modified, deleted, or left unchanged. Tracing the evolution of an entity over time requires us to define the object’s identity, so that we can recognize it later. In our case, we can simplify the problem if we assume that the name of the file containing the page was preserved from one version to the next. In this way, a map exists between nodes in the graph representation of the previous site and nodes in the graph for the current site. The map simply pairs nodes with the same name. All nodes in the old graph without a corresponding node in the new graph represent deleted pages, while all nodes in the new graph without a counterpart in the old graph are added nodes. When we can find a corresponding node in the old graph for a node in the new graph, we classify it either as a modified node or a node left unchanged according to the output of a character-
by-character comparison of the associated HTML pages. Because changing a page’s name has an impact on all referencing pages (including external pages), which must update their links with the risk of missing an update, our assumption of name preservation seems reasonable.

We can enrich a site’s graph representation with historical information by coloring the nodes and associating different colors with different time points. In particular, we can use a scale of colors ranging from blue to green to red to represent nodes added or modified in the far past, in the medium past, or recently.

We can also apply history visualization to a Web site’s links. When a developer first introduces a link from a page to another page, the edge in the graph representation of the site assumes the color of the source node—that is, of the enclosing page. Then, we maintain such a color even when the page evolves, changing its color, if modifications don’t affect the link.

ReWeb

We developed the ReWeb tool to download and analyze Web sites. The tool isn’t in the public domain, but an online Web site analysis service will be soon available from our Web site (http://zeus.itc.it:4444). The difficulties we encountered in constructing the tool were due to irregularities and ambiguities in HTML code and to the current state of Web technology, which offers a large spectrum of alternatives to Web site implementation.

Similar to the Eichmann paper, we classify sites according to a taxonomy characterized by dynamism, page decomposition, and data flow:

- Level 0: static pages without frames
- Level 1: static pages with frames
- Level 2: dynamic pages without data transfer from client
- Level 3: dynamic pages with data transfer from client

The current version of ReWeb downloads and analyzes sites at level 0 or 1. It can partially handle sites at levels 2 and 3, provided that they exploit the FORM construct for user input. ReWeb consists of three modules: a Web spider, an analyzer, and a viewer.

The Web spider downloads all of a target Web site’s pages, starting from a given URL. It downloads all the pages it finds within the site host and marks them with the date of downloading. We don’t consider the HTML documents outside the Web site host.

The analyzer uses different versions of the downloaded Web site—the Web Spider’s output—to calculate the difference between successive versions of the site. In this phase, it also computes the other analyses we discussed in the Web site analysis section. We store the results the analyzer produces in files.

The viewer provides a graphical user interface to display the output of structural and history analysis. Users select a site version with a colored button menu on the left side of the interface. When users select a button, a colored graph showing the history appears in the right part of the GUI. The GUI supports a rich set of navigation and query facilities including zoom, search, focus, and HTML code display. The facilities for focusing on and searching a node are useful when the visualized graphs are large. By exploiting the focusing facility, it’s possible to focus the view on a selected node and to specify the number of upward and downward levels the GUI should display—that is, the focus depth. If the analyzed Web site is organized into subdirectories, it’s possible to visualize its system view, where every subdirectory is a box (see Figure 4). ReWeb also permits expanding the elements of the system view into the enclosed pages. Another feature is a report containing simple metrics at the site level, such as the number of HTML pages, number of links, lines of code, and error links, or links leading to nonexistent pages in the same host.

We wrote the Web spider and analyzer in Java and based the viewer on Dotty. (Dotty is a cus-
Report of Web site http://www.ubicum.it/
This site contains HTML+ pages: LEVEL ≥ 1
Total pages = 87
Total HTML pages = 58
Total links = 322
Total error links = 0
Total lines of code (LOC) = 3273

Case study
Here we illustrate the functionalities of ReWeb, its support for understanding and restructuring tasks, and its Web site visualization and user interaction facilities. We analyzed and restructured the 58-page http://www.ubicum.it/ Web site (an ISP) to improve navigability, solve some structural problems, and above all, make it easier to understand and maintain.

Understanding the current Web site
The process of understanding a Web site with ReWeb starts from analyzing the site’s overall, high-level organization. The system view is the most abstract view among those recovered by ReWeb. (Figure 4 shows the system view of http://www.ubicum.it/.) This site hosts an ISP and is decomposed into eight virtual sites—physically associated with different directories serving different clients—and a bin directory containing a program that computes the Web server statistics.

The betty and len directories aren’t reachable from the root of the site. Expanding the virtual sites into their structural views makes some patterns we introduced in the Web site analysis section apparent. Figure 5 shows the entire site, and we can see that the virtual site piscini matches the pattern diamond; tigullio matches a tree; rigor contains a large full connectivity region; and madmaxpub, marco, and merlo’s structure doesn’t agree with any known pattern. The virtual site betty matches a tree while len resembles a tree (two edges violate it).

Figure 6 gives a partial site report. The site doesn’t have pages with forms and applets, but contains pages decomposed into frames. Two pages are implemented with a scripting language (JavaScript). According to our taxonomy, a Web site with these characteristics is at level ≥ 1.
The availability of information from the server side would let us decide among levels. Because we could access only client-side information, the same exercise was more difficult. We started from the two JavaScript pages, possibly containing input fields and user selections. The ReWeb report helped us identify them. The site’s history indicated the dynamic pages—pages that change their color every day are likely to be dynamic.

Because the two pages containing scripts don’t collect user input, no data communication occurs from the client to the server. ReWeb downloaded the site every day for more than six months. During that time, the page `bin/stat` changed its color every day in the history view, the page `madmaxpub/madmaxpub.htm` changed once on 16 Dec. 1999, and the other pages never changed (see Figure 7). Page `bin/stat` is a dynamic page that computes and visualizes the Web server statistics, with no client–server communication. Therefore, this site doesn’t contain pages that communicate with the server but contains a dynamic page. According to our taxonomy, it belongs to level 2.

**Collecting change requirements**

We examined the internal organization of http://www.ubicum.it/ and the way it supports navigation. We use the label CR to indicate the change requirements we collected (see Figure 8 for a complete list).

By looking at the directories `marco` and `merlo`, we can note a potential problem in the site’s system view (see Figure 4). Users can’t reach the pages in `marco`, a personal site, from the root without passing through the virtual site `madmaxpub`. A better site structure would position `marco`’s directory at the same level as the others. We can make an analogous argument for the directory `merlo`. On the other hand, users can’t reach `betty` and `len` from the root of the site. CR 1 addresses this problem.

Also, the root of this site doesn’t contain a search facility, which we suggest in CR 2. Searching a target virtual site in an ISP is a useful facility, especially if the provider manages a lot of clients.

Figure 7 shows the decomposition into frames of the virtual site `madmaxpub`. Page `madmaxpub/index.html` (the main page) is divided into two frames with identifiers `a` and `b`, and frame `a` is used as a menu to force pages to load into the other frame. The page `madmaxpub/framel.html` fails to divide itself in two frames, because the edge from the second frame is missing. When users load the page `madmaxpub/index.html` into a browser, the page loaded in frame `b`, `madmaxpub/framel.html`, remains gray so they can’t reach `madmaxpub/frame.html`. In the HTML code of the page `madmaxpub/framel.html`, a syntactic error exists: the `FRAMESET` tag is opened but not closed (see CR 3).

The virtual site `rigor`’s structural view (see the respective portion in Figure 5) is, at first glance, very complex. The site has two nodes of type `FRAME` and the connectivity among pages is high. Many edges are dashed and labeled with `main`. We further investigated the reason for the high connectivity. Every page contains two similar menus, with more or less the same links. One is

---

**Figure 7. The history view of the virtual Web site madmaxpub on 3 Feb. 2000.**

**Figure 8. Change requirements for http://www.ubicum.it/**
completely in HTML and the other one uses JavaScript buttons. A reasonable conjecture is that the designer implemented this virtual site in pure HTML the first time and added JavaScript menus (pages rigor/sommario.htm and rigor/som-cro.htm) and frames later, after a restructuring operation, thus in some cases doubling the links among pages. CR 4 gives the change requirement for this problem.

Next, we considered the results of the structure analyses. Computing the reaching frames revealed a major structural problem with the entire site. Pages in the virtual sites piscini, tigullio, marco, and merlo and the site’s main page can appear in the virtual site madmaxpub’s frame b. In fact, frame b is present in their reaching frame sets, making the multiple Web pages active and open in different frames inconsistent. Figure 9 shows an example of this problem. The madmaxpub navigation menu, which the site’s main page generates when selecting the virtual site madmaxpub, is displayed concurrently with the main page itself. This creates a potentially dangerous recursive loop, making the index (left) and informative (right) frames not align. A similar problem occurs when users load the madmaxpub navigation menu together with any page from the virtual sites piscini, tigullio, marco, and merlo. The menu on the left doesn’t index the information displayed on the right. CR 5 addresses this problem.

A similar frame problem occurs in the virtual site rigor. This virtual site exploits two different menus to access internal pages. A menu loaded into the frame indice can coexist with a page associated with the other menu. CR 6 requires the menu and the information to be consistent.

Computing the shortest path for each node highlights a possible improvement. Pages merlo/germania.htm and merlo/inghilterra.htm have a shortest path equal to 4. CR 7 addresses this problem.

As we pointed out before, the page madmax-pub/madmaxpub.htm changed on 16 Dec. 1999. The only difference is a new paragraph announcing a sale. Computing the dominators highlights that users can’t reach this important information without visiting the page madmaxpub/madmax-pub.htm, which isn’t a dominator of the entire virtual site. See CR 8 for the corresponding change requirement.

The virtual Web site rigor resembles a full connectivity pattern. According to the ReWeb
facility computing the strongly connected components, this virtual site consists of 19 nodes, 15 of which are strongly connected. Each of the 15 pages contains two menus providing access to the other pages, thus requiring the intervention we described in CR 4.

Restructuring

Figure 10 presents the system view of the restructured http://www.ubicum.it/ site. In accordance with CR 1, we added new links from the root to the virtual sites marco, merlo, betty, and len. In this way, users can reach the virtual sites marco, merlo, betty, and len directly from the root.

We also added a form to the site’s root that invokes a Perl script (search.pl), running on the server, in accordance with CR 2. Consequently, the change transformed http://www.ubicum.it/ into a level 3 dynamic site. The new pages let users search the virtual sites by account. After restructuring, page index.html has a new link to the dynamic page search.pl (see Figure 11).

We fixed the syntactic error in the page madmaxpub/frame1.html, as CR 3 required. In the new version, when the virtual site madmaxpub’s main page is rendered, the page madmaxpub/index.html correctly divides into two frames: madmaxpub/elenco.html, a menu that forces the pages to load into the other frame, and madmaxpub/frame.html, the site’s presentation page.

As we stated in CR 4, we simplified and improved the navigability in the virtual site rigor. Only one menu (the one implemented with JavaScript buttons), equivalent to the two menus present in the old version of the virtual site rigor, remained. By comparing Figure 12 (next page), which represents the entire restructured site’s structural view (without dynamic pages and with piscini, len, tigullio, merlo, and betty not expanded for space reasons) with Figure 5, the clearer and simpler organization of the virtual site rigor is apparent and is expected to make the site easier to understand and maintain.

In accordance with CR 5, we solved the problems associated with frames in the virtual site madmaxpub. In the virtual site’s new version, the menu loaded into the left frame and information displayed on the right frame are always consistent. Computing the reaching frames confirms this. Our restructuring intervention consisted of changing the targets of some HREF commands. In the site’s previous version, the edges now labeled with parent weren’t dashed or labeled—that is, users could reach the pages this hyperlink pointed to from pages inside frame b, and they were loaded into the same frame instead of the entire browser window (see the related portion of Figure 12).

We performed a similar intervention for the virtual site rigor. As for madmaxpub, the new version doesn’t present inconsistencies between pages concurrently displayed in different frames. Computing the reaching frames confirms it, fully addressing CR 6.

Adding new links from the root to the virtual sites marco, merlo, betty, and len diminished the shortest path of the pages merlo/germania.htm and merlo/inghilterra.htm to 2. No page in the entire site has a shortest path greater than 3, which addresses CR 7.

We moved the virtual site madmaxpub’s sale announcement to the page madmaxpub/frame.html, which is the dominator of all other pages. In the restructured version, it’s impossible to navigate in the madmaxpub site without encountering this important announcement, fulfilling CR 8.

Lessons learned

Analyzing user interactions with ReWeb helped us learn some lessons about Web site understanding and restructuring:

```
Input: key
search.pl
Type: dynamic
Use: key

piscini/index.html  Key=piscini
ligullio/index.html  Key=tigullio
madmaxpub/index.html  Key=madmaxpub
marco/index.html  Key=marco
merlo/index.html  Key=merlo
rigor/index.html  Key=rigor
len/index.html  Key=len
betty/index.html  Key=betty

Figure 11. Portion of the http://www.ubicum.it/ restructured site’s structural view.
```
During the understanding process, the system view is a good starting point for subsequent, more fine-grained analyses. This view, besides providing an overall mental model of the site, helps us recognize some structural problems at a high level.

A typical iteration in the understanding process with ReWeb includes computing the system view, building a high-level mental model of the site, expanding the virtual sites in the system view, using the structural and history analyses, using the report, and rendering some of the site’s pages on the browser. Users need to execute this process, or part of it, several times until they achieve a satisfactory knowledge about the target site.

Users most frequently used the structural view, because it gives a representation of the site in which we can understand the navigation constraints, the page decomposition into frames, and the transmission of data to the server. This view helped us recognize some structural problems at the navigational and organizational level.

A Web site’s history view allows an easy identification of the changes and of their impact on the structure. We can also use this analysis to identify dynamic pages.

The report, partially shown in Figure 6, provides hints for understanding the dimension of the selected Web site and the technology it uses, identifying dynamic pages, and revealing error links.

The reaching frames analysis highlights undesirable reaching frames. An example is when users can load a page where they shouldn’t be able to.

The dominator analysis can reveal some navigation problems. When developers must enforce
traversing a given page (for example, because it contains an important announcement), every node’s dominator set should contain it.

- The shortest path from the root to each page can indicate potential search and navigation problems.

- Strongly connected components containing many nodes reveal a cyclical navigation style.

- All analyses and views are useful even after restructuring. In fact, applying them to the restructured Web site verifies the change requirements’ implementation.

**Conclusion**

Although not definitive, the case study we describe here confirms that an automatic or semi-automatic tool can help developers understand and maintain Web sites. High-level views, describing the overall site architecture, are very useful, and detailed analyses can help with a site’s enclosed subparts. Specifically, we based restructuring on the reaching frames, dominators, and shortest path analyses. In fact, they highlight structural and navigation problems before restructuring and their absence after the intervention.

We’ll devote future work to improving the ReWeb’s robustness, widening the spectrum of analyzable sites, and enriching its set of analyses and facilities. We’d also like to add abstraction techniques to support a high-level view of the site and to partially automate input selection in the presence of dynamic pages.

**References**


**Filippo Ricca** is a researcher in the software engineering group at the ITC-irst (Center for Scientific and Technological Research) in Trento, Italy, and a PhD student in the Computer Science Department at the University of Genova, Italy. He has a degree in computer science from the University of Genova. His research interests include Web application analysis and Web testing.

**Paolo Tonella** is a full-time researcher in the software engineering group at the ITC-irst (Center for Scientific and Technological Research) in Trento, Italy. He has a laurea degree cum laude in electronics engineering and a PhD in software engineering from the University of Padua, Italy. His current research interests include reverse engineering, object-oriented programming, Web applications, and static code analysis.