Project 1: Map That Site

Version 1.0 of 1/11/01 — Subject to Revision

Due Dates

The project concept document is due at the beginning of class on Thursday, 1/18/01.
The group will present an overview of the document in class on the same day.
The project is due Monday, 2/5/01, 10:00 PM.

Introduction

This is the first of two projects that you will complete as a team in CIS 422. The primary purposes of this first project are to give you some practice in working as a team to produce a complete software system with a deadline. You will also get some practice using existing components and tools to accelerate your development by minimizing the amount of new code that you must write.

I have attempted to make the first project technically easy, so most of the issues you will face will be non-technical — how to divide up and coordinate the work, how to work together and not hate each other, how to avoid catastrophe if one person gets sick or flakes out, etc. The project is small, but the deadline is tight.

Web Mapping

Basic requirements

You will construct a tool for building graphical maps of web sites represented as directed graphs in which documents are nodes and links are directed edges. The input of your tool is a web site. The output of your tool is a graphical depiction of the web site, in a form that will be useful to a maintainer of that site. At a minimum, the graphical representation of the site should

- Distinguish html files from other web-accessible documents
- Display all links from html files to other documents
- Distinguish relative links from absolute links
- Distinguish external links (those to documents outside the site being mapped) from local links. Broken local links should be prominently displayed.

When you are done, your project should be a high-quality freeware tool that can be distributed in source form over the internet. It must include full documentation, including installation and configuration instructions and examples.

Other requirements

This handout is not a complete statement of requirements for your product, because it is *your* job to complete the requirements and design decisions. There are many open issues for your to resolve. Here are a few things to think about:

- What platforms will your product run on?
- Will you map web structures that are built on-the-fly (e.g., by CGI scripts), or only static document structures?
- Will your displays be static (e.g., a GIF graphic) or dynamic (e.g., using a Java graph-editing applet)?
- What is the capacity of your tool? Can it display large and complex web sites in a useful manner, or will they be depicted as an unintelligible tangle?

- Can the user customize the display your product produces? Must the customization be reapplied each time the tool is run?

These issues barely scratch the surface. The bottom line is: Build a tool that people will want to use, and that they will be able to use easily. I predict that you will find deciding what to build a good deal harder than building it. You should identify objectives, alternatives, and constraints for your project, as well as you are able, and begin to identify risks and the ways you will obtain more information to assess and control those risks.

Even though in this assignment you are being given some aspects of the solution in advance, your requirements section should start with a clear and concise one-paragraph problem statement that explains what is the fundamental problem that your system will solve, independent of the solution.

Distributing the Work Within Your Team

You must assign responsibilities to team members. It is not a good strategy to simply share all responsibilities. Although all team members should contribute to some extent to every aspect of the project, it is essential to have one person with central responsibility for each major part of the task. Among the responsibilities to be assigned include:

- Manager. This person is primarily responsible for administration of the project schedule.
- Quality control. This person is responsible for administering the quality plan.
- System architect. This person is responsible for the overall architectural design of the system.
- Technical documentation. This person is responsible for documents meant primarily to be read by software developers and maintainers.
- User documentation. This person is responsible for documents meant primarily to be read by people other than software developers, such as end users.
- User interface. This person is responsible for human factors and ensuring a good "user experience."
- Configuration control and product build. This person is responsible for managing the construction of software products, including coordination of multiple developers.

The mapping of people to roles is not necessarily one-to-one. For example, it is common in small teams for the user documentation and user interface roles to be combined. You may wish to identify additional roles. You may also want to spell out responsibilities more clearly, e.g., shall the user documentation or technical documentation person be in charge of the final presentation?

The person with ultimate responsibility for one of these functions does not have to do the whole job alone. For example, the system architect does not design the whole system alone, and the product build manager does not do all the implementation; they are managers of different aspects of the project.

Risk control is an important part of project management. One of the largest risks to any software project is the loss of a key person. Therefore, while it is important that a single person be ultimately responsible for each role, it is a very good idea to assign a "backup" for each role also. The backup person assigns the main person responsible for a role, and should be knowledgeable enough about that role to take over responsibility if the primary person is lost or unable to fulfill his responsibility. The backup role is also an opportunity for "cross-training" in preparation to take a lead role in a later project.

The Mini SRS / SDS / Project Plan

The Mini SRS / SDS / Project Plan document is a small combination of elements that might appear in several different documents in a larger project: A proposal, a feasibility study, a project plan, a requirements statement, a specification and/or external design, and an architectural design overview. This document should convince management, a client, or an investor that this project is worth funding. The quality and content of the document will communicate the likelihood of success of the project if it were to be approved.

Your Mini SRS / SDS / Project Plan should include at least the following:

- A clear and concise one-paragraph problem statement
- A description of the product you intend to build. This should describe the externally visible behavior of your product as precisely as possible, but it should be concise and clear.
- An overall design description. What are the major parts of your system, and how do they fit together? What are the main organizing principles that you used to break your system into parts?
- A management plan. How is your team organized? How is the work divided among team members? How does your team make decisions? How do you check progress against your plan? How will your team meet and how will it communicate?
- A build plan. What is the sequence of steps you will take to build the system? When will each "build" of the system take place.
- A rationale for the overall design and build plan. Why have you broken the system into these parts, and why have you chosen these particular steps to build the system?

Each group will prepare an in-class presentation that covers each of these six topics. The presentation, as with the document, should be intended to convince the audience that your project is worth their support. In both the document and the presentation, solid ideas and good content will go much further than e-marketing mumbo jumbo.

Project Management

The group should keep a record of meetings. This record should briefly note the agenda of the meeting, the date and time, who showed up (and who showed up on time), and what was accomplished during the meeting. It's a good idea to set ending times for meetings to, to help you get through the agenda items.

The group should also keep a record of each of the tasks that are assigned to each group member. This should be in the form of a spreadsheet or table with the following columns: The task, assigned to whom, when assigned, when due, when completed, who did it, who signed off on it.

Reuse Guidelines

The cheapest, most dependable and least risky software components are those you don't build. You can find graph layout and display components, web spiders, and other nifty things freely available on the web. I strongly suggest you take advantage of them. On the other hand, you must do so in a way that is legal and ethical, and while I won't set an upper bound on how much of your project code can be reused, you must certainly provide some "value added" and not merely repackage software available elsewhere.

You may not, however, consult with students or re-use code written for previous 422/522's.

To be legal, you must obey all copyright restrictions in software you use. Beware that a document or file need not contain an explicit copyright statement to be protected by copyright law; you have a right to copy or reuse something only if the author has specifically granted you that right. I am absolutely firm on this, and will not hesitate to fail an individual or a whole team for unethical conduct as regards intellectual property. If you have any questions about what you may or may not do, ask me.

Your product must be freely distributable under the Gnu copyleft agreement. In some cases this may mean that you cannot make use of some software which is otherwise perfect. In other cases it may mean that your product will depend on other software packages that you cannot directly distribute. (Be careful of such dependencies, especially on commercial software, as they can make your product more difficult to install and use.)

To be ethical, you must clearly document the original source of all software and other documents. Every source file must contain header comments clearly identifying its author(s). Derivative work (e.g., code written by you but adapted from a book) must clearly cite each source used in its creation. Falsely identifying yourself as the author of something that is actually someone else's work, or failing to properly cite a reference on which you based part of your work, is plagiarism and will be dealt with very severely.

It is entirely possible to follow these guidelines, making only legal and ethical use of other people's work, and still to avoid a lot of design and coding that would be required if you built this project "from scratch." Sometimes you will find that, even if you cannot directly reuse code (e.g., because it is written in a different programming language), you can still reuse design. You should properly cite the sources of reused design as well as reused code.

Approximate Schedule

Week 1

Before you are assigned your team, start working on the project on your own. Design work and brainstorms are usually more productive if group members first do some thinking on their own, without the inertia of groupthink to pull everyone down one or two paths. Work on the issues that will be in your product concept document due at the end of week 2. Browse the web looking for products (freeware, shareware, and commercial) that do something related to the web mapper. What is the competition? Are there "market niches" for your product? Look also for useful components that you can reuse.

Week 2

You are going to need at least a couple of intense team meetings to agree on your product concept, in addition to more individual research. Think seriously about the feasibility issues: How is your team going to divide up the work and tackle the problems? Produce the product concept document and present it to the class on Friday.

Week 3

Delivery is less than two weeks away, and deadlines that looked easy before are starting to get scary. Don't panic. Do make a plan that includes early production of a running prototype (no matter how lame, it just needs to work) and frequent revisions. Make contingency plans for the failure of anything that isn't already running. You really, really want a running prototype before class Friday of this week, so that you are ready to discuss any remaining problems.

Week 4

It's crunch time. You're on a daily build-and-smoke schedule now. You have an agreed meeting time each day for putting together everyone's pieces and testing the current system version. The build-master is sweating bullets, but she has it under control so that, if it all blows up on Monday, the Sunday build is still good to go. You schedule intense reviews of documents and outstanding design issues. On Monday you declare victory and turn in your project.

Week 5

There's still one more chore. You need to present and/or demonstrate your project in clas. You will also participate in the class discussion. How did your approach to the project differ from Bob's group? Did you encounter some of the same problems? What seemed to work, and what didn't?

Acknowledgments

This assignment is modeled after an assignment created by Professor Michal Young for a previous offering of this course.