Fall 2006

Development of Dynamically-Generated Pages On a Website

Jodi Wagner
Regis University

Regis University
School for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Development of Dynamically-Generated Pages on a Website

Jodi Wagner

Regis University

School for Professional Studies

Master of Science in Computer Information Technology

Regis University

School for Professional Studies Graduate Programs

MSCIT Program

Graduate Programs Final Project/Thesis

Abstract

The purpose of this paper is to examine the project management techniques involved in developing dynamically-generated pages for an existing website. The dynamically-generated pages will allow the editors of a website to add and remove content from pages without directly editing the pages themselves. The project will benefit editors, who will be able to spend more time on creating content for the website, which will lead to benefits for site visitors as well, in the form of uniformly designed pages and more content for the site.

Table of Contents

List of Figures

1 Introduction and Background

1.1 Thesis

The purpose of this project was to make it possible to replace a website's existing static HTML pages with dynamically-updated pages created from data in a database. This project reduced the time spent on updating pages and helped decrease problems common to index-type web pages, such as broken links and out-of-date information. This project was a part of a larger website quality improvement and efficiency project.

1.2 Background on the Project

The website in question, HP Dev Resource Central (http://devresource.hp.com), provided resources to help software developers who write software that interacts with HP's software offerings. The resources on the site are free of charge, and cover a wide range of topic areas. These resources include white papers; software downloads; a newsletter; forums for developers to discuss various topics; technical tips; information about upcoming events; and other topics of interest to software developers. Site offerings generally fit into two categories: content pages that contained information users commonly need to get their jobs done, and index pages that logically group links to the leaf-level pages. A user may navigate through one or two index-type pages in order to reach the content-level pages; for example, if a user were to bookmark the homepage, it would then be necessary to click a link to an index page that lists links to pages that relate to a particular topic he or she is interested in. The topics of these index pages are generally a product or technology to which each offering applies. These product- and technology-specific index pages are called topic pages on the site. This makes it easy for users to find all the information pertinent to a single topic, and to ignore resources that

are not of interest.

The lists of links on these topic pages were originally static HTML, which was edited by hand every time a link was added, removed, or updated. Document metadata must be displayed for each link in order to help users make an informed decision about whether to visit the destination page. The document title, file format (if the format something other than HTML – PDF, for example), the document's date, and a short description are all helpful to users. Additionally, the index pages display the links by document title, a design decision based not on the usefulness of alphabetic ordering, but on the need for a simple paradigm that users can grasp quickly. Since some documents are relevant to more than one topic, it was often necessary to edit more than one topic index page when adding a new document to the site.

When the site was small and the number of links on each topic page was low, coding these pages statically was not overly time-consuming. As the number of topics and resources on the site grew, so did the time required for each edit. The effort involved in editing and testing all of this information would eventually make it impossible to continue making timely updates without adding more employees.

The process for doing manual updates to the static topic pages was simple. Since the information that needs to be displayed about that resource is generally the same on every topic page, an editor could cut and paste the same HTML code from one page into the others that link to the document. Editors would then publish the new resource and each of the topic pages they modified. Updating and publishing index files one by one is time-consuming, and leaves a lot of opportunity for error on the part of the site editors. Furthermore, any time there was an update to any of the information about a resource that was displayed on a topic page, editors had to find out which topics link to the resource

and make the same change on each of those topic pages.

1.3 Customer Need for Project

The primary customers of this project were the site editors. The goal of this project was to decrease the time that site editors spent on site maintenance, allowing them to focus on creating and formatting new content pages. It centralized document metadata into a database where it could be more easily found and quickly changed. Storing metadata in the database rather than on each topic page reduced the chances for error when adding or updating document information on multiple topic pages. Editors were also expected to be more likely to link to relevant documents from more than one topic page because of the lessened amount of time required to do so.

The secondary beneficiaries, the users of the site, benefited from more timely and accurate updates to the topic pages; and the increased cross-linking made it more likely that all relevant information would be linked to from each topic page. They also received new content more frequently, since editors will have more time to work on that.

1.4 Business Reasons for Project

Business reasons for the project can be derived from the customer needs: the project resulted in increased efficiency for editors, more accurate topic pages, and more content. Users' experience on the site is directly related to how much useful, relevant content customers can find, as well as their ability to find all relevant content easily.

This project also made the HTML on the site more centralized, resulting in a more standard look and feel (important to customers who learn visually and therefore expect to find things in a certain way) and making it simpler to update the look and feel of the site or fix errors, since it can be done in a central location rather than by editing each page

individually.

1.5 How This Project Relates to the Field Overall

Every large website has its content in a database. The data itself may or may not be stored in a database (the files that contain the content may be stored on the file system), but at least the metadata about that content must be stored in a database to support the dynamic display of data on index-type pages.

This project was the first step in making the website more cost-effective and more competitive. What makes the site unique is its content. There are many different types of documents, and they can be organized in several different ways: according to the task the developer is performing (compiling, debugging, porting, designing); according to document type (reference manual, case study, white paper, technical tip); according to operating system (UNIX, Windows); according to programming language (C, C++, Java); the list goes on. The types of documents, programming languages, tasks, operating system, and so on are not unique to the site, but the combination of them is ... and the organization of this diverse information presents a special challenge for the site infrastructure.

The long-term goal of the site is to make it possible for users to filter out the content that is irrelevant to them – even within the topic pages some content will not be relevant - and leave only the useful information. That would never be possible in a static world. A site that dynamic would be a major undertaking and require a complete site redesign, as well as diverting site editors from their normal content-creation and formatting tasks.

In order to be able to continue normal work while moving toward the future, it was be necessary to undertake small, incremental projects. The first step was just to get

the metadata into the database. One further option was to put the content itself in the database, but that would create another problem: the URLs of all the documents would change. Site statistics show that the home page is not the only entry point on the page; users of the site have bookmarked various pages, search engines have the current URLs in their databases, and other sites link to the index and content pages. Internet search engines are an important way that users find the site and individual pages of content; if all of the site's URLs change, it would be a major inconvenience to users to re-find the documents they need. Good Web citizenship means avoiding link rot (the dead ends that occur when pages move or are removed). It might be possible to resolve the problem by redirecting each request to the right page, but maintaining those redirects would not have a good return on investment. Redirects like this are not uncommon when sites are redesigned, but the goal of the project is to improve the site in small, achievable increments rather than to drastically overhaul the site.

2 Project History

2.1 How the Project Began

This project was initiated in response to concerns about potential problems for the site's two main user groups: site editors and site users.

First, site editors were spending too much time on updating the site topic pages, and are introducing inconsistencies in the page layout on some topic pages and in the way various resources were being linked to and described. Some of the inconsistencies were intentional – intended to improve the user experience – and some were accidents resulting from copy-and-paste errors, omissions of content, and inability or unwillingness to test rigorously enough. When editors spend too much time simply doing updates to the topic pages, it detracts from the time they have available to spend getting content added

to the site.

Second, site users may not have benefited from the flexibility that hand-coded pages offer: new layout ideas could not be efficiently transferred to other pages, and the resulting inconsistencies in page layout were a usability issue. For pages that have the same purpose – listing documents related to a particular topic, in this case – users should not have to learn a new page layout every time they go to a new page on the same site. There was some discussion about how each page's layout might need to be different simply because of the content mandated by its topic, but it was decided that customers would receive more benefit from consistent layouts; new design ideas and would be considered with their effects on *all* topic pages in mind.

2.2 Issues in Project Definition

The project boundaries were fairly clear. There is a set of index pages for product- and technology-related tasks that all have the same purpose and should all have the same structure. Since these pages are a repository for links to almost all the other information on the website, they were edited the most frequently of any pages on the site.

No outside (hardware or software support) assistance was required. A separate team provides and maintains the site infrastructure consisting of web servers, a database, and the hardware they run on, but was not involved in developing the application to display content on the Dev Resource Central website.

Using existing infrastructure for this project kept costs low. This dictated restricted infrastructure options for the project, but also reduced the amount of time spent on research. The infrastructure allowed use of any of the following:

- Apache Tomcat web server, which supports Java servlets and Java Server Pages (JSP) on either Microsoft Windows or UNIX servers

- Microsoft IIS web server, which supports Active Server Pages (ASP) on Microsoft Windows
- Apache, which supports Perl and various other scripting languages on UNIX servers
- An Oracle 9i database

The project was required to have as little effect as possible on the group that maintains the site infrastructure, since this project was not on that group's plan of record and would not be allowed to interfere with projects already in progress.

This project depended on a separate project to set up storage and publishing of metadata to the production web server database: the Content Management System (CMS) project. Requirements from this project fed into the CMS project, so timely communication between teams was essential. This interdependency dictated one fundamental requirement for the CMS project: the structure of the database needed to be modifiable at all times to allow for changes in the metadata to be displayed on the topic pages.

The initial deployment of the project was a proof-of-concept on one page, which meant manually entering into the database the data necessary for that page to display. Conversion of the rest of the site data was not part of the project upon which this project depended (the Content Management System project); however, a plan for the completion of data conversion and deployment of topic pages was a deliverable for this project.

One concern about the project was the lack of flexibility in layout of dynamically-generated pages, but flexibility leads to questions about inconsistencies in layout. Since the topic pages have evolved separately over time, it was necessary to arrive at a single page design that met the majority of the needs for all pages. Pages that did not fit into

that layout required additional display logic and were added to the project scope where possible.

No team member spent more than 25% of his or her time on this project, so that other work on the site did not fall behind. This was because the primary benefactors of this project – site editors – are not an external customer; since external customers are the primary focus, work that affects them directly and substantially gets priority.

The deployed project could not affect the other sites hosted on the same machines as Dev Resource Central. Potential areas of concern were: display performance, network bandwidth, processor and memory usage, and database performance and size. The development server needed to be as similar to the production environment as possible. The development server's operating system, web server, and database software needed to be the same as that of the production server in order to reduce unexpected problems during deployment. Final testing of the application occurred on a staging server that is nearly identical to the production server.

Of the three project management variables (scope, schedule, and cost), scope was the least flexible and schedule the most. Scope for the project was already pared down to the bare minimum: there are many other pages of the same style that could benefit from the same treatment. Cost was mostly a function of the investment in project members, and was therefore mostly fixed. Schedule could slip as necessary: completing the project was of most direct benefit to the editors of the site, and getting new content to HP's customers took precedence over improvements for site editors, even though that had indirect benefits for customers.

2.3 Comparison of Goals to Final Outcome of the Project

The schedule was wildly different at the end of the project than it was projected to

be at the beginning, due mostly to a six-month slip in publishing project upon which this project depended. Minor schedule slips when content projects took precedence also had a small effect on the schedule.

When deployed to the production server, the response time was substantially slower than it had been during development and test. The original plan to put only one dynamic page in place as a final test instead of immediately converting all topic pages from static to dynamic was a good one. In the staging environment, page display was adequate; load testing indicated that performance would be acceptable, and while there was a slightly longer delay in the page load time than for static pages, it was still less than a second: the page display was not quite as fast as the static page had been, but it was not enough to be annoying. Since the performance test passed, management approved deployment to the production servers. The first test on the production servers was a disappointment. The delay was magnified. The brief pause on the staging server was the result of the return trip between the web server and the database server, which are co-located in the same data center. Unfortunately, the database and the production servers were not co-located, and in production the user's browser window had time to go blank while waiting for the page to display. Due to the slow display, the lowest-volume, lowest-content topic page was chosen to be converted as a test page while investigations into the poor performance were underway. The project team then had to work to figure out why the production system (much bigger hardware than either the staging or development environments provided) was responding so slowly.

There were two possible explanations for the slow response: a query that took too long, network latency, or formatting (a code or processor/memory problem on the server). Benchmarking the formatting and query times separately in the JSP page showed

that the formatting took a reasonably short amount of time, and considerably less time than the database request. The next thing to do was to time all steps involved in making the query: opening the connection, performing the query, and closing the connection.

Testing in the database showed that the query ran in only a few milliseconds. That left the network. When the web server was co-located with the database server, as in the development environment, the performance was excellent. Unfortunately, the primary production server was half a continent away from the database server. At that point, the only way to solve the problem was to convince the team that provides the infrastructure that the primary server and database needed to be co-located. This turned out to be already in the plans, but not for a couple of months. No more topic pages in production could be converted until the servers were physically next to each other.

During this time, enhancements were added to the project to help speed up the connection in spite of the network latency issues. The initial implementation opened and closed a connection each time a database connection was needed, which is not very efficient. The solution was a database connection pool, in which a servlet that is loaded at server startup keeps a number of connections to the database open at all times, and doles them out to processes as they are needed.

The additional slip in schedule meant that the editors would continue to hand-code their pages when they should be spending their time on getting new content on the site; this was acceptable to management since the user experience in the production environment was unacceptable.

The project makes it easier for editors to change the content in the one topic page that uses the Java tag libraries that were deployed as part of this project. Some additional content has had its metadata set, which will reduce the work required to migrate content

when the project enters its second phase: deployment of all topic pages.

The project also benefited its sister project; usability problems in the content management system were found and fixed before the CMS was provided to users outside the core web team.

3 Research

3.1 Research & Analysis Methodology

Online research was foundational to the project. Web software that was likely to be a candidate for this project has most of its documentation online, and the software and associated documentation change so rapidly that it is impractical to get information from printed sources. The project team had experience with Vignette, a content management suite, while working on another incarnation of the site. That experience demonstrated that portal software would require extensive customization and a steep learning curve to become productive; it might be possible to overcome that curve, or at least to put it off, by hiring consultants familiar with the software to build the system, but the project's budget is too small to make that reasonable.

3.2 Research on Products

3.2.1 Web Portal Software

A solution using web portal software such as BEA WebLogic or Vignette StoryServer involves a large learning curve, since web portal software offers very specific components that then need to be customized to create a solution. They also require support and money expenditures greater than what the existing site allows for, and are frequently difficult to modify to the look and feel desired (or required, in this case, by the corporation).

Open source software is far less expensive, but presents other problems: a learning curve similar to that of the web portal software, a high level of complexity to get the resulting pages to look and feel like part of the site, and the support and documentation of open source software packages are often inadequate or inaccurate.

3.2.2 Database

The infrastructure team offers an Oracle database and system administrators, so the decision to go with that database was simple. This had some influence on what software and languages to use: they needed to have good support for Oracle. Fortunately, there is pretty broad support for the Oracle database, and it is a very well-known and stable commodity.

3.2.3 Programming Language

Programming languages and associated tools to be used would be directed by the web application server and database: the learning curve needed to be small; the language needed to be widely accepted and well-documented; and there needed to be stable, preexisting libraries for the web application server and database. For professional reasons, as well, it was important to use a popular language with a broad user group: working in a programming language that has only a small or shrinking user base would not be particularly useful for professional development.

3.3 Review of Research Deliverables

The project team knows the existing infrastructure, data, and the look and the feel of the site well. Because the problem to be solved was straightforward, the Apache Tomcat web application server seemed to be a good choice. The design would have to be extensible in the future, but Tomcat provided the fastest, most flexible, and most

inexpensive option available.

4 Project Scoping, Analysis, & Design

4.1 Project Scoping

The scoping methodology is reflected in the project scoping document. The project is suggested by a person and projects are then prioritized by the management team. Once the project is approved, a project lead is assigned to propose a schedule and defines the project, including what it is, what it is not, the people involved, and the schedule. Among the other things to be considered are whether the team that hosts the site will be involved or will need to be notified of the project. They need to be notified during application deployment, since they are responsible for server maintenance and application deployment. If that group is involved, then their freezes and schedules needed to be taken into consideration, and attendance in regularly-scheduled change review meetings would be necessary to go ahead with the project. The infrastructure team needed to review the effects of the application deployment and consider how the new application would affect the other sites that using the shared server infrastructure (web server software, hardware, and database).

Ideas for projects come from known customer needs. Project requests are posted to a project listing page, and management evaluates them on a weekly basis to determine the priority of each. During that evaluation, projects are ranked as high, medium, or low priority. The projects that are ranked high priority are then prioritized as 1 (top priority) 2 (medium priority) or 3 (lowest priority). When a project is ranked high priority and 1, the project is assigned to a project lead based on his or her area of expertise and other workload.

The lead is responsible for creating a project outline. The project outline is a one-

to two-page high-level description of the project, starting with the project objective.

The project objective, a one-sentence statement of purpose, gives direction to the rest of this document. In the case of this project, the project objective statement was:

> Using a streamlined publishing process, revise methodology to dynamically generate pages.

This clearly stated that the methodology to dynamically create pages on the site would be based on a streamlined publishing process (this was already in the works when the DPG project was initiated), and that a new process for generating dynamic web pages needed to be created (a previous project for dynamically generating pages had never been put into use). It also left changes to the publishing process outside the scope of this project, while acknowledging the dependency. Every section of this document helped to clarify the purpose of the project so that the scope of the work was clear to everyone from high-level managers to those doing the coding.

The Benefits section helped explain the reason for the project:

> Sets up environment for greater productivity and flexibility in the future. This is a companion project to the CMS Publishing Process project. When the new publishing process is implemented, the way dynamic pages are generated will have to change.

This set the context for the project: some of the infrastructure (the publishing process) was changing, which meant that the dynamic page generation functionality that was already in place - and that relied on certain facets of the publishing process - had to change.

The customers of this project were listed next:

> Users - will get a more consistent, accurate, and complete view of available

content

Content Contributors - will have the tools and capabilities to quickly enter content that can be assigned to appear on multiple pages in various locations (for example: in the main body, as a highlight, as a feature, etc.).

Listing the customers of the project was paramount: if no customers could be listed, there would be no reason to do the project. It also provided the perspective from which the test cases should be written.

The deliverables section came next; it is a fairly standard section in that many of the project deliverables remain the same from one project to the next. The deliverables section for this project was as follows:

User acceptance tests (standard)

Implementation plan (standard)

Support plan (standard)

Revised code for current dynamic pages

Data model updates

CMS updates corresponding to data model changes

The deliverables were listed at a fairly high level. The standard deliverables were listed first, followed by the items that were specific to this project. In this case, *Revised code* is one file (or set of files) that was deployed to the production server; *Data model updates* were changes that must be made to the existing database; and *CMS updates* were changes to the content editors' user interface so that they could enter data using the new data model.

The *Milestones* section was a simple enumeration of the project steps and their due dates:

21May03: Sponsor agreement [approval of project outline]

30May03: Analysis checkpoint [analysis complete]

10Jun03, 27Jun03: Design/Construction/Testing checkpoint(s) [check in, completion date]

11Jul03: Implementation checkpoint

18Jul03: Retrospective report complete

The last design, construction, and testing checkpoint was the completion date for those three phases; any other checkpoints were simply reminders to provide management with status reports and let them know what, if anything, is compromising project schedule; there was no formal meeting for projects that did not require interaction with other groups. Informal updates were made on a weekly basis in team meetings or during hallway chats. The project was expected to be in production, and the developers' work complete, on the implementation checkpoint date. It was then the project lead's responsibility to request information from all involved on what went well and what could go better for the next project.

The *People* section was a list of the people involved and their roles in the project. The full list of people was made up of the project lead, sponsor, team members and their respective areas of contribution, and reviewers, who needed to be kept informed on the progress of the project.

The *Boundaries* section simply lists the boundaries of the project:

Other than simple file moves, no involvement on the part of the infrastructure team is anticipated

Only content and data for this site will be affected by this project

A follow-on effort will be launched to address new dynamic pages; only topic

pages are included here.

This made clear which pages on the site would be affected by this project and the amount of project management that would be required (if other groups were involved or if the project were to affect data or content outside the website, project management time would be greater).

The *Assumptions and Risks* section outlined the assumptions foundational to the project:

> Although this project can be done in parallel with the Publishing Process project, it is dependent on the outcome of the new publishing process. Therefore, final testing and implementation must occur after implementation of the new publishing process.
>
> No team member will expend more than 25% of his/her time on this project
>
> Scope is the least flexible project parameter; resources are the most flexible.

This made clear the dependency of this project on another project and let management know how much time the project team would be spending. Most importantly, it made clear to the entire project team what trade-offs should be made when there was a conflict for time: schedule was the least-important component of the project, so slips were likely to be approved of.

The *Completion Criteria* section was a one-sentence statement describing how to know when the project is done:

> When topic pages are generated dynamically using the new publishing process, this project will be considered complete.

This was a clear, easy-to-measure criterion for completion; any task that did not advance the project toward this criterion was outside the scope of the project and should

not be undertaken.

Finally, a *Revision History* section appeared at the end of the document to summarize any major updates to the document; it provided a project history of sorts, although its main purpose was to make it easy to tell at a glance when, if, and how the document was changed.

Once all this information was documented, it was sent for review by email to those listed in the *People* section of the document. This check ensured that all those listed in the document were aware that they would potentially be helping on the project and gave them a chance to add any risks or other foundational information that management should be aware of. After this round of reviews was complete, the document was sent to the project sponsor(s), who then reviewed and signed off on the project. This process ensured that the project was the same as what was expected and that timing and resources for the project were appropriate.

The final deliverable for the scoping phase of the project management life cycle was the project tracking entry on the team's intranet site. This site made program management tracking easier by aggregating status information in one place. Although a view of all active projects was available, the single-project view was the one pertinent to this project: it tracked status (green for 'on track', yellow for 'there are concerns', and red for 'needs management attention to get back on track'); lifecycle state (analysis, design, and so on); and stored status reports, task lists, test plans, and all the other information necessary to track project status. When the sponsor approved the project outline, the project lead moved the project state to the next phase: Analysis.

4.2 Analysis

The Project scoping document that was the end result of the Scoping phase

provided the starting point for the analysis phase. The most pertinent section of the project outline document for this phase was the list of customers of this project. The list of customers provided a perspective from which to write the user acceptance tests that ultimately provided direction for the project.

The user acceptance tests were the primary outcome of the analysis phase. The tests described the things that users would be able to do once the project was complete. User acceptance tests were important because they ensured that the project team was thinking about customers and not about working on projects that would not help them, or adding unnecessary functionality. The user acceptance tests were labeled as Must (M) or Want (W). Tests labeled as *must* had to pass before the project could be deployed. Tests labeled *want* would be completed if time allowed.

During this phase, issues arose that needed to be resolved before the project was complete (in some cases) or before moving on to the next step in the project. An issues log document was created to track these issues, and was attached to the project's entry in the project management website. This document listed each issue concisely, as well as the date each issue was opened, its due date, and its resolution date. If the issue required more description, a more thorough discussion was entered at the end of the document.

Some of the issues that arose were brought to light when creating user acceptance tests for the project. The user acceptance tests provided a list of tasks that each user must be able to perform when the project is complete. A more detailed test plan could be created from the user acceptance tests as appropriate (for instance, a unit test plan may be created for each separate software module to ensure that it produces the technical results necessary, whether the results tested are performance boundaries, calculations, etc.), but this document was the ultimate test suite that must be executed. The user acceptance

tests brought issues to light because creating them forced the team members to put themselves in the place of the user rather than in the place of the doer; questions arose regarding how a user would behave in a given situation, whether it was technically feasible to meet a need that was expected to be desired by the user, and so on. The user acceptance tests were created by the project lead, but were reviewed by the entire team. Ideally, the team would have included a customer (Extreme Programming), but since there were not enough people to do that and because the editorial team itself was the customer base, the project team represented the customers.

Once the user acceptance tests were complete, they were used to determine what functionality was necessary. The list of functionality was then turned into a task list.

The business requirements document described the business justification for the project. Customer needs had already been documented, but even those could be justified (or removed or prioritized based on lack of business need). Business requirements were then derived based on the tasks: whether a database would be necessary; what sort of server hardware and software functionality would have the best support for the customer's needs; and what sort of support would be necessary from outside teams were all documented so the business could be aware of potential costs and risks before the project began.

The business process flowchart showed the interactions between the various business segments that were involved in the project. It helped the support group understand when they might be involved with the software that resulted from the project, and helped the deployment team understand who would provide the software and other information necessary for deployment. It also confirmed the approval process for deployment, which let the developers know the release criteria from a business

perspective.

The data requirements document showed the inputs and outputs of the software. As with most of the rest of the documentation in the planning portion of the project, this documentation helped provide boundaries and details during the design phase of the project. The flow diagram also showed the screens the user would see (without design elements; these screens simply have labels describing their general functions). Any reports that might be required or generated based on user or system outputs were listed here as well.

Security is always a concern for any website; however, there were no user security requirements for this project, so security audit requirements for this project are minimal. The largest concern for this project was keeping the database user name and password private. This meant simply storing that information in a secure area, which is standard practice.

The buy/build criteria for this project were straightforward. The project budget did not provide for large software purchases, so the budget was the least-flexible component of the resources available. Since the server infrastructure was provided by another group, the web application server that was already in place was the best choice, and the dynamic page application would be built using that. In previous web projects, dynamic web applications have required as much effort to customize using large content management systems as they do when they are built from scratch, so the lower expenditure on software would be just as efficient as buying a large web application server.

When the documentation for this phase was complete and reviewed by all responsible parties, the project sponsor gave buy-in for phase end – this was simply

business approval to move ahead with the project as documented to this point. The documents were kept up to date on the central team project website throughout, so the project sponsor used them to grant approval. The project dashboard was updated to show that the project did not require management attention, and to show that the project had moved to the design phase. The project lead then sent out a project status update to notify the team of the phase change.

4.3 Design

Inputs to the design phase were the user acceptance tests and the current implementations of the pages that would become dynamic at the end of the project.

One acceptance test that had a major effect on the software design was that site editors should not be required to know how to program in order to use the software. Because of this requirement, Java Tag Libraries were used to do the display of each section of a page. Tag libraries encapsulate display code into a Java class, which can then be called by a single HTML-style tag. For this project, sections of the page were broken into logical units for which the data can be retrieved one query, and the tag library then formatted the query results and returned them to the user's browser. Figure 1 shows the page design with two logical units outlined. This allowed editors to place sections of the page where they were needed or omit a section altogether if a page did not use it. The downside of this solution was that changes in page layout required recompilation of the code, which reduced the flexibility of the site and responsiveness of the team to change. It also meant that each tag required parameters to set widths of tables, heading styles, and other display attributes, in case the tag was used in an area of the page where those display elements varied from the original design. This was acceptable in return for simplified ease-of-use for editors, and because most editors were not able to edit or recompile the tag library code, they wouldn’t be tempted to spend time rewriting the HTML code.

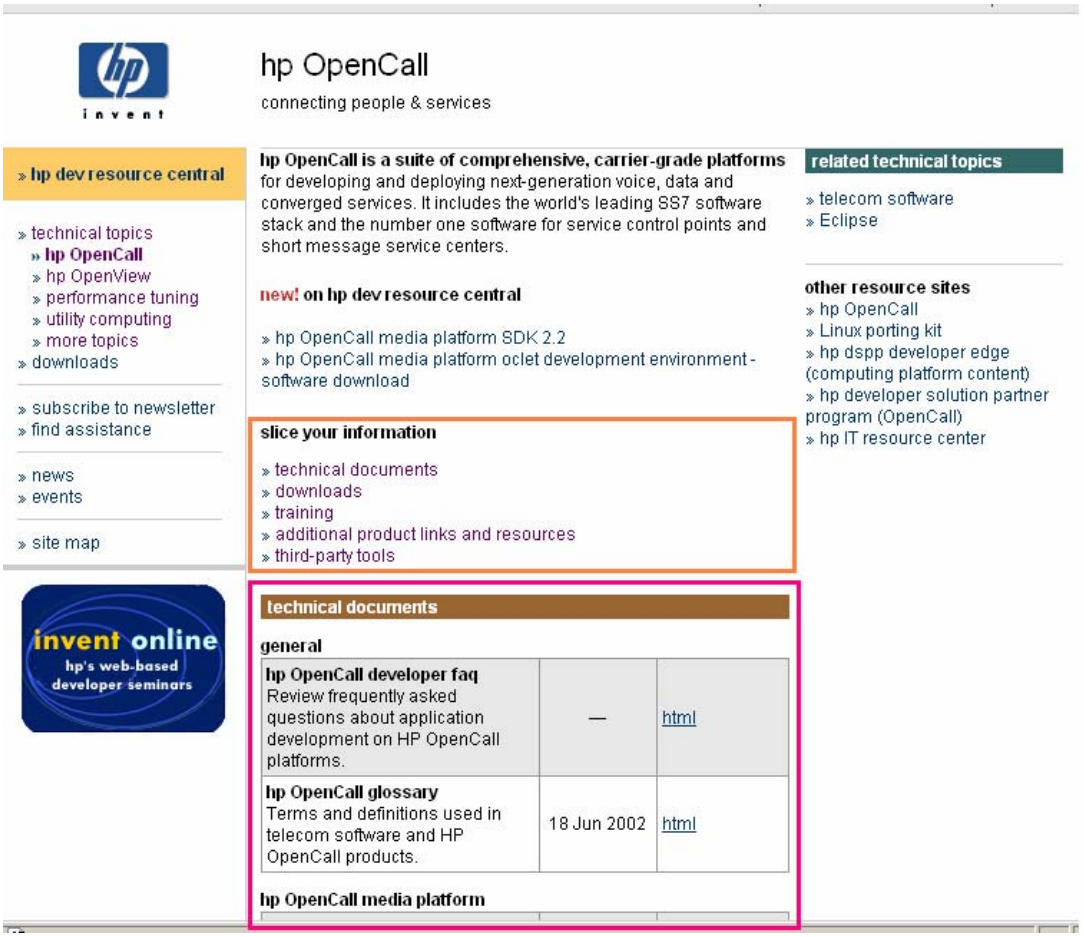

Figure 1

Another important set of acceptance tests stated that the final display must match a compromise design that meets HP.com style and layout guidelines, and must incorporate the most common or most important elements of the disparate topic pages on the site. Many of these changes were included in the final design as design enhancements, but some were discarded or entered for consideration as enhancements to future versions of the software. Part of the testing included comparing the final page layout to the agreed-upon compromise design.

The main outputs of the design phase were the page design, the list and description of fields that were necessary to accomplish that design, and the design document. The design document's contents are detailed in the following paragraphs.

4.3.1 Application Design

The data flow diagram described, at a high level, how information flowed through the website, from the editors who input the data to the website user who consumed it. The diagram followed standard DFD rules, and was created using Microsoft PowerPoint.

The data model described the database tables that were necessary to support the dynamic page display. Requirements were determined by examining the information and layout of the pages to be displayed.

The integration requirements described what hardware and software were required to add this new software to the existing system. For this project, there were no new hardware or software requirements, and it was important to state that so that management and the infrastructure team are aware that those needs had been considered.

Application inputs are the things that tell the application what to do. These included the request made from a customer's browser and the data from the database.

The application process described how the application consumed data from the database and transformed it for use by customers of the website.

Application outputs are the results that are produced by the application in response to a request from a customer. The main application output was the response the server returns to the customer's browser. The other application output was a message written to log files when errors or warnings occurred in the application.

Next was a list of error conditions that could occur in the application and procedures to follow to fix them. Some error conditions and procedures for this project were:

Pages display navigation but no contents. This indicated that the database connection is down, since the navigation portions of the page are static. To confirm that

this is the case and correct the problem:

1. Refresh the screen.
2. Check the source code; the tag libraries usually write an HTML comment for debugging purposes when an error occurs.
3. Check other pages that use the dynamic code.
4. Check that the backup server works; if it does, fail over.
5. Check that the database is up; contact the infrastructure team if it is not.
6. Contact the infrastructure team to restart the web server.
7. Contact the infrastructure team for further help if none of the above fixes the problem.

The potential problems listed in this document were not application errors, because those errors should be caught during testing before application deployment; generally they were errors in the system that would result in catastrophic failure of the application. For example, the problem listed above might result from a network outage or a hardware problem on the database server. Problems with data missing from pages where it was expected are generally user error: the data was most likely not set correctly or was not published to the correct server.

Security requirements described the ports and machines that had access to the database, and the user that must be used to log in to the database.

The scheduling requirements section of the document described factors that could affect scheduling of the deployment of the project. Higher-priority projects, such as content requests, could push out the schedule, and the deployment of the project depended on the release of the publishing project. Deployment was scheduled during the

bi-weekly change management meeting held by the infrastructure team.

Audit requirements described the tests that pages had to pass in order to be deployed. The web pages resulting from this project were required to meet corporate look-and-feel guidelines. Additionally, the pages were required to meet site design guidelines, conform to XHTML standards, and be well-formed.

Application performance requirements defined the performance required of the page before deploying the individual pages. There must not be a noticeable difference in the time it takes to display the current static pages and the time it takes to generate the dynamic pages.

The testing requirements section simply mentioned the document that contained the user acceptance tests. The tests themselves were documented separately since this document is just a reminder that all tests had to pass before the application can be deployed.

The implementation plan detailed the steps to take to deploy the code to the production servers. A test plan that could be executed without help from the development team was included so the infrastructure team could deploy and test on its own schedule. In case the tests failed, a rollback plan was also included.

The conversion design was a mapping that described the conversion of the existing data tables to the tables that supported the new page layout. This design was relevant to both this and the Content Management System project.

The conversion process design described the process to follow when converting the database for the new data design, and also the order to convert topic pages from static to dynamic using the new Java tag libraries.

4.3.2 Application Design for Conversion

Conversion validation requirements are the set of tests to be performed in order to confirm that the data has been converted and the page deployed correctly. For this project, the test involved deploying a copy of the page and comparing the data and functionality of the new page to that on the old.

4.3.3 Design Deliverables, Accomplishments, and Milestones

The main deliverables of the design phase were the topic page design, the list and description of fields that were necessary to accomplish that design, and the design document. The topic page design was used to create the tag libraries that generate the page display, and the list of database fields was provided to the Content Management System project for implementation. The design document listed the outside influences on this project, described how the application functioned, explained how the application would affect the system in which it existed, and how it would be deployed. The information in the design document was used by the project sponsor to decide whether to approve the project or not, and to be sure that there is a plan for development. When the project sponsor approved the design, the design phase was complete.

4.4 Summary and Phase Outcome

At the end of the Scoping, Analysis, and Design phase, the project team was ready to begin construction and test of the software.

The project moved forward based on the information defined in this phase: the functions that the software must perform were defined, editors would be able to enter appropriate metadata to be stored in the database, and users would be able to get the information they need based on that document metadata. Additionally, management

knew the status of the project, and agreed that it was on schedule.

5 Construction and Test

5.1 Methodology

The construction and test methodology for this project was probably the part of this process that most resembled the XP process, which prescribes a continuing process of test-build-compile-test in small increments (Extreme Programming). The process overhead was minimal, and testing was a very important part of the process. Testing was so important, in fact, that it was done hand in hand with the development of the actual software: a unit test for each new portion of the software was created, and then the functionality was added until the test passed. That test was performed along with all the other tests as part of integration testing when the code was checked in.

Compared to other test methodologies, the process overhead for managing this project was very low. Semimonthly progress checks made management aware of progress and allowed adjustment of priorities as necessary.

Most traditional methodologies advocate testing towards the end of the build cycle. This does not acknowledge the reality of software development, which is that developers typically do not write large chunks of code and then compile. Leaving testing to the end of this process leaves more opportunities for small parts of code to be overlooked in testing at the end of that process. Writing tests for functionality before the functionality is written ensures that the code being written has a predefined purpose, and directly relates to the goals of the project.

If the project team were to follow the XP process strictly, there would be a pair of people writing code together (Extreme Programming), but resources were slim: there was really just one person doing coding. Also, there would have been a slightly more

structured approach to writing portions of code: each team member would sign up to make a particular user acceptance test work, and would need to estimate the time required to complete each task. Since there was only one team member working on the code for the project, the process had to be more informal.

Inputs to the construction and test process were all the deliverables from the design section. The infrastructure provided by the infrastructure team was also included.

The completed software was comprised of a method to connect to the database, an object that represents each document's metadata, and a unit of code that formatted the metadata for each section of a topic page. These deliverables were continuously tested as each increment was completed, and the automatically-generated test results were posted to a central website.

The database connection was fundamental to the project and to the page display. The connection was made via a tag that is called like the display tags, and made query results available to all nested tags. The connection, once established, was used by the display tags to send queries to the database. The data for each tag is available only to the tag that made the query. Tests for the database connection included performance tests and data integrity tests.

The results of the query were marshaled into a Java class that represented the document, which had getters and setters for each field that was retrieved from the database. This meant that any changes to the names of database columns or changes in the database structure wouldn’t affect the display code. It also meant that any changes to the document model would require that the Document class be recompiled and redeployed. Unit tests for the Document class included tests for the getters and setters of the class.

The display tags used the Document class to retrieve metadata about a given document in a conventional, easy-to-understand way. Tests for the display tags included HTML validation, look and feel tests, and tests for completeness of data as compared to the pages in production. In addition to comparing the generated display with the existing pages and the design, it was important for the person writing the code to check with the designers frequently to reconcile vagaries and variables in display.

One of the fundamental tenets of the XP process is to do work in small increments (Extreme Programming). This makes the process very flexible, and takes into account the fact it is very unlikely that all requirements will be discovered in advance. Since there are frequent releases and iterations, users can still gain benefit while the project evolves. In this project, there were several metadata changes required and compensated for using this approach.

The biggest lesson learned in this project was to test with real data. In order to avoid causing a strain on the production system, the software was developed against a test database, with information that was populated manually as it was needed. Although most of the data was similar to the real site, it did not cover corner cases. Editors, who were most familiar with the data, were reluctant to spend time populating data, especially since there was no documentation and therefore no way of knowing how the data they entered would be displayed. Because of this, some document metadata was initially left out of the database. For example, some documents needed to be opened in a separate browser window, and that information needed to be noted in the database, but was originally overlooked. There also needed to be an explicit flag to note when a resource had recently been substantially updated. Although the system could tell when the last update was made, it could not differentiate between a major update to the content and a

minor update that corrected spelling errors or page formatting. It was desirable to note the major updates on the topic pages, but not the minor ones that users would not care about.

There were other problems to reconcile: some resources that the site linked to – documents that are not hosted on the Dev Resource Central site – were new to the site, but had been created months before. To help customers who visit the topic pages frequently, the links that are new to the site should be marked as such, but the actual publication date of the article can not be used as an indicator of newness in this case. This necessitated multiple date fields in the content management system: one for the publication date, and one for the date the document was initially linked to from the Dev Resource Central site. Furthermore, if both dates are the same, editors should not have to set both, and the application or query code needed to be able to determine which date to use.

The features and highlights areas of the topic pages were used to draw attention to important news about the page topic. In many cases, the features and highlights areas of the page provide extra exposure for a new link. In some cases, editors may want to change the description of the new link for each topic page the link shows up on, so the initial project deployment included the ability to make this alteration. In practice, however, editors rarely had time to customize the text for each topic, so that functionality will be removed in a future project to make the CMS user interface cleaner and remove irrelevant data from the database. In the case where an editor felt it was really necessary to have a different description, it would still be possible to work around the removal of the extra metadata fields by creating another link in the CMS to do the highlighting work.

To help set customers' expectations about what will happen when they click on a

link, the site convention was to display the format of a file next to the link when the destination file was not an HTML file. This was useful when a browser plug-in was required to view the content, as was the case with videos and PDF files. The content management system did provide an automated file format field, but its convention was confusing in some cases, so it was necessary to add a metadata field to contain a text description of these files.

In the original page design, the document title, date, and description were not links. The file format was off to the side and was used as the text for the link. In some cases, the file format did not work well as the link text, so an additional field was added to allow the default link text to be overridden.

The data access tag that created the connection to the database for topic page display initially created an individual connection for each page view. Tags that queries for metadata and formatted it for display then used that connection, and when the closing data access tag was reached, the database connection was closed. When the initial performance problems were encountered, the data access tag was modified to get a connection from a pool created by a servlet at server startup instead, which maintained a steady connection to the database and would reduce the time needed to make that connection for each page. Although most of the initial performance problems were not related to the database connection tag, this change would improve performance, help avoid deadlocks, and keep the overall number of connections to the database from growing unnecessarily.

5.2 Deliverables, Accomplishments, and Milestones

Deliverables of this phase were unit and integration tests and a test suite, a build script, completed software, system documentation, user training documentation, and

sponsor buy-in for phase completion. Sponsor buy-in for phase completion came only when all construction-test iterations were complete. All iterations were complete when all user acceptance tests and the test suite passed.

6 Implementation and Project Ending

6.1 Implementation

The inputs to this phase were the completed software and test results, and the implementation plan from the Project Scoping phase, and the user documentation for the application. These were used to gain approval for deploying the application on the production servers in the formal change review meeting.

The outputs of this phase were the deployment of the application, a plan for the ongoing support of the application, and training for the users and support staff.

In order to implement the application, the infrastructure team had to give approval to confirm that they were aware of the project; that its effects on the infrastructure shared by other websites had been considered; that testing was complete and thorough; what machines will be affected; and that they had an implementation plan including backup, install, test, and rollback plans, and the final files to be deployed. When the infrastructure team approved – meaning that an executive council approved of the implementation plan after it was approved by an infrastructure team support engineer who worked on the system every day, and by a project manager who kept track of what software deployments are going on and schedules deployments to minimize user effects – the application could be deployed.

The support plan details the logical pieces of the system and their primary and backup support engineers. While the system needed to be well-enough documented that anyone could get the software rebuilt and running again if necessary, a primary and

secondary support person were documented. The primary support person was generally the primary developer of each logical portion of the system, and the secondary person needed to be familiar enough with the software that recovery from errors would not be overly time-consuming. This generally required a little bit of training, so support staff training was another deliverable. The support plan also detailed the most likely failures for the system and software along with a brief description of the symptoms of each type of failure and the steps to solve the problem. This document was not meant to be a comprehensive support manual, but rather an aid in emergency situations and to help management understand the risks they might face, as well as who to turn to in case one of the failures occurs.

Finally, the users and support staff were trained: while this sometimes meant actually holding a meeting to provide training, for this project it meant that all users needed to read the documentation created in the construction and test phase; support staff were expected to read the documentation as time allowed and need demanded.

6.2 Project Ending

After the software was deployed to the production servers, the system was monitored for a few days to make sure there were no problems. At the end of that time, the project sponsor's approval was solicited to end the project. The project outline did list the application as a deliverable, but did not list the conversion of all topic pages as such. Since the performance issues made conversion impossible, the pages were explicitly excluded from the deliverables section at this point. Since it was possible to generate pages using the application, the completion criteria specified in the project outline were technically fulfilled. Approval to end the project was granted, and the project was considered complete and moved to maintenance. The project management website was

updated to show the maintenance status.

6.3 Summary and Phase Outcome

This project was considered complete even though the software application created during the project was not immediately put to use in all topic pages. The conversion of the topic pages was specifically stated as being outside project boundaries because of the potential to keep the project open for a long period of time.

7 Outcome & Project Maintenance

7.1 State of the Project

After deployment, the software was not immediately put into use on the system: there was not enough data in the database, and there was a question about what the performance would be in the production system. As it turns out, the performance was not acceptable. The physical distance between the web server machine and the database was too great, and although the query speed was excellent and the display time was acceptable, the data could not be returned from the database to the web server fast enough. This meant escalation: none of the other sites using the database had their normal pages as tightly coupled with the database as a back end, so it was not an issue except on the Dev Resource Central site. A large infrastructure improvement project was underway that would address the problem, so this meant a delay in the rollout. Instead of converting a high-traffic topic page first, the lowest-traffic topic page was converted as a test; that page would require the smallest amount of data entry time and would affect the fewest customers, while still allowing a test on the production server. Testing showed that the software was stable enough to be in production despite the performance issues, and the test page would be used to find whatever remaining design flaws existed before

all the pages had to go into production using the software.

7.2 Summary of Project Outcome

The application that was the result of this project performed satisfactorily in the right environment. Due to forces outside the control of the project team, it could not be put into use immediately. This was disappointing, but meant only a delay in the editorial team's ability to make use of the application – the time spent on the project was not wasted; just premature.

7.3 Project Maintenance

The first plan for the project during maintenance phase was to convert topic pages to use the new software. This required that all the metadata for all the documents on the site be entered, but it also required faster performance from the query. Until the database could be moved, the editors were encouraged to enter metadata when they entered new content so that the burden of entering metadata would be lighter later on. Once the database and web servers were moved to the same location, performance was acceptable and editors began entering metadata more often. Ultimately, the amount of data that needed to be entered was too daunting to be done ad hoc, and a small project was required to get all the metadata entered and the topic pages migrated.

8 Future of the Project

8.1 Future Plans:

8.1.1 Enhancements

After the additional topic pages are converted, there are several projects that could make enhancements to the topic pages easier.

It would be nice to decouple display more from the back end. The HTML that is generated for each page is currently hard-coded into tag libraries. Moving the HTML strings into properties files to make changes easier is one option, but it would not alleviate the need to recompile the display logic when page layout changes; it would simply make it easier to make minor modifications in the case that the HTML code is invalid. Any changes to the display require changing Java code, compiling, testing, and deployment, a process that can be time-consuming and process-heavy due to the involvement of the infrastructure team. If the performance is not adversely affected by writing the display code in JSP, the Java coding and compiling steps would be simplified, and the extra process requirements for deployment that are imposed by the infrastructure group for compiled code would be removed. This would make the team more able to respond to changing requirements. This improvement will be more complicated than any of the others, and will only be considered when it would take a significant amount of time to make a change or implement a new tag.

Another part of the software that is too inflexible for a fast-changing web environment is the Document object that is used to hold data retrieved from the database. The metadata is marshaled into the Document object, which gives the display code a convenient way to refer to each metadata field. The database is not expected to change often or drastically, but even when it does, the Document object isolates the display code from some of those changes: for instance, if a column name changed in the database but still contained the same data, the column name would only have to be changed in the code that does the marshalling. In practice, the column names will not change, so this precaution turned out to be overkill. Additionally, since code changes would be required whenever a new column is added to the database, the database changes can't be taken

advantage of very quickly. If the display code were migrated to JSP, JSTL tag libraries might help with this: query results can be automatically marshaled into easily-accessible objects that have a getter- and setter-like interface, but are more flexible than the Document object. This change would be implemented when the pages are converted to use JSP for the display, and the Document object would simply be removed when the tag libraries that use it are no longer in use.

Finally, it would be ideal to move the queries into a properties file to make it easier to modify them. Currently, each tag that makes a query has the SQL statement hard-coded; if even a minor change is made to the SQL, the entire application must be recompiled and redeployed. Since moving the queries into a properties file would require the least amount of time and can be used in all of the other potential improvements, it has first priority.

These enhancements will be scheduled to fit between content generation and formatting tasks. The process will not require the same amount of documentation, since the scope of the enhancements is not as large as that of the original project: there will be no project outline, and no need to get approval for the changes from the infrastructure team. The user tests will be reused since the enhancements simply re-implement the same site functionality in a different way. For instance, the test that states “Topic pages should have a ‘related technical topics’ section” still applies, regardless of the technology used to get it done. This is strength of the user acceptance tests: they should always apply and pass, as long as the tasks a customer would want to perform do not change. Unit tests will change as the code itself changes. As with the original project, all unit tests must pass, and unit tests for new code should be written before the code. Deployment will still be subject to approval from the infrastructure group, but JSP code

and the removal of code from the system when it's no longer in use will not require as much scrutiny on their part.

8.1.2 Maintenance

The application will not be able to remain in use in an unchanged state forever; the web environment changes constantly.

The application will need to be updated for display changes. These changes may be required in order to conform to corporate look and feel guidelines. As content and audience change, the layout of a page may change as well, which will require changes to the display tags. This type of change may also require changes to the database and relevant queries. Any change to the database will have to be more closely scheduled than changes to the code; the code can be deployed quickly and without customers noticing thanks to backup servers, but the way the database works means that columns can't be added or removed without effecting pages that rely on it for display. Database changes will therefore require more coordination with the infrastructure team and within the project team, and in order to reduce the effect of these changes, it may even be necessary to temporarily replace dynamic pages with a static copy.

While this project focused on displaying only one type of page dynamically, there are other pages on the site that would benefit from the application. Creation of these pages is included in the maintenance phase of this project.

The web application server and Java version used by this project will inevitably be upgraded as well, and the unit tests will need to be run when this happens.

9 Conclusions & Lessons Learned

9.1 Investigation and Planning Phases

The research, scoping, analysis, and design phases of this project went fairly smoothly. Although the project team was limited to the software provided by the infrastructure team, the limitation kept the investigation from being bogged down in nearly limitless possibilities. The project scoping document, known to the team as the project outline, made clear the boundaries, deliverables, and resources for the project. These were useful when later parts of the project threatened to grow too large, and helped keep the team from spending too much time considering alternatives.

The design phase resulted in documentation that was a good starting point for the implementation phase. Although the application design turned out to be incomplete, the project outline document helped track changes in the project, and the project was able to move forward and accommodate changes when they were discovered to be necessary. The user acceptance tests provided a comprehensive set of functionality for which software needed to be created.

The design could have been improved with more people involved to notice the vagaries of display on the various topic pages in advance. Fortunately, the project management style used allowed for a lot of flexibility in the project: when new requirements were discovered, the project manager needed to document the requirement, evaluate the new requirement's importance compared to the other tasks, and inform the project sponsor if any schedule or resource changes were required. The project sponsor could then evaluate proposed changes in the schedule to decide whether the new requirements were more important than other concurrent projects and whether the changes would affect follow-on projects. This arrangement allowed the project team to

compensate for its small size, yet complete the project on time.

9.2 Construction and Test

The iterative style of development and testing suited this project well because it took into consideration that designs are frequently incomplete to start with and provided a way to be successful even with an incomplete initial design. Although the design was as complete as possible at the end of the design phase, it would be nearly impossible to cover every area of design in a short period of time with a very small team.

The user acceptance tests provided direction for the project and were a good way to prioritize development tasks. The tasks that were marked *want* were lower priority, and in some cases did not get done. Some of the tasks that were marked as *musts* were downgraded to *want* when it became clear that they were not really necessary for the project to be successful. It would have been good to be more conservative in the estimation of what was really necessary to complete the project.

The unit tests helped show when the code was working and when it was broken, provided up-to-the-minute status updates for management, and helped improve code quality overall.

9.3 Project Management

The project outline was a very effective tool to manage and track the progress of the project. It clearly communicated the project status and showed the schedule at a glance, which was valuable to the project team's management. The largest delays in the project were the results of forces outside the project: the project on which this one depended slipped, and when the project was completed, it couldn't be fully put to use due to infrastructure problems. The other impediment to using the application, which was the

large amount of data that needed to be converted, should have been defined in advance as a follow-on project. Fortunately, this oversight did not affect the project's schedule.

Overall, the project was a success. The project management overhead was kept to a minimum, but still met the needs of management and the project team. With future enhancements in place, the editorial team will be able to save time editing topic pages and spend more time working on delivering content pages to customers.

10 References

Extreme Programming: A gentle introduction. (n.d.) Retrieved June 15, 2006, from http://www.extremeprogramming.org/

11 Exhibits

11.1 User Acceptance Tests

Test No.	Weight 1 (lo) – 5 (hi)	Description	Criteria	Comments	Status (p/f)
A-1	5	An 'other resource sites' section should show up on the devresource home page when other resource sites exist. See the orange-outlined section of figure 2, appendix.	Does the 'other resource sites' section of the home page contain the links identified by the editor and only those links?	3/9 no test data entered for home page yet.	P 3/12
A-2	3	A 'features' section of the page is dynamically displayed for both the home page and topic pages. See figure 1, appendix. This section of the page may be comprised of one or more documents that have been specified to be featured documents for the home page. The features section should not have a heading, and should be at the top of the center column.	a. Does the home page have 1 or more featured documents? b. Does the OpenCall topic page have a features section? c. Does the features section appear at the top of the center column? d. Does the features section have a header? (it should not)	The contents of this section of the page are specified in the "long description" field in the CMS; the length of this field is 2000 characters. 3/9: no test data entered for item a yet.	a. P 3/12 b. P 3/9 c. P 3/9 d. P 3/9
A-3	5	A 'developer programs' section of the page should show up on the home page and all appropriate topic	a. Does the "developer programs" heading appear on the home page? b. Are there one	3/9: no test data for item a yet.	a. P 3/12 b. P 3/9 c. P 3/9

		pages where developer programs exist. See the orange-boxed section of the home page, figure 3, appendix.	or more links below the heading? c. Does the "developer programs" heading appear on the OpenView topic page?		
A-4	5	An image appears in the top-right corner of the content area of the home page. This image is not dynamically generated or placed on the page.	Does an icon appear on the page?		P 3/9
A-5	5	Topic pages should have a full listing of content, including an in-page navigation section.	a. Is there an in-page navigation section on the page, with the heading "slice your information"? (See orange-boxed area, figure 4, appendix) b. Is there a list of content following the index? (See pink-boxed area, figure 4, appendix) c. Does the number of color-bar headings in the content list match the number of links in the in-page navigation section? d. Do all the in-page navigation section links jump to the correct place		a. P 3/9 b. P 3/9 c. P 3/9 d. P 3/9 e. P 3/9 f. P 3/9

			on the page? e. Does each content type have a color bar heading? (it should) f. Do content listing sections have sub-headings?		
A-6	5	Topic pages should have an 'other resource sites' section on the right side of the page, if the data dictates it. See figure 5, appendix.	a. Does this section show up on the OpenView page? (it should) b. Does this section show up on a dummy page with no resource sites? (it should not)		a. P 3/9 b. P 3/9
A-7	5	Topic pages should have a 'highlights' section in the rightnav. See figure 6, appendix. The highlights section should be indicated by a colorbar.	a. Does the Highlights section display (dynamically) on a dummy page with highlights specified for it? (it should) b. Does the Highlights section display on the OpenCall page? (it should not) c. Is the highlights section header a color bar? d. If no highlights are specified for a topic or the home page, does the highlights heading show up? (it shouldn't)	3/9: Highlights will not function until the ext_subject field exists in Autonomy.	a. P 3/12 b. P 3/12 c. P 3/12 d. P 3/12
A-8	5	Topic pages should	a. If a topic page	3/9: need to	a. F

		have a 'related technical topics' section. See related technical topics section, figure 4, appendix.	is specified to be related to another topic, does it appear as such on the appropriate topic? b. If a topic page is specified NOT to be related to another topic, does it fail to appear as such on the appropriate topic? c. If there are no related technical topics for a topic page, is the section heading absent?	enter test data	b. F c. F
A-9	5	Rightnav items on a topic page should appear in the following order: - highlights - related technical topics - other resource sites - developer programs - news	Do rightnav items appear in the order specified on all topic pages?		P 3/9
A-10	5	Manually-created pages and dynamically-generated pages should appear identical except where human error is present in the manually-generated pages (for example, the inconsistent ordering of content types).	Is there a noticeable difference in the appearance of the pages?		P 3/12
A-11	5	Manually-created pages and	Is there a noticeable	There will be a noticeable	P 3/12

		dynamically-generated pages should take approximately the same amount of time to display.	difference in the amount of time it takes to retrieve the page?	difference the first time a page is generated after being published, since the JSP must be compiled at that point. After the first compilation, the page should not take noticeably longer to display.	
				Pass	10
				Fail	1