Project Number. GFP0707

WEBFOOT DOCUMENT VIEWER

A Major Qualifying Project Report:

submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by:

Tim Richardson

Michael Moscardini

Date: April 24, 2008

Approved:

_____

Professor Gary F. Pollice, Major Advisor

1. Webfoot
2. Collaborative Development
3. Eclipse Plug-in

Abstract

This project allows for teams of software developers to manage documents related to development projects from within the development environment. Two separate tools are integrated in this project, the SourceForge project environment and the Eclipse development environment. The project allows for the viewing and manipulation of the documents on SourceForge from within Eclipse. The plug-in created uses the Webfoot project as its base for connecting to SourceForge, and as such is part of the Webfoot project itself. The project allows documents to be created and deleted, it also allows for the viewing and editing of documents details, such as version, status, and associations.

Acknowledgements

We would first like to the thank Professor Gary Pollice for his dedication to our project. He was always willing to answer questions and was always very approachable, which we greatly appreciated. We owe much of our success to his guidance, both on our project and in the classroom.

We would also like to thank the other teams working on Webfoot. It can be difficult to gauge the progress of a project when there is no customer. Their feedback was extremely helpful in developing the necessary features to make this project a success.

Table of Contents

List of Figures

List of Tables

1. Introduction

Software engineers, both academic and professional, use collaborative development to brainstorm ideas back and forth between each other. This need has brought forth certain tools for which to do so. In the scope of this project a tool that has been used to accomplish such collaboration is a tool called SourceForge. SourceForge contains features such as discussion boards, documents, code repositories, wikis, file releases and other tools to help manage projects. This project's purpose is to assist in bringing the tools of SourceForge into the development environment through expanding a plug-in for Eclipse by the name of Webfoot.

Webfoot is used to connect to SourceForge and allow the manipulation of a subset of the tools available on SourceForge into Eclipse itself. There are many features that have been already added to Webfoot, such as task manipulation and view and the ability to view and modify trackers, as well as other SourceForge and collaboration tools.

We created the ability to view and manipulate documents from SourceForge within the Eclipse development environment using Webfoot. The team focused on the following operations: the ability to view all of the documents on SourceForge from within Eclipse, along with their details, and the ability to open the documents as well as edit the content and details of these documents.

This report describes the background of the project, our methodology in implementing this part of the Webfoot, our results and the analysis, and our insights on future work that could go into this plug-in. The background section outlines and describes many of the tools and ideas that were used, incorporated and expanded upon

during this project. These tools and ideas are collaborative development, open source software development, SourceForge, Webfoot and Eclipse. Our methodology explains how we approached the problem, went about creating a solution, and the problems that we encountered while doing so. Our results and analysis section demonstrates why we consider our plug-in a success. Although we deemed it a success, we feel that there is always room for improvements and additions, which are highlighted in the future work section.

2. Background

This project highlights a need that is at the heart of both the professional and academic environments, collaboration. Technology has allowed development teams to evolve into robust and flexible entities. Teams no longer rely on a centralized location to work on a project. However, teams still need to collaborate within themselves. When a project team is centralized, this is a much easier task to accomplish. When a team is dispersed throughout different parts of the globe then the teams need tools to assist with the collaboration. These tools range from basic emails, to online chat features, to video and voice conferencing. Developers are finding a way to integrate all of these different tools to best benefit the project and themselves. The goal of these tools is to allow for productivity from a non-centralized team to be similar to that of a centralized team. There are many tools available today, from simple code repositories, to centralized storage systems like SourceForge.

Collaboration has always been a part of the business world, whether it is a brainstorming meeting, or joint input on reports and presentations. The rise of the Internet has led to the increase in distant collaboration, whether through email, instant messages, chat rooms, or newsgroups. Some of these tools such as instant messaging and email have become the standard for of intra-office communication. Communication throughout an intra-office network has become a fundamental tool for collaboration in the workplace. Productivity software, such as Microsoft Office® and Open Office™, play a major role in this also. The ability to save what you have typed and send that to another person to review without printing out a copy revolutionized intra-office collaboration.

Productivity software is used in many aspects of software development today as well, be it design documentation or specification, or simply a spreadsheet containing certain data on a project. Centralized storage for these types of documents, along with centralized code repositories and the widespread use of the Internet, has allowed for development teams to reliably work from home, and also entirely from remote locations.

Software development is, in large part, done in integrated development environments, or (IDEs). There are many IDEs that are used by developers such as Microsoft Visual Studio, Eclipse, and NetBeans. Such environments allow for new features to be added through the addition of plug-ins. These plug-ins allow for additional features to be added to the IDE. Eclipse is a open source Java development environment that allows for expansion of its features through plug-ins.

Collaborative Development

The landscape of software development has transformed over the past twenty years, with the emergence of object oriented programming into the mainstream, the introduction of IDEs, and the advent of the Internet. Software development has also seen a higher demand for software, as well as a greater complexity of the software that is being written. This has led to software teams becoming larger and more complex. New roles have been created to deal with responsibilities such as graphics design and networking. The Internet has also brought with it the need for some developers to work from remote locations, be it working from home, or from another branch of the same company. This change landscape of development has spurred the creation of more sophisticated collaboration tools.

Collaboration tools have been an integral part open source software development, which has gained popularity in recent years. Collaborative development tools and practices have allowed for groups of people to work on projects from all over the world. These projects are both free as well as commercial, but their source code is usually open to anyone who would like to view it. This openness with code has pushed software to a different place, with the possibility of many different users adding to, revising, and cleaning up code that is used by many different people. Without advances in collaborative development, open source software development could never have taken off the way it has in the past decade. (1)

Open Source Software

In addition to the rise in collaborative development tools, open source software has also risen. Open source software is a set of guidelines and rules on how to write software, the main principle of which is that the source code is openly available. The Open Source Initiative was formed in 1998. This group sought to bring awareness and attention to open source software. (5) The basic tenants of open source software are defined in the open source definition as outlined by the Open Source Initiative:

1. Free Redistribution: the software can be freely given away or sold. (This was intended to encourage sharing and use of the software on a legal basis.)
2. Source Code: the source code must either be included or freely obtainable. (Without source code, making changes or modifications can be impossible.)
3. Derived Works: redistribution of modifications must be allowed. (To allow legal sharing and to permit new features or repairs.)

4. Integrity of The Author's Source Code: licenses may require that modifications are redistributed only as patches.

5. No Discrimination Against Persons or Groups: no one can be locked out.

6. No Discrimination Against Fields of Endeavor: commercial users cannot be excluded.

7. Distribution of License: The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product: the program cannot be licensed only as part of a larger distribution.

9. License Must Not Restrict Other Software: the license cannot insist that any other software it is distributed with must also be open source.

10. License Must Be Technology-Neutral: no click-wrap licenses or other medium-specific ways of accepting the license must be required. (5)(6)

Examples of widely used open source products are the web server Apache, the Internet browser Mozilla Firefox, and the Linux operating system.

The guidelines created by the Open Source Initiative are followed to varying degrees. This broader definition generally applies to software that's source is viewable. Developers license open source software under several different licenses. The most popular open source licenses include the GNU GPL(General Public License), the LGPL(Lesser Public License), and the MPL(Mozilla Public License). These software licenses have different rules regarding how software can be released and used. Figure 1 below describes some of the main differences between these three licenses versus proprietary software. In the figure the acronym IP stands for intellectual property. Code can contaminate other work by forcing the other work to follow its license. For example, if proprietary code uses GPL code, the proprietary code must now follow the GPL rules.

Code based on:	Can be proprietary	Exposes IP to open source[1]	Becomes open source[2]	Contaminates larger works
Proprietary Code	Yes	No	No	No
"MPLed" Code	Yes	Yes	Modified files only	No
"LGPLed" Code	No	Yes	Yes	No
"GPLed" Code	No	Yes	Yes	Yes

(7)

Figure 1: Open source licenses compared to proprietary ones.

Open source software usually has a general model of development practices. In the open source development process there is generally one person or group that delegates portions of code for "official" releases. These portions of code are then packaged up and made available for distribution to the desired audience. Tasks in open source projects are not generally assigned; the general convention is that if a particular part of the project is appealing to a person then he or she is able to contribute to that area of the project. A core set of developers have control over what code actually gets committed to the repositories, but other than that there can be contributors of all skill sets and skill levels at least partly involved in the development of open source software projects.

Peer review is seen as a major advantage over more traditional practices in open source development. A peer review is when a person, other than the original author of the code, goes through the code looking for defects that may be missed by the original author. This step is a great quality control and assurance method. This step also allows

others on the development team to familiarize themselves with the code that others have written. (11)

There are varying tools used in open source software development. Figure 2, below, shows some of the commonly used tools in open source software development.

Version control	CVS, Subversion, WinCVS, ViewCVS
Issue tracking	Bugzilla, Gnats,DebBugs,Bonsai
Communication	Mailman, IRC, MajorDomo, Ezmlm
Build systems	Ant, Make,Autoconf
Design and code generation	ArgoUM ,XDoclet, Castor
Testing tools	DejaGnu, Tinderbox, JUnit,Lint, CodeStriker
Collaboration Environments	SourceForge,Tigris, SourceCast

(4)

Figure 2: Common tools used in open source software development.

Version control is used to house code and its different revisions. For example, every time someone commits to the version control repository new versions of the files that have been altered are stored. If for some reason the committed code does not work you can usually revert back to a previously working version of a file in the repository. Issue

tracking is used to log current defects that need to be fixed in the code. Communication protocols are used to convey information between members of the development team. Build systems are used to make compiling code easier. Design and Code generation tools are designed to produce code or partial code more efficiently. Testing tools are used to run against new and old code to make sure that the program is doing what it is designed to do. Collaboration environments are used as a central place that information for a particular project can be kept so that a project team has an easier time accessing and sharing information about the project.

SourceForge

SourceForge is a centralized collaborative development tool for project, communication, and code management. SourceForge allows for software development teams to have a centralized place to manage a software project. The website for the project can contain documents pertaining to the project, file releases and reports as well as a wiki. SourceForge also allows for management of trackers, tasks, along with feature and support requests. SourceForge also interfaces with several popular source control protocols, Subversion (SVN) and Concurrent Versioning System (CVS). (12)

SourceForge is available in both a free and enterprise version of its software. The free version of the software, which is provided by SourceForge, Inc., can be accessed at SourceForge.net or downloaded and privately hosted with a limited amount of users. The free version is not packed with as many features and does not provide as much expandability. The enterprise version, which is provided by CollabNet, is more secure along with having a more robust feature set and is geared toward a larger user base.

CollabNet also provides an API with its enterprise version which allows for external applications to access the features of SourceForge. (8)

Eclipse

Eclipse is an open source integrated development environment originally created for java development. In November 2001 Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and Webgain all formed the Eclipse.org board of stewards. By the end of 2003 the group had grown to include over 80 members. By the end of 2004 Eclipse had been reestablished as a not for profit corporation.

Eclipse at the present allows for extensions to be downloaded to support many other languages, such as Ruby, C++ and others. Eclipse also offers an extensive plug-in framework so that individuals or groups can expand upon the current framework to add functionality that they feel needed. Many of these plug-ins are available online for download. Eclipse even offers a tool within itself to manage the plug-ins. This expandability has made Eclipse a very useful tool for many developers, and with coming new versions, and new plug-ins should allow for that usefulness to grow. (8)

Webfoot

Webfoot is a plug-in for the Eclipse integrated development environment. The main source for development of Eclipse is through projects done at Worcester Polytechnic Institute with Professor Gary Pollice. Webfoot allows for integration between Eclipse, and the collaborative development website SourceForge along with the incorporation of a few other features as well. Webfoot provides many of the features that are available on

SourceForge to be available right in Eclipse via different views. Some of these features include task management, viewing of trackers, and with the addition of our project, the viewing of documents. Other features that are either being added, or are already added to Webfoot are geared towards enhancing the communication between developers that are working on a project. These features include, but are not necessarily limited to, a screen sharing ability, an instant messaging ability, and a voice chat ability as well.

These tools allow a developer to stay within their development environment for all their tasks. They can do everything from reading and writing code, to reading documents related to the projects instead of have to constantly switch between a browser and Eclipse.

3. Methodology

This project intended to incorporate the ability to view, edit, and update documents between SourceForge Collaborative Development Environment and the Eclipse Integrated Development Environment. Based on the Webfoot Project, the team developed an Eclipse plug-in that replicates the SourceForge CDE user experience within Eclipse.

Goals and Objectives

In order to fulfill the goals of the project, the team completed a number of objectives. Our methodology is represented in Figure 3 below:

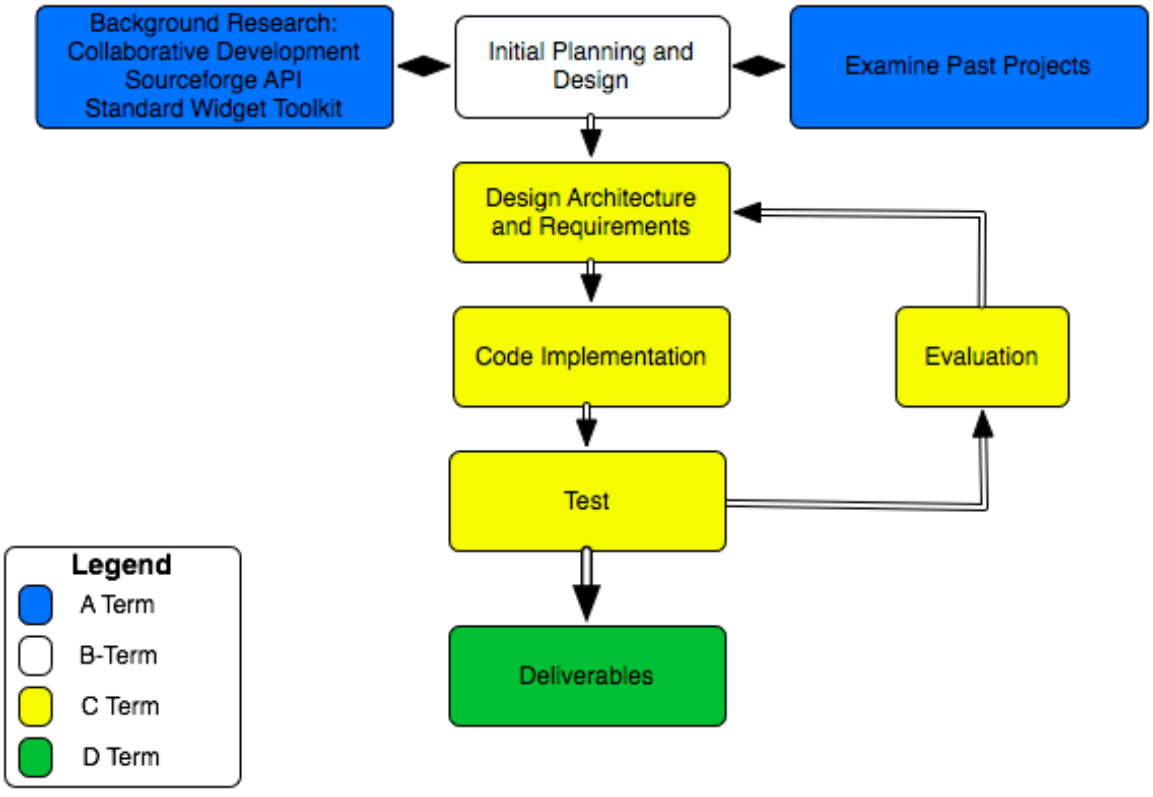


Figure 3: Project Flow Chart

As shown in the project flow chart, the team completed the following tasks in order to develop the plug-in for Eclipse:

- Initial Planning and Design
    - Understand the needs of collaborative development
    - Research SourceForge API and Standard Widget Kit
    - Examine similar past projects
    - Create initial implementation design
- Begin final design and Implementation
    - Establish iterations and implement
    - Test completed work
    - Evaluate work and re-iterate
- Create deliverable in the form of an Eclipse plug-in, help, and a paper

With our design and implementation complete, the Webfoot Document Viewer team can incorporate our addition into the Webfoot plug-in. The timeline below shows the schedule we followed:

Goal	**Term**			
	A	**B**	**C**	**D**
Research Collaborative Development and Past Projects				
Design System				
Iterative Development				
Package Deliverables				

Figure 4: Schedule

Implementation

The team worked from August 23, 2007 to October 11, 2007 researching information concerning collaborative development, Eclipse plug-in development, and the specific design requirements for our project. Initial development took place October 23, 2007 to December 13, 2007. In this time, we created initial packages and source on which all the required features could be implemented. The bulk of development took place from January 10, 2008 to February 29, 2008.

Implementation Polices

Before development could begin, preliminary requirements were created to get a better understanding of the project. We analyzed what actions SourceForge allowed a user to perform. With this analysis, the team created a detailed set of user stories. The users stories allowed us to establish estimates on how long certain features would take to implement. The team realize that there was not enough time to implement everything SourceForge provided. We thought it would be important to spend more time on a small number of core features than to spend less time on a large number of features that might get half implemented. In order to do this, we needed help determining what these core features were.

Without a physical customer, we used feedback from Professor Pollice and the other Webfoot developers on what features would be important to the overall user experience of the plug-in. After this brainstorming, the team had a much better idea as to what features should be implemented. We combined this information with the our earlier prioritization to create a final list of what we planned on doing. The team utilized

iterative development cycles in order to complete these objectives. Although the team designed the architecture of the plug-in before beginning development, it never stopped designing. These development cycles would allow us follow the principles of extreme programming and agile development that we learned in classes such as Software Engineering. Each of these cycles would contain design, development, testing, and evaluation of the work.

At the end of each stage, we tested the development that was done to ensure that the new features worked. This involved going through all of the use cases and user stories to ensure that no bugs existed in the new code and that no regressions were introduced. The team believed that the software should always be in a working state.

At this point, the team evaluated the iteration. Success of the iteration depended on whether we underestimated or overestimated the amount of time that features took. If a certain task took too long, and a similar task still needed to be completed, we could tweak the next iteration to allow for more time. We also looked at any regressions that took place and the root cause behind why these new bugs were introduced.

Shared Libraries

The team used several shared libraries in order to complete our objectives. We implemented the user interface using the Standard Widget Toolkit to stay consistent with other Webfoot plug-ins and Eclipse. Interaction with SourceForge was handled using the SourceForge 4.3 jar. The following sections document the detailed responsibilities of each shared library.

SourceForge API

In order to interact with the SourceForge Collaborative Development Environment, our Eclipse plug-in utilized SourceForge Application Programming Interface jar file provided by Collabnet. The SourceForge API offers developers access to various managers that can perform functions pertaining to tasks, documents, users, and other aspects of the environment. Our plug-in implemented the document manager. The document manager allows an authenticated user to create, update, and deletes documents. The document manager also manages the creation and deletion of folders.

Every time a user interacts with the SourceForge CDE through our plug-in, or any plug-in utilizing the SourceForge API, the manager creates a request to encapsulate the action. This is done using Simple Object Access Protocol (SOAP). SOAP creates XML requests with information on what the user wants to do along with the data required for that specific action. This is all performed over the hypertext transport protocol (HTTP).

In addition to the document manager, the plug-in also incorporates the SourceForge file manager. Anytime a file, document or not, needs to be downloaded or uploaded the SourceForge file manager needs to be used. These two managers interact with each other when creating, updating, and viewing of documents. For example, when creating a new document, the manager uploads the file, and returns a file identification string that is specific to that file and the user's session. This file ID is then used when calling for the creation of this new document. All of these managers and classes were implemented in the SFCore in Webfoot.

SourceForge Documents

In the SourceForge API, SourceForge documents are represented by a DocumentSoapRow. The DocumentSoapRow holds information about documents including file name, file size, identification string, and version. In order to encapsulate this representation, the team created a class called SFDocumentItem. By encapsulating a DocumentSoapRow in a SFDocumentItem we were able to add more functionality. This allowed us to store the text, url, or file associated with the document along with who locked it and the file ID that is unique to the session and File on the server.

The team also created a SFDocument which encapsulates a SFDocumentItem and Document Manager. This was done so that SFDocumentItems could have an associated manager that could be bundled together in case other plug-ins wanted access to this information.

The team also implemented classes for Folders. Folders are represented by the SourceForge API as DocumentFolderSoapRows. The class SFDocumentFolder encapsulates these details. From this class, the documents and sub-folders of a folder can be determined.

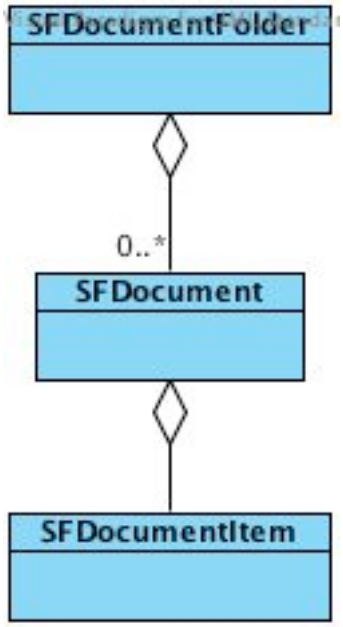


Figure 5: SFDocument, SFDocumentItem, and SFDocumentFolder

Document and Folder Manager

In order to view, edit, or update a document, the Document Manager had to be created. The SourceForge API provides SOAP web services that can be instantiated. The SFDocumentManager is a singleton that provides an interface to the IDocumentAppSoap. This ensures that only one instance of the SFDocumentManager is created at any given time. The SFDocumentManager provides access to only some of the IDocumentAppSoap functionality. The manager can create new documents, update existing documents, and delete documents. Due to the time constraints on the product, some of the other functionality such as finding documents wasn't implemented. The IDocumentAppSoap is also responsible for managing folders in SourceForge. This functionality was excluded from the SFDocumentManager and spun off into the SFDocumentFolderManager.

Similar to the SFDocumentManager, SFDocumentFolderManager is a singleton to ensure more than one instance isn't created. A singleton instantiates one instance of the class the first time an object of its type is created. All subsequent new operator calls to this class are given this instance. The team separated the SFDocumentFolderManager from the SFDocumentManager in order to make the architecture less fragile. The manager can create, modify, and delete folders. Added functionality was added such as getting a list of the all of the folders and documents under a folder. This was used extensively when building our View.

File Manager

The SFDocumentManager and SFDocumentFolderManager allow for creating and editing documents and folder but do not handle viewing or uploading files or text from the user. SFFile was created in order to upload and download files. When downloading files from SourceForge, the document Manager provides a fileID that is valid only for that user and that session. SFFile allows the authenticated user to download this file. SFFile is also responsible for uploading files.

Standard Widget Toolkit

The Webfoot Document Viewer plug-in would not be useful without the ability to actually view and interact with the documents and folders present in the SourceForge CDE. The Eclipse Platform takes advantage of the Standard Widget Toolkit. SWT is an alternative to AWT and Swing. Swing provides a consistent interface across multiple platforms all using java code. AWT and SWT are similar in that they both provide access to the platforms native widgets. However, they approach this in different ways.

AWT allows for inconsistencies because not all widgets are available on all platforms. The team chose SWT to get around these issues along with staying consistent with Eclipse.

The various GUI elements that make up the document viewer are controlled by the DocumentView class. The DocumnetView class is responsible for creating the document selection pane from which all other features are executed. Based on the what the user does, DocumentView calls the corresponding action. Each action can then call other GUI elements such as dialog boxes. For example, by double clicking on a document in the selection pane, the MakeDoubleClickAction is called. This action opens the active version of the document in Eclipse.

The DocumentView class is responsible for creating the two main aspects of the Document View plug-in. The first is a selection tree from SWT that allows documents and folder to be represented by parents and children. From the selection pane, users can call actions using a contextual menu or through double click. The other main aspect of the document viewer is the details pane. The details pane displays document details when a document is selected from the tree. The selection tree is shown on the left Figure 6 below, while the details pane in shown in the right:

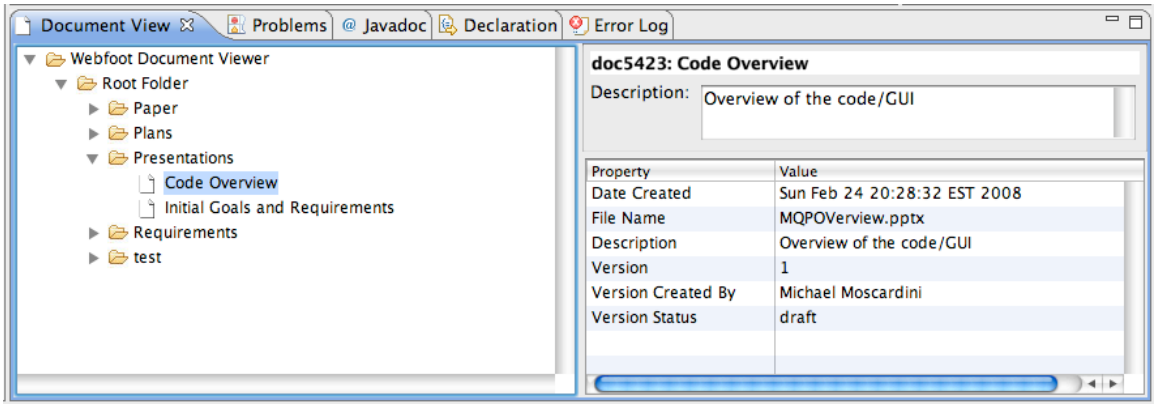


Figure 6: Document View

Eclipse Core

Eclipse Core is the foundation of the Eclipse platform. Of its many features, we focused on its ability to provide developers with tools to properly manage Eclipse plug-ins preferences. The use of preferences in the current plug-in is limited, however the core allows for a flexible method for adding new preferences.

When files are downloaded, a user is asked where they want to save the file. This becomes tedious to the user. To get around this issue, we implemented a preference store. A preference store is a mechanism provided by the Eclipse core that allows developers to reliably save information within a workspace. In this case, the user can specify where they want to download the document every time they open one, or choose a default folder.

There are several components needed to implement a preference store. The first is the DocumentViewPreferenceInitializer. Every plug-in is given a preference store. The DocumentViewPreferenceInitialize is tasked with recovering the preferences from this store and setting the default values. When preferences are first read, if they contain null values, the preference initializer sets them to default values.

In order for preferences to be viewed and/or changed in the Eclipse preference workbench, a preference page needs to be created. The DocumentViewPreferencePage draws the radio buttons and other widgets used in preference configuration. The preference page allows for increased flexibility. For example, if there are no preferences are set when a user downloads a file, the DocumentViewPreferencePage can be loaded

into a workbench and displayed to the user. At the same time, this preference page can be included in the main Eclipse preference workbench with the other SourceForge Webfoot items. Therefore, there is no code duplication in making two preference windows.

Actions

Actions are performed by the system to complete certain tasks. When a user triggers an event, such as double clicking a document in the selection pane, a double click event occurs and the MakeDoubleClickAction is triggered. Actions in this plug-in are triggered either using event listeners or the contextual menu in the selection pane.

During the design stage, the team examined other projects that brought SourceForge functionality into Eclipse. We wanted to understand how they implemented similar features. Their implementation could then be scrutinized to find what worked well and what didn't. One of the issues was with the DocumentView class. With other projects, all of the actions were defined within their respective *View class. With our project implementing so many actions, this file soon became unmanageable. Another issue is that several of the actions were based on the same code which caused unnecessary code duplication.

To remedy the situation, the team created an actions package. For every action that would be called, a class extending Action was created in the package. This shrunk the size of DocumentView class substantially. For the actions that shared code, we created an abstract class that they extend. The actions used in the plug-in are shown in Table 1 below:

Action	Class	Description
Connect to SourceForge	MakeConnectAction	Provides dialog to connect to SourceForge.
Delete a document	MakeDeleteDocumentAction	Deletes a document selected from the selection pane
Delete a document folder	MakeDeleteDocumentFolderAction	Deletes a folder selected from the selection pane
Double click on the document	MakeDoubleClickAction	Extends AbstractOpenAction. Opens the document in Eclipse
Create a new document	MakeNewDocumentAction	Opens a dialog to create a new document in the folder selected in the selection pane
Create a new document folder	MakeNewDocumentFolderAction	Opens a dialog to create a new folder in the folder selected in the selection pane
Open the document	MakeOpenDocumentAction	Extends AbstractOpenAction. Opens the document in Eclipse
Select a different version	MakeSelectionVersionAction	Updates the details pane to show information for the new version
Select a project	MakeSelectProjectAction	Provides a dialog to select a project
Update the document	MakeUpdateDocumentAction	Provides a dialog to update the document

Table 1: Actions

Milestone One

In order to add any functionality, the user has to be able to view the documents on SourceForge. The team decided that adding a document tree would be the focus of the first milestone. The plug-in was to be contained inside an Eclipse view which could be displayed and hidden by the user as desired. This tree allowed for more functionality to added later on in development and provided a flexible base in which to add features.

Additionally, this is the same method that the web interface for SourceForge uses. Therefore, we felt that users would be familiar with the concept of the tree.

The design of the tree is based on the tree used by the Webfoot Task View but modified slightly. The tree consists of a number of tree objects. For the Document View, a tree object can either be a tree parent, which represents a document folder, or a tree leaf which represents a document. A tree parent contains any number of tree objects. This design allows us to have folders that contain not only documents, but also other subfolders. The Task View team chose this design because it follows the Model View Controller design pattern. The relationship between Tree objects is shown in Figure 7 below: (13)

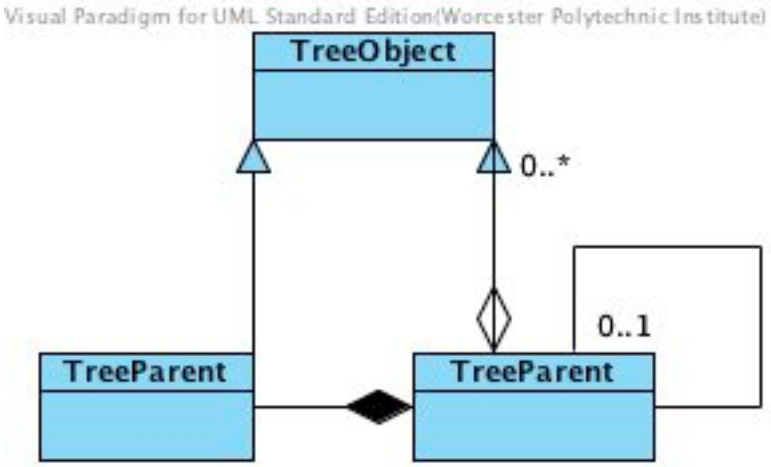


Figure 7: Tree Objects

Because of the importance of this milestone, no work was split up among the team. The document view was created during a number of pair programming sessions in which both team members took turns driving the sessions. We created a document tree during the first milestone.

Milestone Two

For the second milestone, the team did two things. The first was the addition of a contextual menu to the document tree. The contextual menu would be the base in which all actions would be called from. The second was a details pane that would show important information about the selected document.

Instead of implementing buttons that a user could click on, the team implemented a contextual menu for executing actions. This was done for several reasons. Because there are different actions that would be needed depending on whether the user was selecting a folder or a document, we felt that there would have been too many buttons. Some of the buttons would have been inactive most of the time, just taking up space. We also chose to use a contextual menu to stay consistent with the other SourceForge Webfoot plug-ins.

When a user right-clicks on a document or folder, a listener draws the contextual menu and adds the desired actions. Different actions could be added depending on whether the user was right clicking on a document or a folder. There were only three actions attached to the contextual menu in this milestone. We wanted to be able to connect to the SourceForge server, select a project , and refresh the view all from the contextual menu.

The second feature we added is the details pane. The details pane displays the title, description, file name, date, and all of the other document details that are present in the web interface. Again, we based this code off of the Task View details pane. A table was created in which a content provider would display information about the document given in the details pane. A selection listener was attached to the document tree. The contents of the details pane changes depending on this selection. We made a modification to the

Task View's implementation. Our details pane stores a copy of the document that is selected. This is used in a later iteration when we wanted to be able to select different versions of the document. This design choice will be explained further in section Milestone Five.

Milestone Three and Four

With a contextual menu in place, we began to add more advanced functionality. This included the ability to create and delete document, create and delete document folders, and opening documents. Because of some technical issues that we will discuss later on, this development cycle ended up being extended by two weeks due to bugs beyond our control.

Early on, we decided that our project should remain consistent with the SourceForge web interface whenever possible. Therefore, we designed the new document dialog and edit dialog based off of the web interface. We wanted them to look as similar as possible so that users wouldn't be confused. In order not to duplicate code, an abstract class was created to handle this task. We then extended it with a create dialog and update dialog.

Creating documents proved not to be very difficult to implement. However, we ran into several issues trying to implement an update of a document. Because we followed iterative development, and the practice of always having working code, the plug-in itself continued to work without this new functionality while we investigated the problem. Through this investigation we found the issue was with the SourceForge API. There was no valuable documentation given on how to update a file already on SourceForge. We attempted to contact Collabnet through their support forums, but we were only met by

other developers with the same questions we had. Eventually, through much trial and error, we fixed the issues.

With the ability to create documents and post updates in place, we wanted to be able to open the documents on SourceForge. We agreed on two ways to do this; with a contextual menu and with the double click action. Because there would be a lot of duplicate code, we created a abstract class AbstractOpenAction that handled the downloading of the file. Selecting the file that is double clicked or right clicked is handled by the classes extending it. A SWT directory chooser was added so that you could specify where the file would be downloaded to upon executing the action.

When a user downloads a file and the file type is supported by an Eclipse editor, we choose to have the file open in Eclipse. This development was done on Mac OS X. When a team member running Microsoft Windows tried to test this feature, we soon realized that there was an issue opening Microsoft Word documents inside the Eclipse editor. This issue was not present on the Mac. We were able to track the bug down through the Eclipse Bugzilla Bug tracking system(Eclipse bug 206752). This was a regression that was introduced in Eclipse 3.3. This bug was fixed in Eclipse 3.4M4, with the stable version of Eclipse 3.4 slated for a June 2008 release.

Because of the issues trying to implement these features, this milestone ended up being extended into two milestones. We felt that this was necessary in order to ensure that our problems didn't cause us to miss other issues and that the plug-in met our standard of always working.

Milestone Five

Originally, the team planned this milestone to be short. We wanted to create a way for a user to view document details of a different document version, along with opening a different document version. However, when demonstrating file downloading to the weekly project meeting, many of the students thought that users should be able to select and save the path that files are downloaded to. The need for preferences had not been considered before and therefore no design had been made. The team agreed with this feature, and we added it to this milestone.

The team thought the version sub menu would be an extremely useful feature. It allows the user to change which version of the document the view is looking at. Earlier, we had decided that the details pane would store the document it was displaying. Because the details pane was based on a selection listener, anytime it was clicked, it would load the details pane again. This made it difficult if the user wanted to right click on a file to change the version. To remedy this problem, any time a document is selected, the document id is compared to the one present in the details pane. If they are the same, then no update is performed on the details pane.

4. Results and Analysis

The team produced is a fully functional Eclipse plug-in that incorporates a view within Eclipse. The plug-in allows the same type of document interaction that is available on the SourceForge website. The main features implemented in this plug-in are:

- The ability to view the document tree for the selected project.

- The ability to view document details of the selected document.
- The ability to open a document in the document tree.
- The ability to add, delete and edit a document in the document tree.
- A contextual menu containing when a document or document folder is right-clicked in the document tree.
- The ability to select a version of the document from the contextual menu.

When implementing the features for the Document View plug-in, the team wanted congruency to the interface on SourceForge web interface. Figure 8, below, demonstrates the consistencies between the dialogues we developed for the project and the ones on SourceForge. We felt that these similarities would provide a more usable, as well as a more intuitive interface for the user. The similarities also allow for a smoother transition from using the web interface on SourceForge to using the Document View plug-in.

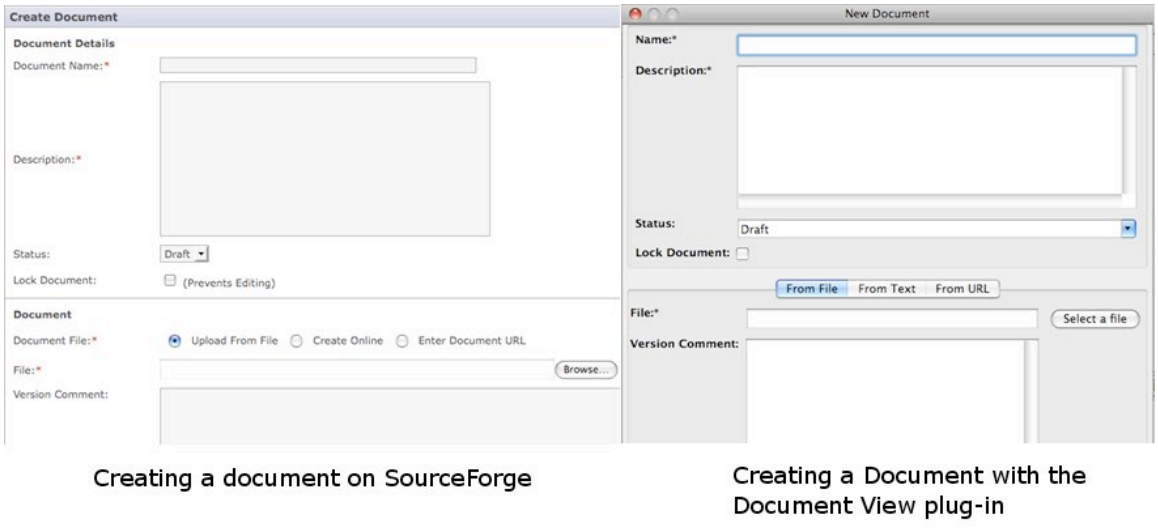


Figure 8: Interface Comparison between SourceForge and the Document Viewer

During the development of the project we also took into account existing Webfoot conventions. The major consistent elements between the other functions of Webfoot are the SourceForge connection dialogue and the SourceForge project selection dialogue. We took these elements into account during the development of our plug-in. The connection dialogue prompt and the project selection dialogue accessed through the document viewer are congruent with the dialogue prompts in other parts of the Webfoot project. We felt that maintaining this standard was very important to the overall usability of both our plug-in and the Webfoot project as a whole.

To further augment the usability of the Document View plug-in, a help section for the Document View plug-in has been added to Webfoot's help file. The help file details how to use the various features of the plug-in, such as adding or deleting a document, selecting a document, accessing the view, etc. The purpose of the help section is to provide a comprehensive guide to the plug-in, and to introduce the user to the ins and outs of the plug-in.

Metrics

In order to capture the scope of our project we gathered metrics on the code base of the project. The tool we used to do this is the Eclipse Metric Plug-in. The plug-in has a view that allows for details about a package or source code file to be displayed. The gathering of these metrics allowed us to look at the project in a more quantitative way.

In total the team wrote 2776 lines of code. This code was spread over 7 packages, with 44 classes, containing 328 methods. When taken into the perspective of the 14 weeks of total development time focused on writing the code, our code velocity was

roughly 200 lines of code per week. This number does not accurately reflect the total amount of work put into the project. Code velocity does not take into account the time spent wading through the SourceForge API, or the time spent refactoring and debugging the code, etc. Although the code velocity does not accurately describe the amount of work put in on the project, it still gives a perspective on total scope of how much usable code was written over the length of our project.

Webfoot Beta Release

Real world testing of a software project is a very important aspect of the development process. This testing allows for real people to use your software and find any bugs, be they large or small, with it. The original target date for a Webfoot beta release including the work of the four 2007-2008 Webfoot projects was scheduled for early April. The beta missed its mark by about month due to a few bumps along the road, between merging issues, and the occasional show stopping bug with the other Webfoot additions. Those issues are all, as of now, resolved. A beta release that includes the work from the 2007-2008 Webfoot projects should be out by the end of April 2008.

With the lack of a beta release to provide some real world feedback on the Document View plug-in our team relied on testing the plug-in against the specification. Whenever new features were checked into the code, all operations that were required by the use cases and user stories for the project were tested by hand to make sure that the functionality was still intact after the code change. Using this method we were able to find if any problems had been introduced in the new code base. Although there is no substitute for real user testing, this method did facilitate the finding of quite a few bugs in our code.

5. Future Work

The team created User stories as its first task before it began development. When the project completed, the team analyzed these user stories to gauge the project's success. Through this process, the team discovered several areas of the project that could be improved or expanded. Some of additions are features that we were not able to complete. However, several of these additions are features that we realized would be beneficial to the overall usefulness of the plug-in.

Using our discoveries, the team has created a set of recommendations for future work. These recommendations are broken up into two categories. The first set cover functionality present in the SourceForge web interface that is not present in the plug-in deliverable. The second set of recommendations give suggestions on how to improve the ease of use and robustness of the user interface.

SourceForge Functionality

SourceForge provides a wide range of functionality through its web interface and supporting java API. Throughout our iterations, the team tried to determine which of these features were the most important to our objective. Most of these features were implemented in the plug-in, however, a few were left out. These features would be a worthwhile addition to the current plug-in's feature set.

In addition to version control, SourceForge provides two other tools for documents: Associations and Reviews. Reviews allow a document to be tagged for review by another SourceForge user with a corresponding due date. The team believes that the ability to add reviews through the plug-in would be beneficial to the user. This could be

accomplished by assigning a review to an existing document, or even from a code snippet in an open source file. In addition to creating reviews, it would be beneficial for a user to be able to see what reviews have been assigned to them.

Associations allows a user to connect an action performed in SourceForge, such as committing a file to source control, to a document in SourceForge. A user could make a commit into source control, and assign an association to the commit all within one Eclipse window.

On larger projects, there is the possibility of having hundreds of different documents each with many versions. The SourceForge web interface provides the ability to search for different documents and versions. A user can search within a document given keywords, search by who created or edited the document, and even by date. Adding this ability to the plug-in, would further reduce the need for ever opening the web interface.

Users who want to watch a document or a folder, can monitor it using the web interface. By monitoring it, the user is notified by email if any actions are performed on the document or folder. This feature is not currently accessible by the using the plug-in and must be done through the web interface.

UI Functionality and Enhancements

The plug-in interface provides a document folder tree where users can right click on an item to perform an action. For example, if a user wants to add a document to SourceForge, they right click on a folder and select “Create a new document”. The dialog presented allows the user to create a document from a file on the users machine.

However, the plug-in has no concept of that documents are located in the user's Eclipse workspace.

Eclipse provides a sub-system for handling drag and drop between views. This could be incorporated into the plug-in. This would allow the user to drag a document from the Eclipse package explorer and instantly create new document by releasing it on a folder in the document view, or by creating an update of a document by releasing it on a file. In addition to drag and drop support, the ability to right click on a document in package explorer and add the document for SourceForge should be added.

The document view allows for one project to be opened and viewed. If the user wants to view another project, they have to close the current project and select a new one. The ability to view multiple projects should be added in the future. Not only will this allow the user to look at multiple project, but also added functionality. For example, with multiple projects, the user might want to copy a document from one project to another. This addition would make that possible.

6. Conclusions

The importance of collaborative development tools continue to grow as technology allows teams to become further spread apart and less reliant on physical interaction. In the not too distant future, it's conceivable that members of the same team may never meet in person while working on the same project. Development tools such as Eclipse and SourceForge allow for this progression.

Eclipse and SourceForge provide mechanisms for users to add or enhance functionality. Eclipse provides developers with powerful tools for the creations of plug-

ins. Plug-ins are added to applications to add functionality instead of the user being forced to use another tool. SourceForge provides an API that allows developers to interact with their system in a customizable way.

To aid the paradigm shift toward more flexible collaborative development, the Software Tools and Technology Group at WPI along with other projects under Professor Pollice have utilized these methods to develop Webfoot. Webfoot provides collaborative development tools such as interaction with SourceForge tasks and user stories all from within Eclipse. This allows teams to shed the clutter of having to use different tools for every task.

The team used its understanding of agile software development over the course of three terms to successfully develop the Document View plug-in. This plug-in for Eclipse allows users to create, edit, update, and download documents managed by SourceForge in a transparent manner. A user can manage documents without needing to open any application other than Eclipse.

Appendix A Plug-in Help for Webfoot Document View

Webfoot Document View

Welcome to the WPI Environment Built For Object-Oriented Teams (WEBFOOT) Document View.

Abstract

The plug-in allows for the viewing and manipulation of the documents on SourceForge from within Eclipse. The plug-in created uses the Webfoot project at its base for connecting to SourceForge, and as such is part of the Webfoot project itself. The project allows for documents to be created, deleted, and upload, it also allows for the viewing and editing of documents details, such as version and status.

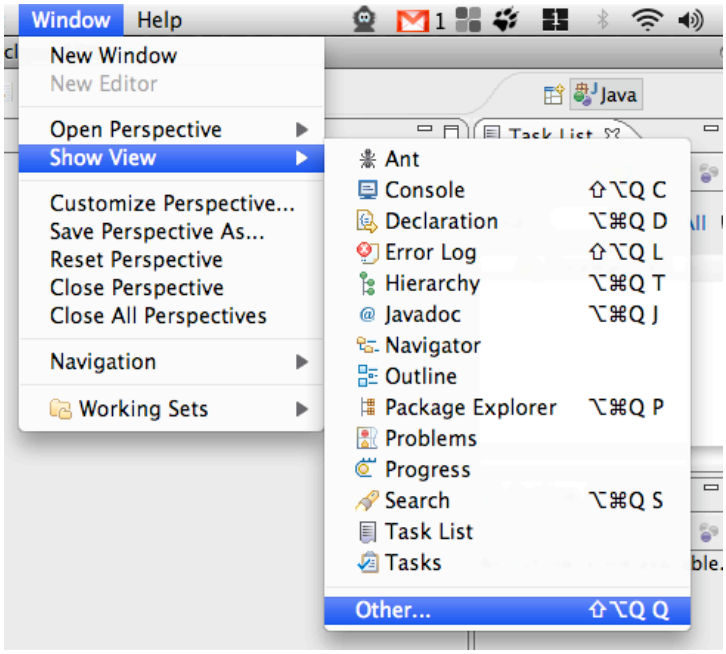
Getting Started with the Document Manager

This section will cover opening the Document View to use in Eclipse as part of the Webfoot project.

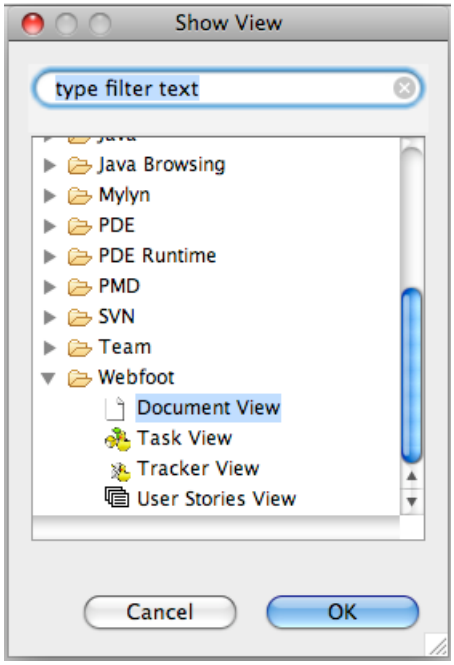
Opening the Document View

With the Webfoot plug-in installed and viewing in Eclipse:

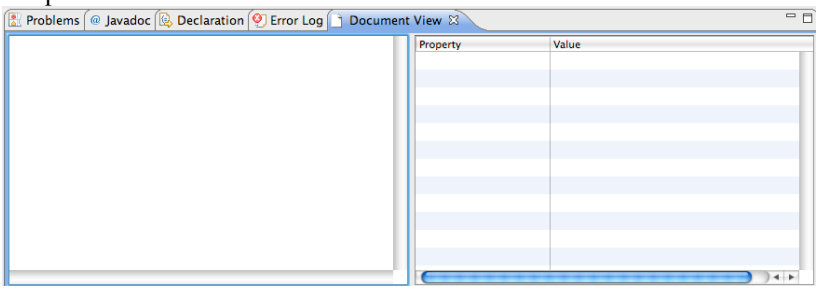
1. Go to Window -> Show View -> Other...



2. Select Document View under the Webfoot Folder and press "OK"



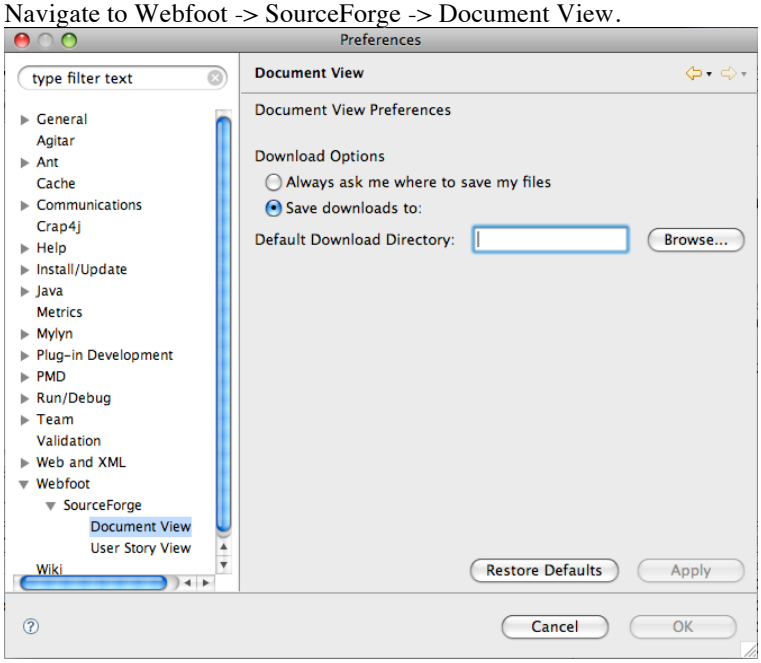
3. Eclipse should now look like this:



Configure Preferences

The Document View preferences allow you to designate a default download folder. You can also choose to select a path each time you open a document.

1. Open the Eclipse Preferences
2. Navigate to Webfoot -> SourceForge -> Document View.



Download Options

There are two options for downloading files. If you select "Always ask me where to save my files" you will be prompted every time you open a file where to save it. You can also set a default path to save your downloads. One of the must be selected.

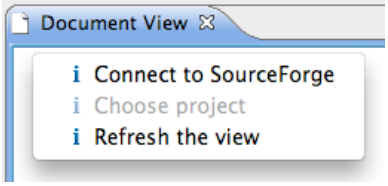
Connecting to the Document View

This section will cover connecting using the DocumentView. While it is possible to connect using the Webfoot plug-in as well as other Webfoot plug-in views, it is also possible to connect using the Document View.

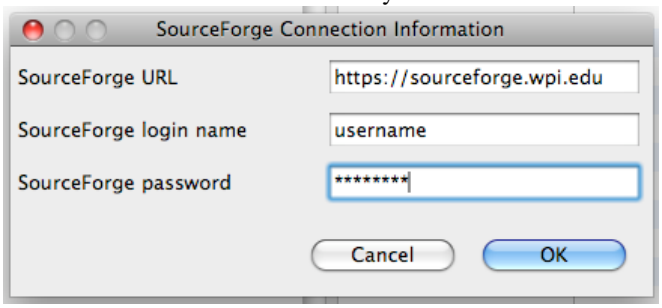
Connecting

1. Right click on the Document View window

2. Select "Connect to SourceForge" from the Contextual Menu

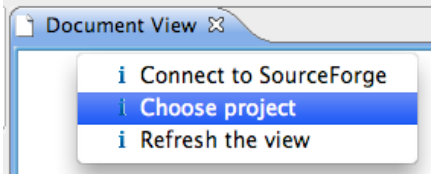


3. Fill in the SourceForge URL, your login name, and SourceForge password in the dialog as shown below. Some of the information may be filled in with default values taken from your preferences.

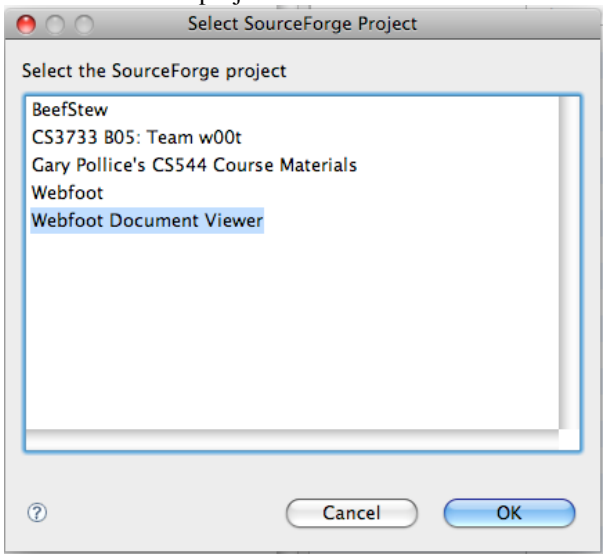


Select "OK" to login

4. Right click on the Document view
5. Select "Choose project" from the Contextual Menu



6. Select the desired project and Press "OK"



You are now ready to start using the Webfoot Document Manager

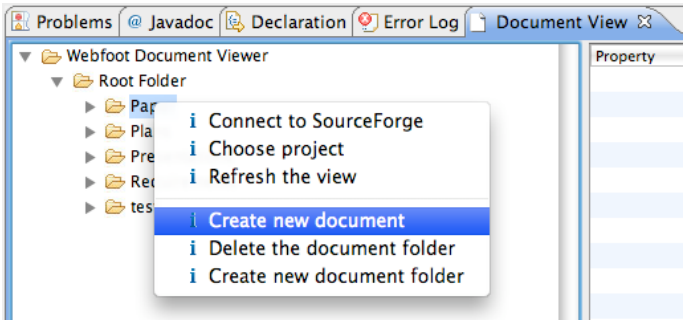
Creating a new Document

This section will cover creating a new document to be uploaded to SourceForge

Opening the Wizard

After connected to a project, do the following:

1. Right click on the folder you wish to house your new document
2. Select "Create a new document" from the Contextual Menu



3. Fill in the required fields: Name and Description. All other fields are optional

File Origin

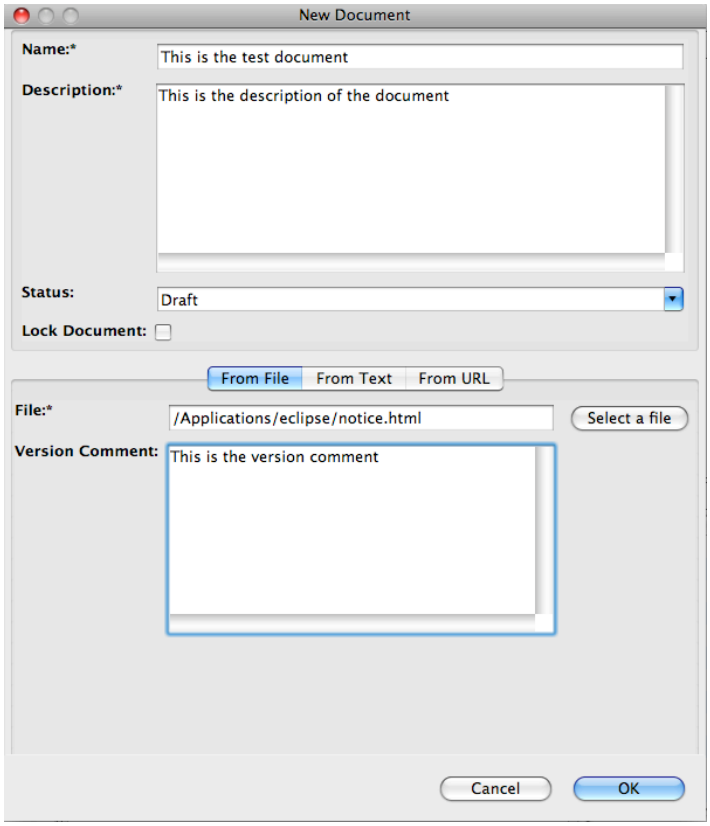
There are three methods for creating new files: From File, From Text, and From URL

From File

This method uses a file chooser to find the file that you want to upload.

1. Select the "From File" tab

2. Press the "Select a file" button. Navigate to the file you wish to upload and press ok. Your screen should look like this



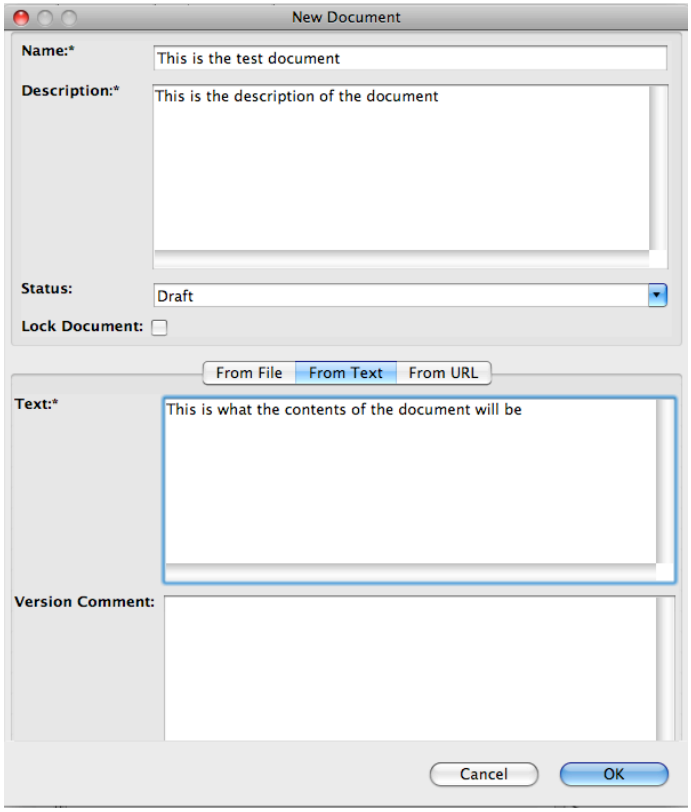
3. [OPTIONAL] Fill in version comment
4. Press the "OK" to create the new document. If no error appears, the document is now on SourceForge.

From Text

This method gives you a box that you can type directly into. The document will be created with that text.

1. Select the "From Text" tab

2. Fill in the Text box. Your screen should look like this



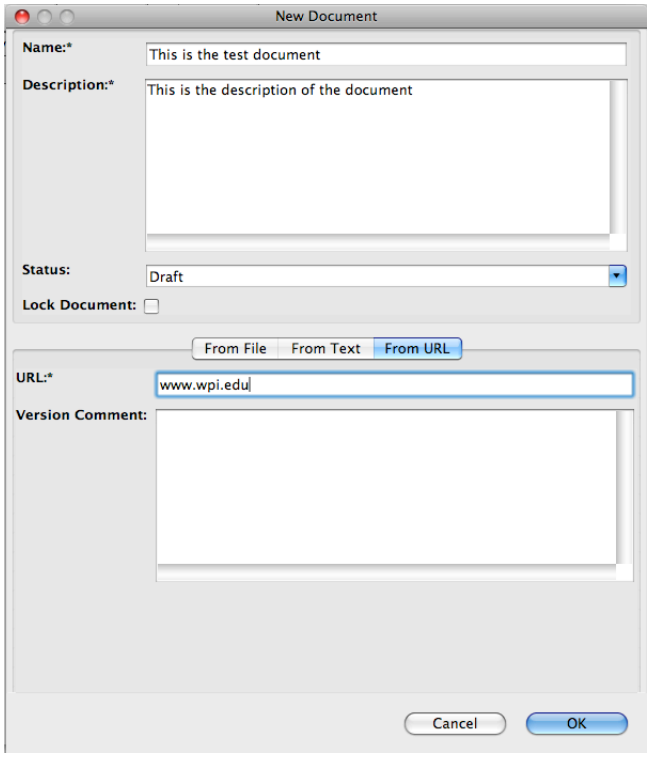
3. [OPTIONAL] Fill in version comment
4. Press the "OK" to create the new document. If no error appears, the document is now on SourceForge.

From URL

This method allows you to provide a URL. The document will be a link to that URL.

1. Select the "From URL" tab

2. Fill in the URL field. Your screen should look like this



3. [OPTIONAL] Fill in version comment
4. Press the "OK" to create the new document. If no error appears, the document is now on SourceForge.

Updating a Document

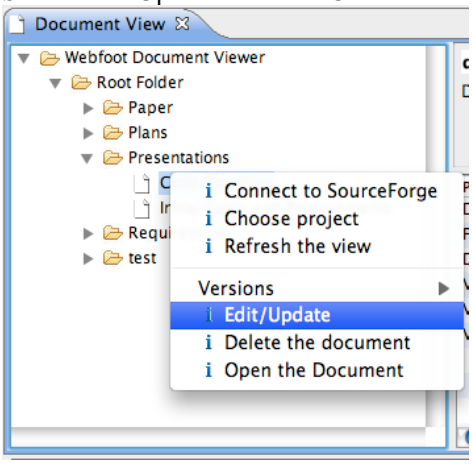
This section will cover updating a document that resides on SourceForge

Opening the Wizard

After connected to a project, do the following:

1. Right click on the file you wish to update

2. Select "Edit/Update" from the Contextual Menu



3. [OPTIONAL]Update the required fields: Name and Description. All other fields are optional

File Origin

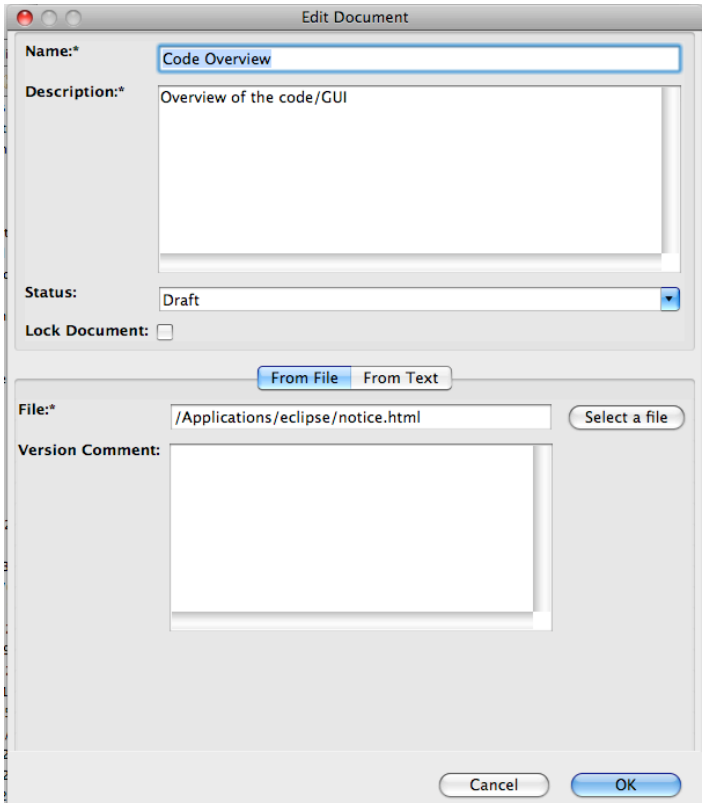
There are two methods for updating files: From File, From Text. From URL was removed due to issues beyond our control.

From File

This method uses a file chooser to find the file that you want to upload.

1. Select the "From File" tab

2. Press the "Select a file" button. Navigate to the file you wish to upload and press ok. Your screen should look like this



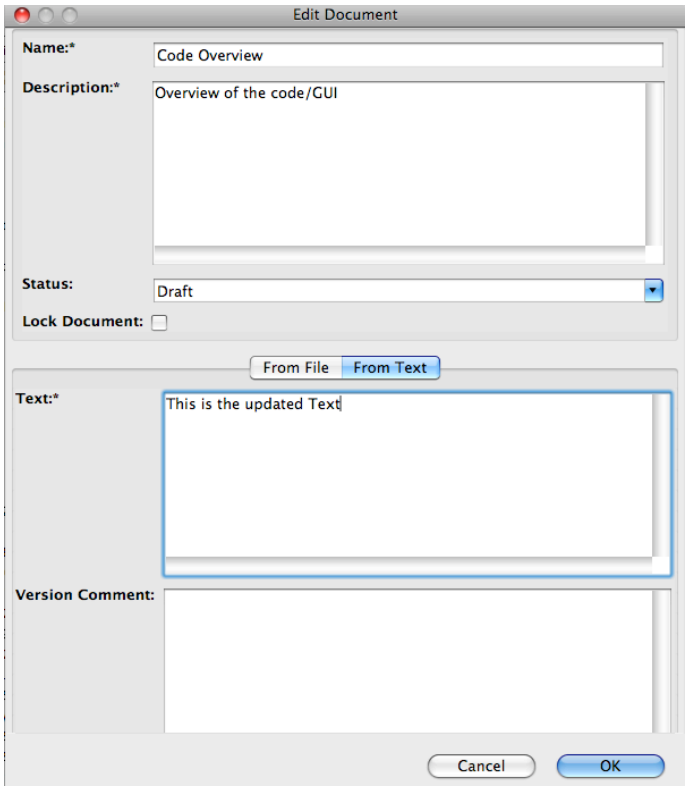
3. [OPTIONAL] Fill in version comment
4. Press the "OK" to update the document. If no error appears, the document is now on SourceForge.

From Text

This method gives you a box that you can type directly into. The document will be created with that text.

1. Select the "From Text" tab

2. Fill in the Text box. Your screen should look like this

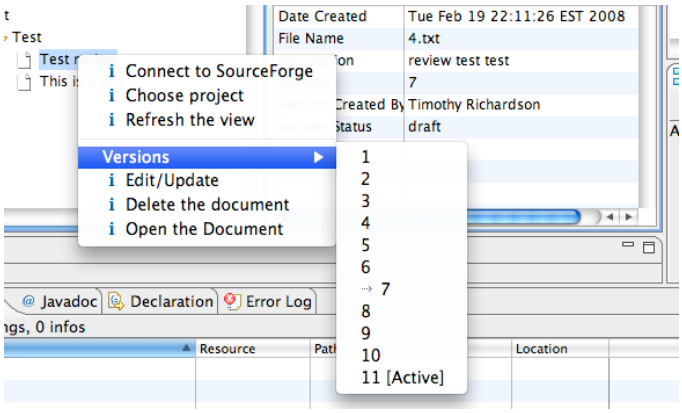


3. [OPTIONAL] Fill in version comment
4. Press the "OK" to update the document. If no error appears, the document is now on SourceForge.

Selecting a Version

This section will cover viewing a different version of a document

1. Right click on the file that you want to select another version of.
2. Select the Versions sub-menu from the Contextual Menu.



The arrow ⇢ indicates the version of the document you are viewing. The active tag [Active] indicates the version that is set as active in SourceForge. The active tab cannot be changed though this plug-in.

3. Select the version you want to view. This will update the document details pane. Now, when you open the document, it will open the version shown in document details pane.

Opening a new Document

This section will cover opening a document. There are two ways to open a document. By Double clicking it, or by contextual menu.
Files are opened in Eclipse when possible. Make sure your Download Options are set in your Document View preferences. For more information about preferences, please visit Getting Started .
IMPORTANT NOTE: Due to a bug in Eclipse 3.3, some files, such as word documents, may not open in Eclipse correctly. This bug has been fixed in Eclipse 3.4M4+. If you receive an error trying to open a document in Eclipse, locate the file on your system and open it manually using the appropriate application.

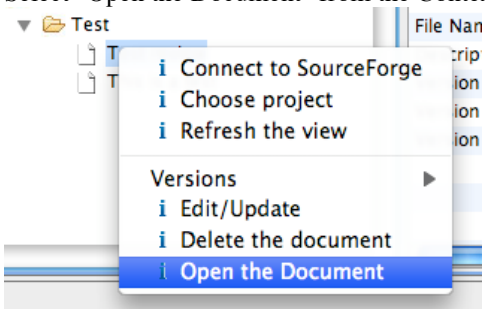
Double Click

1. Double Click on the document you want to open. This will open the version of the document you are viewing in the details pane. If no document is in the details pane, the latest version will be downloaded.

   If you have not set up your preferences, a preference window will appear. For more information about preferences, please visit Getting Started . If your preferences are set up, those setting will be respected.

Right Click

1. Right click on the file you want to open.
2. Select "Open the Document" from the Contextual Menu.

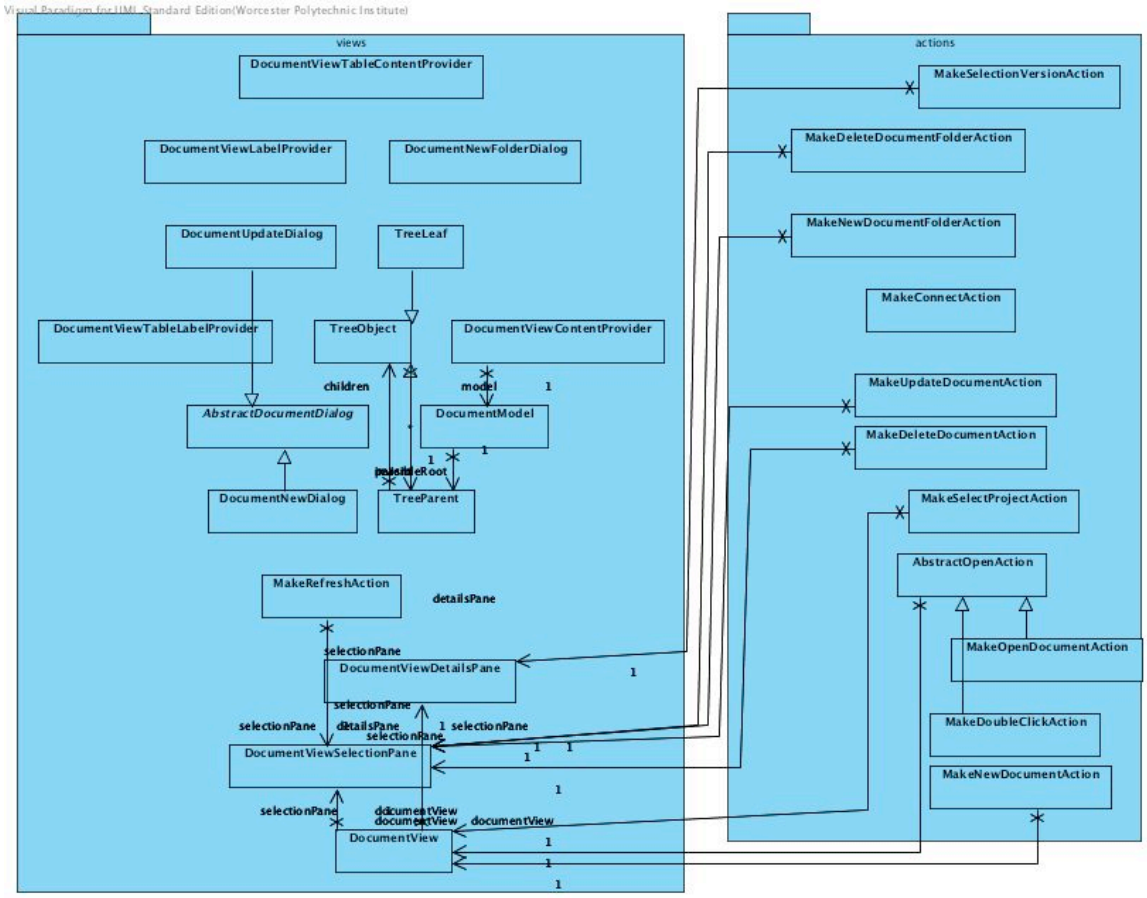


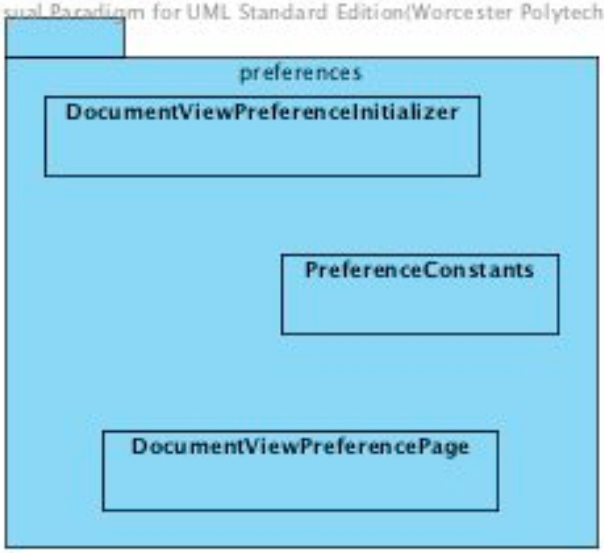
This will open the version of the document you are viewing in the details pane. If no document is in the details pane, the latest version will be downloaded.

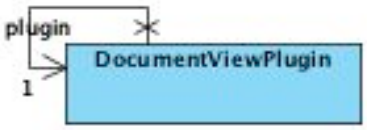
For more information about preferences, please visit Getting Started . If your preferences are set up, those setting will be respected.

Appendix B UML Diagrams

This section includes the UML for our project.







Glossary

API Application Programming Interface

Collaborative Development The sharing and brainstorming of ideas with the purpose of completing a common goal

Eclipse An open source integrated development environment platform built on extendable frameworks

GUI Graphical User Interface

IDE Integrated Development Environment

Java An object oriented language developed by Sun Microsystems

Plug-in Software that alters, enhances or extends the operation of a parent

Regression Un-intentionally returning to a less developed state.

Software Tools and Technologies Group A group at WPI that focuses on new and alternative software tools to meet the needs of developers

SourceForge An Enterprise workflow management tool that provides a central secure location for collaborate development.

UI User Interface

User Story A software system requirement formulated as one or two sentences in the everyday language of the customer

WPI Worcester Polytechnic Institute

References

1. Cheng, Li-Te. “A Need-Based Collaboration Classification Framework”. IBM Research Report. September 16, 2004. Retrieved February 21, 2008 from <http://domino.research.ibm.com/cambridge/research.nsf/2b4f81291401771785>.

2. Booch, Grady. “Collaborative Development Environments”. Rationale Software Corporation. October 28, 2002. Retrieved February 20, 2008 from <http://www.booch.com/architecture/blog/artifacts/CDE.pdf>.

3. Eclipse. “About Us”. The Eclipse Foundation. Retrieved February 21, 2008 from <http://www.eclipse.org/org/>.

4. Ambati, Vamshi. “How can Academic Software Research and Open Source Software Development help each other?”. Carnegie Mellon University. May 25, 2004. Retrieved February 21, 2008 from <http://speech.iiit.net/~speech/publications/OSSE_ICSE04.pdf>.

5. “Open source.” Wikipedia, The Free Encyclopedia. February 20, 2008. Retrieved February 20, 2008 from <http://en.wikipedia.org/wiki/Open_source>.

6. Coar, Ken. “The Open Source Definition (Annotated)”. July 24, 2006. Retrieved February 22, 2008 from <http://www.opensource.org/docs/definition.php>.

7. Cheri, Yann. “Open Source Best Practice”. August 21, 2007. Retrieved February 21, 2008 from <http://www.linuxdevices.com/articles/AT6973816311.html>.

8. Cernosek, Gary. “A brief history of Eclipse”. November 15, 2005. Retrieved February 22, 2008 from <http://www.ibm.com/developerworks/rational/library/nov05/cernosek/index.html>

9. Halloran, T. J. “High Quality and Open Source Software Practices”. Retrieved February 22, 2008 from <http://opensource.ucc.ie/icse2002/HalloranScherlis.pdf>

10. “Collaborative Software Development on Demand”. Retrieved February 22, 2008 from <http://enterprise-development.open.collab.net/files/documents/86/23/collaborative_development_on_demand.pdf>.

11. Taft, Daryl. “Father of Wiki Speaks Out on Community and Collaborative Development”. March 20, 2006. Retrieved February 23, 2008 from <http://www.eweek.com/c/a/Application-Development/Father-of-Wiki-Speaks-Out-on-Community-and-Collaborative-Development/1/>.

12. “SourceForge, Inc.” Wikipedia, The Free Encyclopedia. February 20, 2008. Retrieved February 20, 2008 from <http://en.wikipedia.org/wiki/SourceForge%2C_Inc.>.
13. “Webfoot Task Manager” Greenwood, Brendon and Hlasyszyn, Michael. Retrieved October 21, 2007 from <https://sourceforge.wpi.edu/sf/projects/webfoot>.