Four Web Tools to Support Academic Design Activities

Filippo A. Salustri, PhD, PEng
Ryerson University
salustri@ryerson.ca

Abstract

In this paper, the author describes four Web-based tools currently being developed to support academic design activities (teaching, collaboration, and research). The author proposes that CDEN should adopt tools such as those described as building blocks for a more vital and active organisation.

1. Introduction

The Web is quickly becoming a fundamental element of all manner of activities including academics. Everything ranging from student registration, to course delivery is being augmented with Web tools. In many cases, these tools are based on monolithic, proprietary, commercial software systems (e.g., Blackboard) that are difficult to maintain, costly, and inflexible. Universities are usually limited by the corporate mandate of such companies and rarely have the opportunity to customise these large software systems as much as they would like. The result is a burdensome exercise for academics, especially when compared to many other, low-cost, usually opensource solutions.

Specific details of this matter will be discussed with respect to one of the author’s tools, Xiki (see Section 3), but those arguments generally hold for many other types of software commonly used in academic settings.

The author believes that the academic community is large enough and has sufficient expertise to develop its own solutions. Most engineering faculties in Canada also have close ties to Schools of Computer Science at their respective Universities; these relationships afford access to professional expertise unparalleled in industry. Students, if carefully chosen, can show tremendous “ownership” for such projects, more so than one would expect from commercial organisations that have profit as a primary driver. Depending on circumstance, one can also envision easily that such projects could constitute work done for academic credit, at least in schools of computer science. Finally, students able to participate in such projects can come to feel far more connected to their educational institutions, which can in turn improve student morale.

To introduce some of the possibilities arising from a so-called DIY (Do It Yourself) solution, this paper will discuss four lightweight Web-based tools that could be quite useful to support teaching, collaboration, and research.

After a brief description of these tools, the matter of organising a cross-Canada initiative to develop Web-based tools will be discussed. CDEN could play a pivotal role in providing the infrastructure to help such an initiative.

2. SERF

The first tool is the Searchable Engineering Resource Facility (SERF), which is a database of web resources dealing with design and related disciplines. It grew out of the author’s own desire for a systematic yet flexible way to maintain a large collection of interesting Web links.

SERF has gone through a number of incarnations. The current one, version 9a, is implemented in Perl, on top of an opensource Web spider/indexer, HtDig.

Briefly, HtDig is a complete web indexing solution for relatively small collections (tens or hundreds of thousands) of Web pages. The HtDig spider is used to “crawl the Web”, collecting and indexing Web pages according to customisable rules. It can rank keywords based on how and where they occur (e.g., keywords found in page titles are more important than keywords found in body text). The HtDig search program uses the database developed by the spider to find Web pages that match the user-provided query, and supports a variety of options (e.g., Boolean searches). The interface to the search program is entirely customisable user-selectable options as well as conventional templates.

HtDig provides a generic way to index and search a heterogeneous and unclassified set of Web sites. It does not provide any way to classify those sites.

The Perl layer written by the author adds a facet-based classification system on top of HtDig. It does this by taking advantage of a unique property of the HtDig spider. Typically, a spider uses a list of
“starting URLs” to begin its search. For each URL to be indexed, the corresponding Web page is fetched and then analysed. HtDig provides a “hook” that allows an installation to run an external program in place of the usual simple fetching of Web content. The interface for this is such that the external program must accept a URL and must provide the corresponding Web content on the standard output. HtDig reads the standard output as if it were the result of a plain fetching operation.

In SERF, this external program fetches the content from the given URL and adds to it certain HTML META tags that contain classification information. The modified content is printed out, and the HtDig spider then indexes the content, including the special META tags. The META tags are configured to have a very high weight. This allows a conventional query to the search engine to rank resources containing the requested META tags higher than other resources, ensuring that hits with the right tags rank higher than any other.

SERF also has a Perl front-end for HtDig’s search program. (Refer to Figure 1 for a sample screenshot.) The front end implements a facet-based classification scheme of the author’s invention. Facet-based classifications have several advantages over conventional systems (such as the Dewey Decimal System). Most importantly, facet-based systems have been demonstrated to be more usable, especially in cases where users are not sure what they are looking for (i.e. “browsing”), mostly because they tend to be flatter – less hierarchical – than other systems. Indeed, the facet system used in SERF is entirely flat.

SERF supports searches based only on keywords, or only on facets, or on a combination of the two. Search results are ranked and paginated much as one would expect from commercial systems.

One of the specific issues the author sought to address with SERF was to capture only key Web resources, rather than indiscriminately indexing every URL. In order to control the size SERF’s database and help ensure that only the most significant Web resources are indexed, the author uses only a semi-automated method of finding new sites.

New sites can be contributed to SERF by anyone via a separate script. Each contributed URL must be vetted by an “editor.” The author is currently the only editor of SERF, but the software can accommodate others. During the vetting stage, done with yet another script, the editor visits the contributed URL. If the URL is suitable, the editor selects the facets that best describe the resource (i.e. the editor classifies the resource). The vetting script also provides for the editor a list of all links in the contributed Web resource. Using checkboxes, the editor selects links that are also worth considering. These links are added to the list of URLs to be vetted in due course by the editor(s). This approach is based on that originated in the original version of Yahoo and is the current approach at the Open Directory Project.

Recently, the author re-implemented the facet structure, which necessitated rebuilding the database from scratch. As of this writing, there are over 250 resources listed in SERF and over 10,000 contributed links waiting to be vetted.

The author’s experience suggests that generally it takes an editor accustomed to the facets and their definitions less than five minutes to vet a URL. If 10 editors could spend just one hour per week vetting URLs, all 10,000 contributed links would be treated within two years.

SERF itself is less than 1,000 lines of Perl, which makes it a small and easy to maintain. Still, it provides added value to HtDig that many users (including students and other design instructors and researchers) have acknowledged as beneficial.

The greatest shortcoming of the current implementation of SERF is that changes to the facets must be done by changing the script itself. It would be much better if there were a Web interface to allow this kind of maintenance. This is planned for the next major release.

3. CORC

As part of the author’s efforts to discover better ways of organising design resources, he supports a second tool, called the CDEN Online Resource Centre (CORC). This tool uses a public domain package called Gossamer Links that implements a Yahoo-like directory structure entirely in Perl. (The package is no longer freely available since its makers have turned it into a commercial system, but since it is written in Perl, the source is available to the author.) CORC does not index web resources but rather depends on text descriptions entered by site “editors.” Users can search through the database, but searching is not as reliable in CORC as it is in SERF because CORC does not index the actual resources; this makes the role of “editors” far more important (and onerous) for CORC than for SERF.

There is an extensive administrative facility intended to support quick and easy maintenance of the database. The system automatically tracks new additions to the database, updated entries, and popular sites. However, since CORC does not crawl the Web, the author notes that it takes longer to create a new entry in CORC than in SERF. This is because more information must be provided manually and because special care must be taken by the editor to ensure that
all appropriate keywords are mentioned in the text summary for a resource.

On the other hand, CORC does have a significant advantage over SERF in that SERF can (currently) only index HTML files. There are extensions for HtDig for indexing other file types (e.g. PDF) but they have not yet been installed. Since CORC relies on human editors to generate the entry for a Web resource, it does not suffer this shortcoming.

As opposed to SERF’s flat system of classification facets, CORC has a hierarchy of categories and sub-categories, as does Yahoo. A sample screenshot of CORC is given in Figure 2.

There are two important philosophical differences between SERF and CORC. These differences result in each tool containing a substantively different set of Web resources.

Firstly, CORC contains resources relating directly and only to design engineering, while SERF relates to design in a broader sense (including software design, industrial design, graphic design, etc.) as well as certain ancillary fields such as formal systems and AI. This has to do with the design goals for the systems set by the author, and not with the inherent nature of the software itself.

Secondly, because SERF indexes actual Web pages whereas CORC does not, it is natural to add only properly annotated “root” pages to CORC, but “leaf” pages to SERF.

For example, the author maintains an extensive web site describing his research. In SERF, it makes sense to include all “leaf” web pages that describe actual research projects because SERF will automatically index all keywords with no prompting by the editor. However, in CORC, it makes more sense to list only the root page and devise a description that contains sufficient keywords to attract the visiting user who is browsing the hierarchy of resources.

Thus, CORC and SERF provide two significantly different ways to organise Web resources. It is unclear at this time which approach is better – indeed, it is not yet clear how “better” is defined in this case. However, by offering and maintaining both services, the author hopes to accumulate enough data to make such a decision in the future. Indeed, the “best” solution may be to write a CORC-like hierarchical system that hooks into HtDig for its search capacity.

4. Xiki

The third tool, a Wiki, is actually more of a platform for a number of tools. A Wiki, originally invented by Ward Cunningham [1] is a light-weight, web-based collaboration tool that permits anyone to edit Web content directly using simple HTML forms and a special, simple notation called Wiki Shorthand.

Wikis are a kind of simple content management system (CMS) that separate Web page layout from logical structure and content. Layout is defined by the Wiki itself, while users provide all the content. Since Wikis require only a conventional Web browser, they can be accessed from virtually anywhere using existent infrastructure.

With respect to other Web-based and Internet-based collaboration tools, Wikis are more powerful than simple email list servers, but less powerful than “full-blown” content management systems like Blackboard.

First, email and news groups are relatively unstructured systems, and therefore cheap and easy to operate and maintain. However, these systems have drawbacks. A common problem in email-based systems (which includes news groups) is that a discussion consisting of a collection of messages can “drift” off the topic used in the subject line. This means that content becomes disconnected from the principal source of keywords for both human users and automated search engines. Emails will therefore be misclassified and information will be effectively lost.

Second, errors in email messages are preserved in email archives. Even if a correction is posted, there is no way to “go back” and correct the original error. This means that users may miss the correction and use only the erroneous information.

Third, email/news archives are notoriously difficult to search because of the lack of structure in the stored messages. One may search only through email headers, but because of the two problems noted above, the quality of the results of such searches is suspect.

Full-blown CMSs, on the other hand, typically impose significant structure on the information they store, and provide a very rich functionality. However, these systems have drawbacks of their own.

CMSs also range in size and power. CMSs range from very small, opensource systems – such as WebApp – that are written only in Perl, to monolithic commercial packages like Blackboard. No matter what the size and scope, however, the system provides a certain fixed set of functions (e.g. calendaring, discussion forums, announcements, personal/private areas, etc.) beyond which particular installations cannot easily extend if at all. Furthermore, the templates used to control the look and feel of these sites are rarely well constructed for particular installation because they try to “do everything for everyone” and end up pleasing no one instead.
Wikis fit between these two extremes. They provide greater structure and control than do email lists, but do so with much more flexibility that full CMSs.

Wikis, of course, have their own disadvantages. Implementations designed to support a very large audience (e.g. Wikipedia, the largest of all current Wikis) depend on relatively large infrastructure packages (usually PHP and MySQL). Smaller implementations, usually written in Perl, such as TWiki, can be quite slow. However, small to medium sized communities that are satisfied by relatively few functions but significant flexibility and ease of use, will find Wikis generally ideal.

There are dozens of different Wiki implementations currently available, nearly all of them opensource-based. They range from very simple single scripts less than 100 lines of Perl long, to large multi-script systems that are built on top of MySQL database servers. There is even a company (jotsot.com) that offers Wiki functionality as a Web service. Detailed study of existent Wikis and their advantages and disadvantages in engineering design are beyond the scope of the current paper. Readers interested in further details may consult [2].

Wikis have four key characteristics that distinguish them from other classes of collaboration software.

First, a Wiki allows users to edit content. This is almost always done by providing a link on each page that downloads the page content via HTML forms and CGI scripts. A textarea is used contain the content, described in a shorthand notation (described below), and conventional buttons that allow the content to be saved via CGI.

Some implementations provide varying degrees of user authentication and protection. However, Wikis are generally intended to be “wide-open” resources. For example, anyone at all can edit a page on Wikipedia (which currently contains over 500,000 pages).

Second, Wikis provide a shorthand notation used when editing content that is very simple to learn and intuitive to use. For example, in order to make a heading, one places a number of = signs at the start of a line, followed by the heading text; the number of = signs is the “level” of the heading; and *bold text* is rendered as bold text. This shorthand is designed to make the actual content source as easy to read and manipulate as possible within the confines of based HTML. We note, however, that there is no standard for this shorthand and different Wiki implementations tend to vary at least some parts of the notation.

Third, Wikis are able to detect certain syntactic patterns as names of other Wiki pages and automatically create links at browse-time to those pages. This means that linking Wiki pages together is extremely simple.

Fourth, version management of content is automatic, usually done via either RCS or CVS, which are both opensource systems for version control. This means that every Wiki page has a revision history, that saving changes to a page triggers a version update, and that various operations like finding the differences between versions of a page are universally and uniformly available.

There are many other very useful features that distinguish Wikis. For example, all Wikis support backlinks – the ability to automatically generate a list of pages that refer to the current page. Refer to [1] for more about typical Wiki operations.

The author has studied Wikis for over a year and has determined that (a) they are quite suitable as vehicles to aid academics involved in design, but (b) there is no existent Wiki particularly well suited to these needs, especially those of the author’s own research. Therefore, the author has undertaken to develop his own Wiki, called Xiki, to address these needs. In particular, the key requirements of Xiki are:

1. High flexibility.
2. A shorthand notation that is easily understood and intuitive to design academics who may not be particularly computer literate.
3. Very low maintenance.
4. Extensible, to accommodate unforeseen needs.

While the details of Xiki are not pertinent to the current discussion, it is important to note that Xiki does implement authentication – only registered users can edit content, and restrictions can be placed on access to groups of pages via user IDs and user groups. This is done not to artificially limit access to Xiki, but rather to help ensure that (a) all content is properly attributed, and (b) users take responsibility for their actions.

Xiki is accessed at http://deed.ryerson.ca/x.

The details of Xiki itself are available elsewhere [2]. Here, however, we are more interested in describing its use in academic settings. The author is still implementing the registration process to allow people other than the author to edit content, so the projects currently undertaken in Xiki are largely about the author’s own research. However, it should be evident that Xiki could be used easily in other situations.

The author uses Xiki to provide online lecture notes and related course material. All such material for the author’s design courses is gathered in one collection, or WikiWeb. Each page is tagged with
keywords that indicate the courses to which the page pertains. This allows students to get a full listing of every pertinent page for a given course, while allowing the author to develop highly integrated courseware that can be shared between courses.

Another project that the author maintains via Xiki is an online, annotated bibliography of articles relating to topics of research interest. There are currently over 1300 entries in the annotated bibliography. Xiki recognises the BibTeX “alpha” syntax (e.g. [Sal05b]) as a reference to an entry in the bibliography and automatically links the bibliographic entry to the citation. The author has found that the “alpha” style provides a useful mnemonic tool to help users remember citations. Xiki has a form-based system to create new entries in the bibliography so that the appropriate citation labels are automatically generated.

Once Xiki is opened to the community at large (expected by August 2005), any registered user will be able to add a bibliographic entry to the database.

The bibliographic functionality of Xiki could also be used to support open review processes for online journals. In a typical double blind reviewing processes, neither authors nor reviewers are aware of the others’ identity, and much of the reviewers’ commentaries are kept confidential. In open review, on the other hand, all information is made publicly available on the premise that such openness will promote more candid and meaningful discussion, lessen the responsibility on single reviewers, and protect against ethically suspect activities. Open reviewing has been gaining favour in a number of areas recently, including the Electronic Transactions on Artificial Intelligence; Nursing Research; the Journal of Interactive Media in Education; the Journal of Chemical Education; and the Turkish Journal of Neurological Sciences.

Xiki can help here. A separate WikiWeb could be set up for a conference or journal publication. Xiki’s bibliographic citation labelling system can be adapted to suit the needs of the particular publication source because it is defined in a plug-in that can be replaced at run-time. Each submission would have a separate page with a link to a downloadable version of the paper. Xiki’s authentication model can be used to allow only the paper’s authors and the editor to upload new versions of the paper. The page itself would take on the appearance and function of a discussion forum in which reviewers and the author exchange thoughts and ideas about the paper. New drafts can be uploaded and linked to the page. Eventually, when an editor determines that the review process has run its course, the final version of the paper can be moved to a “proceedings” area where final papers appear. No modifications are necessary to the Xiki code itself to support open review, unless a labelling system different from the default convention were used; all that would be required would be to customise the organisation of the open review web, which can be done simply by editing the appropriate Xiki pages.

A design dictionary is also being implemented in Xiki. The dictionary is intended to eventually capture in a descriptive way all the various senses of key terms in design. The approach taken to the design dictionary is similar to that of the Oxford English Dictionary [3]; namely, to capture the “living language” of design by noting samples of the usage of terms, and building definitions from those samples collaboratively. Only a few terms exist in the dictionary at this time, because the system is still under development. However, the author hopes to significantly increase the content in the coming year.

Wikis have classically been used to implement pattern languages, which are collections of solutions to classes of problems presented as carefully written natural language descriptions. The pattern language approach was pioneered by the architect C. Alexander [4]; it has been very effectively used in software design, architecture, and process management. However, patterns have yet to be applied to more conventional design settings (e.g. product or mechanical design). The author is using Xiki to experiment with patterns for mechanical engineering. For example, we are currently starting a project to develop an online textbook for introductory mechanics based on the pattern language approach. Further details of this project are available in [5].

Finally, the author is also starting a design collaboratory, a WikiWeb intended to allow design researchers to exchange information and discuss topics of interest, with the goal of advancing the our overall understanding of design. This is the most “wide open” project; virtually any topic is suitable. Individuals with particular interest in topic pages will be expected to assume leadership roles in maintaining the page content, facilitating discussions, and summarising content as the need arises. The author is currently developing a set of patterns to assist collaboratory participants to keep tidy Xiki pages without restricting the openness and flexibility of the site.

5. A Concept Mapping Tool

The final tool the author is developing is a simple concept mapping tool. This is part of a larger project intended to develop diagrammatic visualisation tools for qualitative information typical in the early stages of design projects.

Concept maps, originated by Novak [6], are simple diagrams that have been shown to be able to
capture significant collections of information in a form that is easily absorbed by others. In a concept map, each concept is shown as a node (typically a box with a text string in it). Relationships between nodes are shown as labelled arrows drawn between nodes, where the label indicates the nature of the relation. Formally, a concept map can be thought of as either a constraint graph [REF] or a bipartite hypergraph. Perhaps the best concept mapping tool currently available is the opensource CmapTools package [7].

The author is using Tcl/Tk to develop a simple concept mapping tool that will be used as a baseline upon which more advanced functionality will be built. Tcl/Tk is a scripting language especially intended to prototype graphical user interfaces. The current experimental implementation, of admittedly limited functionality, can draw nodes as coloured boxes with containing editable text. The box reshapes itself automatically to accommodate the text as it is edited. Nodes can be moved around the screen arbitrarily. Links between nodes are shown as arrows that remain fixed to the nodes even when the nodes are being moved.

While this may seem like a minimal program, it is noted that:
(a) the author never used Tcl/Tk before;
(b) the author developed this program from a “standing stop” within 50 person-hours; and
(c) the program is only about 200 lines long.

Tcl/Tk has a plug-in to parse and manipulate XML files; XML will be used as a neutral format to save concept maps generated by the tool.

With recently secured funding from Materials and Manufacturing Ontario, the author has hired a student from Ryerson’s School of Computer Science to develop the program further over the summer. We fully expect to have a complete, though relatively simple, concept mapping tool operational by August 2005. A key feature of the tool will be ability to load run-time plug-ins that will implement geometric constraints on the layout of concept maps. This means we will be able to quickly devise and test different ways to arrange information diagrammatically.

Armed with the tool, the author’s graduate student will be able to experiment with diagrammatic visualisation, and to apply those results to the refinement of the tool.

Again, the key point here is not so much the tool itself, but the speed of development, the lack of substantial human resource requirements, and the robustness of the resulting software. That is, meaningful software can be developed quickly, easily, without significant human resources. Students are also extremely able to contribute meaningfully to such undertakings.

6. Discussion

It has taken the author years to develop these tools, but only because he has not had enough time to devote exclusively to them. In actual person-hours, the work invested in these tools is no more than one person-year to develop everything that was discussed above, while also learning Perl, Tcl/Tk, and how to use the packages like HtDig upon which the author’s software depends.

It is easy to envision that only a few competent programmers could easily advance all these systems dramatically, given the opportunity.

There are many other opportunities in this area as well. For example, the US Department of energy makes available a series of relatively small tools (some of them implemented in nothing more than Excel spreadsheets) for assessing industrial components like piping systems and the energy usage and waste production of industrial installations (http://www.oit.doe.gov/bestpractices/software_tools.shtml). It is evident to the author that we could develop similar tools for use by our students and researchers under a system like the one being proposed here.

Many members of the CDEN academic community (a) have access to students who are good programmers and (b) maintain good relations with colleagues Schools of Computer Science of their Universities. If each CDEN member-University could find one reliable student programmer, we would have a development team of over 30 persons, far more than many successful opensource projects.

There are many programs, such as Ontario’s Work/Study program, that provide funding to employ students for research and development activities. In combination with possible funding through CDEN, this kind of work can help alleviate the financial burden faced by students.

The computer skills and experience gained by students can also be very useful once they graduate and enter the job market.

The author recalls his undergraduate industry internship at the DeHavilland Aircraft Company well and to this day feels pride whenever he sees a DASH-8 aircraft, knowing that he contributed in some small way to it. This same sense of pride and involvement could be instilled in at least some of our students by having them contribute to the development of software that all the Universities could share.

These activities could also conceivably be given academic credit under the proper supervision and conditions. For example, designing and implementing layout algorithms for the concept mapping tool in Section 5 would easily constitute an appropriate undergraduate thesis in computer science; and
performing a proper content analysis and facet classification development would be a good collaborative project for students in engineering (as content “expects”) and library sciences.

It is also important to bear in mind the growth model of opensource software, which is different from that of conventional commercial software. Opensource software projects rarely attract much attention initially. Perl, Mosaic (the Web browser on which Netscape was based), Apache, Linux, and many other very successful opensource projects all languished in obscurity for years—sometimes decades—before suddenly “catching fire” and becoming widely accepted. A preliminary implementation (though often at a version number no higher than 0.1) is another key feature of all successful opensource projects. People will not consider opensource projects as viable until some small group have invested the time and shown the dedication to the project to start creating software.

This means that one cannot wait for acceptance of an idea or plan for opensource software. One must actually pursue it with the expectation that acceptance and adoption will come later. Therefore, all that is needed is to find a small group of academics (no more than five) willing to start the project by finding students able to program, and to work with them to development software.

The development of software by academia for academia could also partially free Universities from the restrictions arising from the use of commercial software systems. The opensource model is ideally suited to academic settings, and the development of software that could provide a flexible infrastructure for sharing design knowledge and collaborating at a distance on design projects falls squarely within its purview.

Let us not forget also that software design is in fact a kind of engineering design. One might then envision a small group of students, each under local supervision, working either for credit, or in a position of summer employment, or even part-time employment during the academic year. Some students might even participate purely on a volunteer basis, either because they wish to gain experience in such technologies or just to contribute. These students would collaborate using well-understood and supported Web-based tools such as SourceForge (http://www.sourceforge.net) to develop tools for use in academic settings. The direction of the team as a whole could be set and managed by a small working group (two or three people should be enough) of CDEN members. The software would be designed and packaged to run on the computers of any CDEN node/university. CDEN would retain control of the development (since CDEN member schools would constitute the primary user community).

CDEN has an opportunity here, to facilitate the creation of a national network of opensource developers of software for academics. It is hoped that the tools introduced in this paper demonstrate that one need not rely on large, costly, and overly constrained commercial software to facilitate engineering design academic work.

References

Figure 1. Sample screen from SERF in the Safari Web browser.
Figure 2. Sample screen from CORC in the Safari Web browser.