2007

ProjectSnap: Addressing the Project Fragmentation Problem

Erin Michael Brimhall
The University of Montana

PROJECTSNAP: ADDRESSING THE PROJECT FRAGMENTATION PROBLEM

By

Erin Michael Brimhall

Bachelor of Arts, Carroll College, Helena, MT, 2003

Professional Paper

presented in partial fulfillment of the requirements
for the degree of

Master of Science
in Computer Science

The University of Montana
Missoula, MT

Summer 2007

Approved by:

Dr. David A. Strobel, Dean
Graduate School

Dr. Yolanda Reimer, Chair
Computer Science

Dr. Joel Henry
Computer Science

Dr. Cameron Lawrence
Business

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

PROJECT OVERVIEW

Introduction

This paper will discuss the activities behind the creation of a software application designed to assist students in higher education with managing and revisiting electronic work processes related to their academic tasks. First, the paper will explore the background of this area of study in terms of other research that has been conducted and the related systems that were developed. This review will help to illustrate the problem this research addresses as well as put the solution in perspective. A majority of this paper will cover the design and implementation of the application itself, as this was the primary focus of the project. Next will be a look at the different aspects of the system, including requirements, design, implementation, and user testing. Finally, this paper will conclude with an evaluation of the software system and future uses and enhancements.

The Challenges of Personal Information Management

Teevan, Jones, and Benderson (2006) define Personal Information Management (PIM) as the use of tools to "…support the activities we, as individuals, perform to order our daily lives through the acquisition, organization, maintenance, retrieval, and sharing of information." While the general concept of PIM extends beyond the realm of computers (e.g. managing physical artifacts), this discussion focuses strictly on research involving the management of electronic information. PIM is a fundamental aspect of

1

computer use, as millions of individuals manage their personal data on a daily basis in order to facilitate both work and leisure activities. By exploring the general characteristics and problems associated with PIM, especially the challenges individuals face when organizing and revisiting electronic information, researchers stand to enhance user experiences through the improvement of software functionality and usability.

Extensive research has been conducted to further our understanding of how individuals manage their electronic data. Specialized software applications are often developed in concert with research efforts in an attempt to address the challenges uncovered during a study. One paper entitled, "Stuff I've Seen: A System for Personal Information Retrieval and Re-Use" focuses on the difficulties users have revisiting electronic information previously encountered (Dumais, Cutrell, Cadiz, Jancke, Sarin, and Robbins, 2003). The authors' solution was an application that tracked users' computer activities, enabling it to later list and detail the information items the user had already seen (e.g. files, emails, web pages, etc). The focus of this particular research effort was on information seeking behavior, specifically, the seeking of information that had been previously found. Many other aspects of PIM exist, including organizing and combining information, as well as seeking *new* information.

Other research efforts to study and address the challenges of PIM include an all-encompassing system entitled "MyLifeBits" (Gemmell, Bell, Lueder, Drucker, and Wong, 2002), which was designed to emulate the concept of a Memex; an information management concept first described by Vannevar Bush (1945). The goal of MyLifeBits

was to first act as a general information store that included everything from notes and paperwork, to audio and video data. It then expanded upon the general notion of the Memex to include extensive querying abilities and to allow "multiple visualizations in the user interface" (Gemmell, et al., 2002). Related systems already exist, such as Haystack (Adar, Karger, and Stein, 1999), which provides functionality similar to MyLifeBits, including the creation of information collections and annotations, and unique visualizations onto the stored data. Systems like MyLifeBits and Haystack (Adar, et al., 1999) attempt to tackle numerous aspects of PIM at once. The project described in this paper adopts a more focused effort that seeks to address a single problem in PIM related to the way users manage their electronic work processes. While the author certainly does not discount the value of tackling broader, more intricate problems, the scope and magnitude of this particular issue paired well with the average complexity of a graduate-level research project, especially in terms of the allotted timeframe and resources. In the next section we will discuss the challenges of managing electronic work processes and projects, and detail several potential solution approaches to assist users in this particular area of PIM.

The Project Fragmentation Problem

As mentioned earlier, there are numerous challenges associated with PIM, including how information is sought, gathered, stored, organized, combined, and recalled. While it is certainly possible (and in many cases beneficial) to tackle several of these concepts in a single research effort, many studies choose to focus on a single problem or concept. One such study conducted by Bergman, Beyth-Marom, and Nachmias (2006),

identifies what they call the "Project Fragmentation Problem" in personal information management. This problem, they say, occurs "when someone who is working on a single project stores and retrieves information items relating to that project from separate format-related collections" (Bergman, et al., 2006). **Figure 1.1** helps to illustrate this concept.
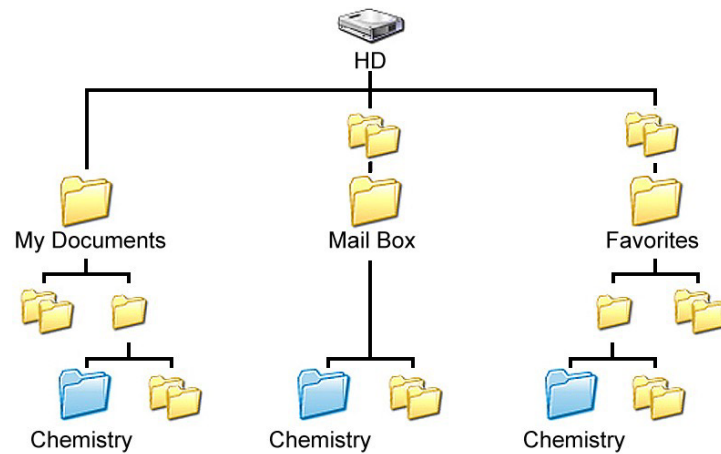
Figure 1.1. Example of the project fragmentation problem.

Here we see information related to "Chemistry" that is separated into different collections (i.e. documents, emails, and web "favorites") based on the data format. Because of this separation, a user faces unnecessary difficulties accessing and using the information when it comes time to perform Chemistry-related tasks. The research of Bergman et al. (2006) showed that interface design has a dominant effect on users' organizational strategies. For example, users will tend to separate their information according to format when the application interface seems to suggest this strategy. Conversely, users often choose to combine their information regardless of format when this behavior is supported by the interface. Another interesting trend described by

4

Bergman et al. (2006) is the tendency for users to separate their information according to projects, as opposed to storing different formats apart. It is possible then that users may lose many of the relationships and contextual cues of their project data because of the project fragmentation problem.

Bergman et al. (2006) describe three potential software approaches to address the project fragmentation problem; integration through search, integration through an additional structure, and the single hierarchy. Integration through search includes such current applications as *Stuff-I've-Seen (SIS)* (Dumais et al. 2003), *PC Data Finder*, and *Google Desktop*. These tools address the project fragmentation problem by allowing users to search for project data across multiple different file formats using a single query, eliminating any reliance on how the information is organized, either in concert or disparately. While this approach is clearly beneficial in aiding the finding and retrieval of project information, it fails to completely capture the context in which the data is used, and what relationships the different pieces of information share. This approach also cannot guarantee that all information resulting from a query is pertinent to the project sought by the user.

The *integration through an additional structure* approach to addressing the project fragmentation problem relies on the use of an outside application that provides the user with the means to specify a personal organizational hierarchy. Project data items are typically incorporated into a hierarchy using "shortcuts" that point to a particular piece of information. While the use of an additional structure allows the user to organize and

5

view project data through a single interface, it also presents potential difficulties. Users would be taxed with managing many structures, such as the ones where the different pieces of information are actually stored, and the hierarchy of "shortcuts" in the additional structure. There are currently a number of experimental and commercial systems that employ this strategy, including *Raton Laveur* (Bellotti and Smith, 2000), *UMEA* (Kaptelinin, 2003), *Drag Strip*, and *OneNote*.

Bergman et al. (2006) present a third solution to the project fragmentation problem that they refer to as the *single hierarchy*, in which all project data is stored in the same folder regardless of format. They describe a possible implementation of this strategy in a system called *ProjectFolders* (shown in **Figure 1.2**), which organizes information on a per-project basis and uses tabs to separate data according to type (e.g., a tab for files, a tab for emails, etc.). Still, this solution has problems of its own. Users would be required to abandon their existing organizational strategies and favor an approach that has them storing all related project data in a single location. Rather than adapting to users' current habits and conceptual models, the *single hierarchy* solution would present an organizational strategy that may be very different from what some users are accustomed to.

Figure 1.2. Implementation of *ProjectFolders*.

The Solution

Computer users employ a wide range of PIM tools and strategies to organize personal, electronic information, and still more ways to correlate and integrate different pieces of data into a single project or work process. Rather than try to force different information organization and storage behaviors on users, a solution has been developed that takes the best parts from the *additional structure* and *single hierarchy* strategies described Bergman et al. (2006), and combines them into an application that allows for the efficient creation and modification of project workspaces across different data formats and collections. This application, which has been dubbed, *"ProjectSnap"*, allows users to take "snapshots" of their computer's currently running applications and any documents or web pages open therein, creating a single reference (e.g. file) that the user

7

can access at a later time to instantly revisit and continue with the work process or project as it was at the time of the "snapshot."

The "reference" mentioned above essentially stores application information, such as the name and path of an application, as well as information about the files opened, including their names, types, and paths. When a reference is revisited by a user, *ProjectSnap* instantly re-opens all documents and applications captured at the time of the "snapshot."

One of the driving requirements decisions behind *ProjectSnap* was to avoid a forcing function that would necessitate users alter their current organizational strategies in order to benefit from the system, such as in the *single hierarchy* approach described by Bergman et al. (2006). *ProjectSnap* functions independent of the formats and locations users choose to store their project data, and instead relies simply on what applications are running and what documents are open at the time a "snapshot" is created. Another high-level requirement for *ProjectSnap* is simplicity in managing projects or work processes. Unlike applications using the *additional structure* approach that impose added organizational overhead by having the user manage yet another electronic hierarchy, *ProjectSnap* focuses on efficient creation of project "references" using a small number of steps and controls. Overall, *ProjectSnap* helps to streamline the capturing and revisiting of electronic work processes in an effort to address just one of the many challenges associated with personal information management. The next chapter will detail the

functionality and usage of *ProjectSnap* by detailing the different interface components

and the underlying conceptual model the application is based on.

CHAPTER 2

THE APPLICATION

Overview

ProjectSnap's functionality has been divided into three primary categories: Work Processes, Snapshots, and Snapshot details. In this section of the paper we will discuss each of these components in terms of the interface appearance, behavior, and available features. The basic appearance of ProjectSnap can be seen in **Figure 2.1**. A late addition to ProjectSnap was a splash screen (**Figure 2.2)** displayed before the main application actually opens. The displaying of the splash screen introduces the application and shows the user that progress is being made on loading any existing work processes and snapshots.
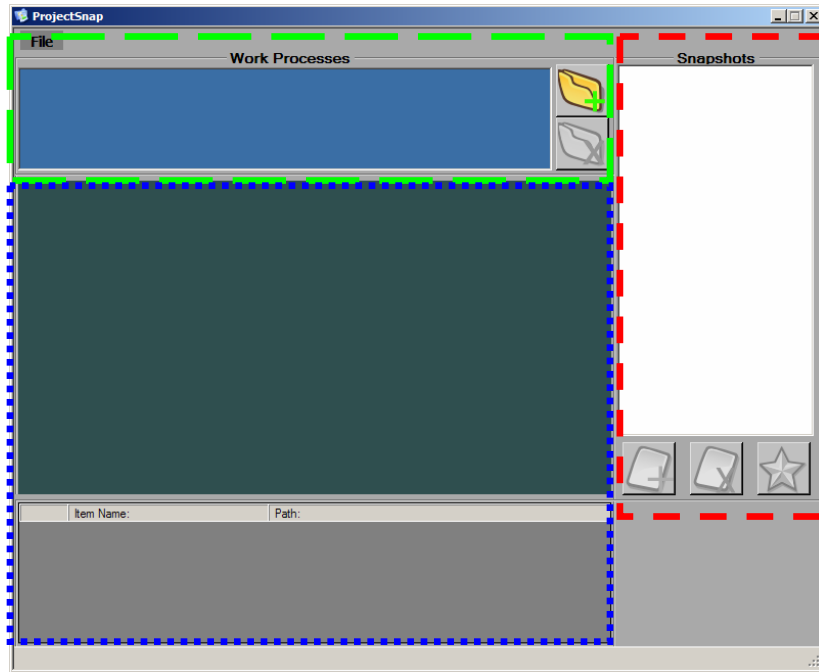
Figure 2.1. The default interface appearance of ProjectSnap. The area at the top is the Work Processes section, the area to the right is the Snapshots section, and the center-most area is the Snapshot details section.
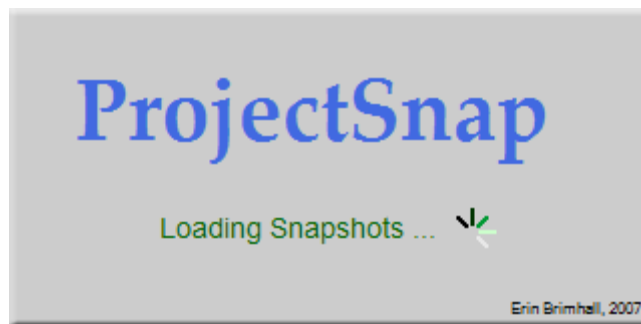
Figure 2.2. The ProjectSnap splash screen.

Work Processes

The first step a user must take in using ProjectSnap is to create a Work Process. A Work Process represents a specific project that the user is working on, e.g. writing a research paper. ProjectSnap was created such that a work process must always be selected in the list, excluding the situation where the user has not created any work processes. This was done to establish some guarantee of what configuration the interface would be in, as well as limit potential confusion for the user. **Figure 2.3** shows an example Work Process, "CS101 Research Paper," in the Work Processes section.
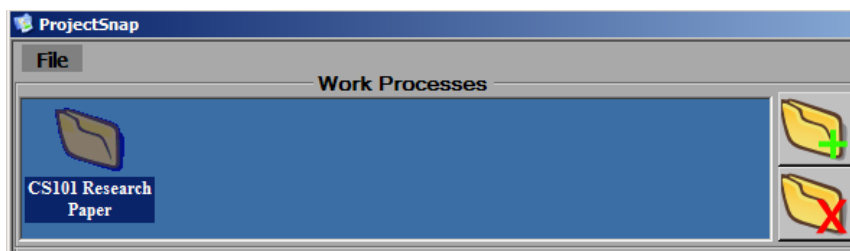
Figure 2.3. An example Work Process.

The user is presented with two functions in the Work Processes area: creating and deleting work processes. When the "New Work Process" button (the folder with "plus" on it) is clicked, the user is prompted to specify a name for the new work process (**Figure 2.4**). Once a name is entered and the user has clicked OK, the new work process will appear in the Work Processes list.
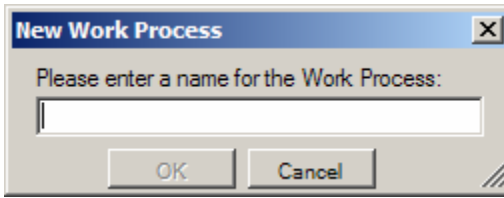
Figure 2.4. The "New Work Process" dialog box.

The user has the option of deleting an existing work process using the "Delete Work Process" button (the folder with the red "x"). When this button is clicked, ProjectSnap will prompt the user if they want to delete the currently selected work process (**Figure 2.5**). Deleting a work process causes all snapshots contained within it to also be deleted.
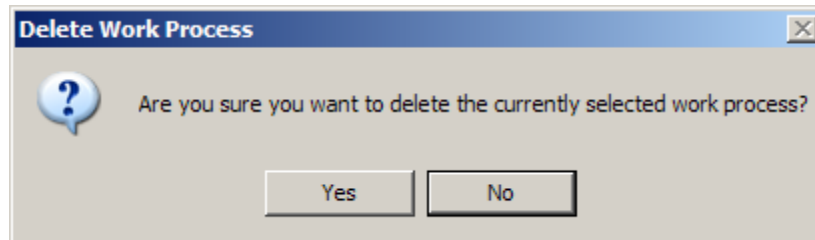
Figure 2.5. The "Delete Work Process" message box.

Snapshots

Once the user has created at least one Work Process, they are then able to start taking snapshots. Snapshots represent a particular moment in time when work was being performed on a work process. They include images of what the computer display looked like at that moment, the date and time, a name for the snapshot (specified by the user), and, most importantly, information about the running applications and open documents.

13

The snapshot image and name, along with the date and time it was created, provide visual cues to assist the user with identifying particular snapshots. ProjectSnap was created such that a snapshot is always selected in the list, excluding the situation in which the user has not created any snapshots. This was done to establish some guarantee of what configuration the interface would be in, as well as limit potential confusion for the user. **Figure 2.6** shows an example snapshot in the "Snapshots" display list.
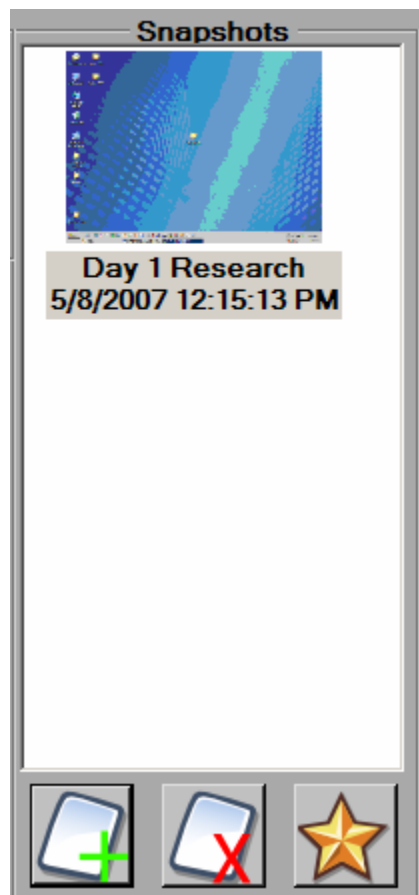
Figure 2.6. An example snapshot.

The user is presented with four functions in the Snapshots area: creating, deleting, launching, and copying snapshots. While each of the first three features has a button

related to it, copying a snapshot is accomplished by clicking and dragging a snapshot to a different work process. When the "New Snapshot" button (the note with the green "+") is clicked, the user sees a small form (see **Figure 2.7**) that prompts them to enter a name for the snapshot. After a name is entered, the "OK" button is clicked and the new snapshot appears in the list.
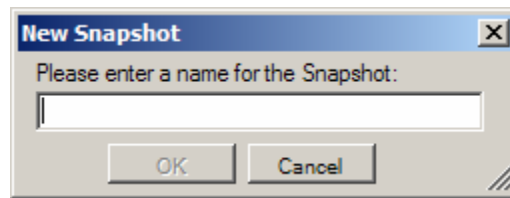
Figure 2.7. The "New Snapshot" dialog box.

The user has the option of deleting an existing snapshot using the "Delete Snapshot" button (the note with the red "x"). When this button is clicked, the user is asked if they want to delete the currently selected snapshot (see **Figure 2.8**). Clicking "OK" effectively removes the snapshot from the list, while clicking "Cancel" returns the user to the primary interface with no changes made.
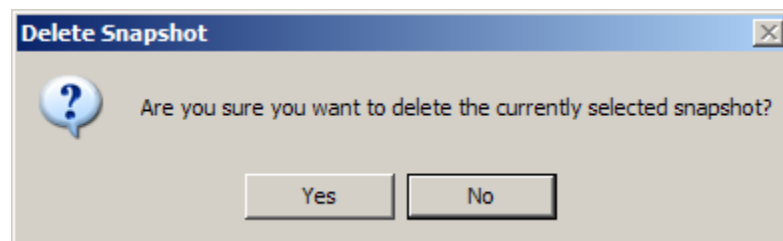
Figure 2.8. The "Delete Snapshot" message box.

In order to revisit or "launch" a specific snapshot that is part of a work process, the user must click the "Launch Snapshot" button (the yellow star). Clicking this button effectively launches all the items that were capture at the time the snapshot was taken. For example, if the user was browsing a folder and had a web page and a Word document open when a snapshot was taken, launching that snapshot would re-open the folder, web page, and Word document.

There are a number of problematic scenarios that can occur when the user attempts to launch a snapshot. The first scenario is when one or more snapshot items (i.e. web pages, documents, etc) is identical to an item that is already open. For example, the user might already have a web browser window with Google open, and then try to launch a snapshot that contains the Google homepage. Rather than ignoring this, ProjectSnap is able to detect these duplicates and ask the user if they intended to open a second copy (see **Figure 2.9**). If they click "Yes," a second copy of the document in question is opened; clicking "No" causes ProjectSnap to move on to trying to launch the next item in the snapshot.
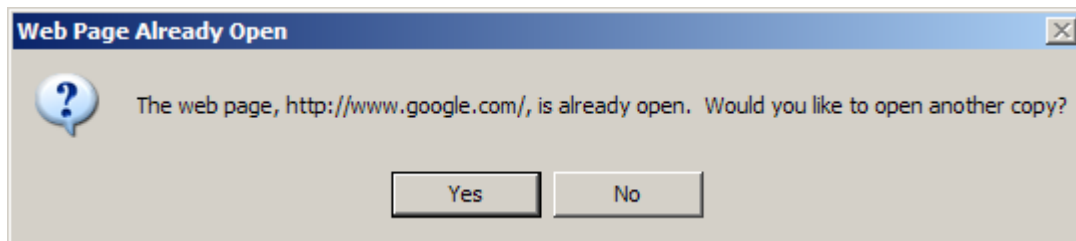
Figure 2.9. The "Web Page Already Open" message box.

The second scenario that can occur when a user goes to launch a snapshot is if a file or folder belonging to the snapshot has been moved or renamed. For example, if the user has a Word document open called, "Assignment1.doc" when the snapshot is taken, and they later rename that same document to something else, say, "Assign1.doc," ProjectSnap will detect this the next time the user tries to launch the snapshot. If ProjectSnap cannot find a file, it has been renamed, relocated, or deleted. Whichever the reason, ProjectSnap will notify the user of this error and ask if they want to keep the item in the snapshot or remove it (see **Figure 2.10**).
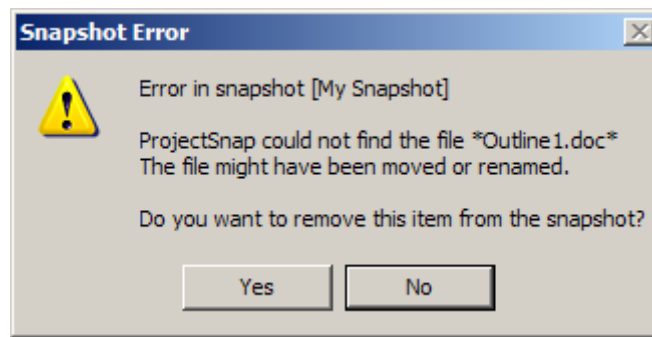
Figure 2.10. A window that notifies the user that a file could not be found and asks if they want to exclude it from the snapshot.

The third scenario that can occur when a user goes to launch a snapshot is if the snapshot contains one or more web pages, but the user's computer is not connected to the Internet, or the web page in question is unreachable for any number of other reasons (e.g. the web page's server is down). Before launching a web page from a snapshot, ProjectSnap first determines if the web page link is reachable. If it is not, the user is

shown a message (see **Figure 2.11**) similar to **Figure 2.10**, notifying them of the error and giving them the option of removing the erring item from the snapshot.

Ultimately, it is the responsibility of the user to manage the windows open on their computer in a way that best facilitates their activities and prevents confusion and clutter. If a user wishes to have an excessive number of windows open or display multiple copies of the same file, ProjectSnap will allow this. It is the author's assumption that users will work on a computer in a way they best see fit and not expect ProjectSnap to manage this aspect for them, as this is simply not its intended purpose.
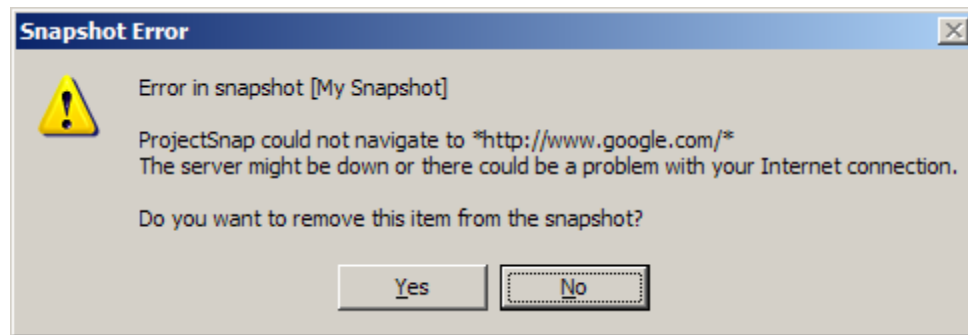
Figure 2.11. A window that notifies the user that a web page could not be found and asks if they want to exclude it from the snapshot.

Snapshot Details

In addition to the "Snapshots" list (**Figure 2.6**), ProjectSnap also presents the user with two other views of the information contained within snapshots. The first view is a larger image of the computer screen, taken the same moment as when the snapshot was created (see **Figure 2.12**). This image is identical to the one used for a snapshot's
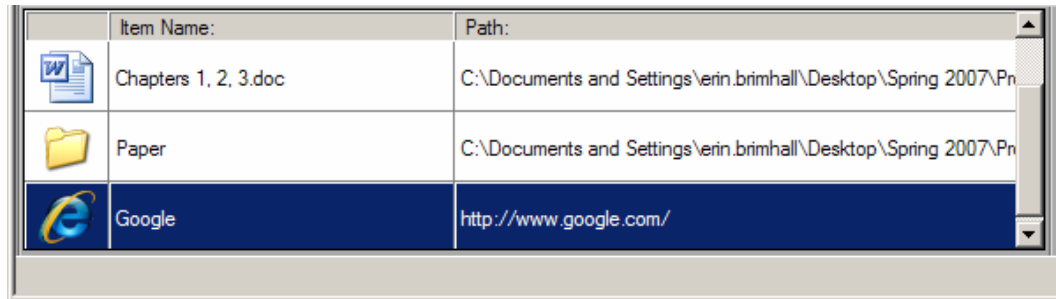
thumbnail in the list display. The enlarged preview provides a more detailed view of the windows and programs that were open when the snapshot was created, giving the user a valuable contextual cue for when they return to a work process.
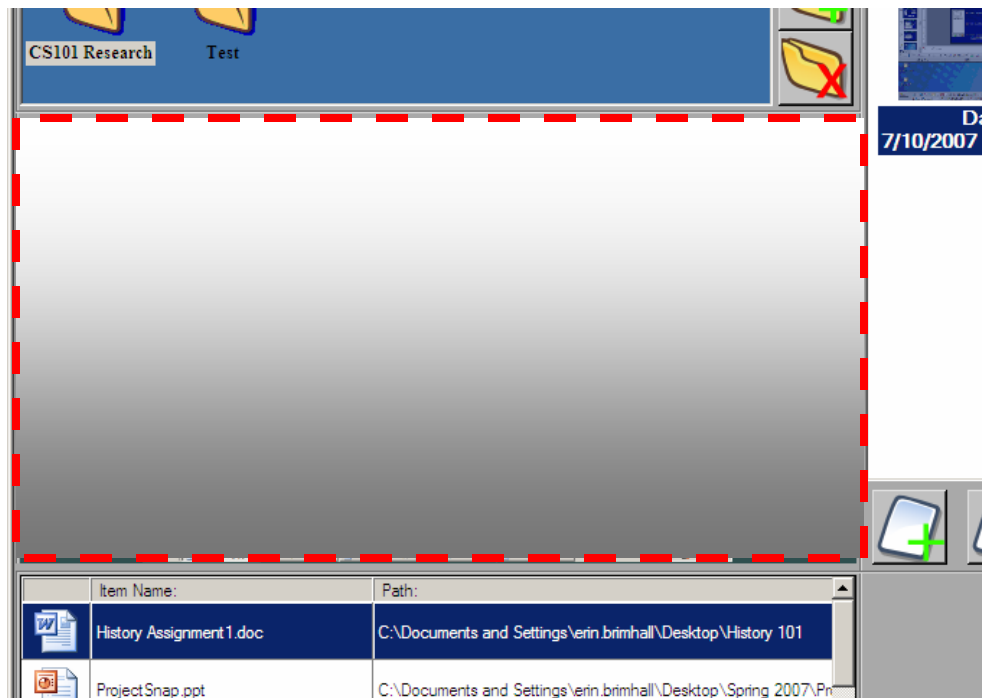
Figure 2.12. The larger snapshot preview image area (denoted by the dashed line).

The second viewing area for snapshot information is the data grid list of all the documents, folders, and web pages that were captured in a particular snapshot (see **Figure 2.13**). From this list, users can quickly and easily identify what items are contained with a snapshot, and even launch the items individually (by double-clicking or through a context menu). The data grid has three columns: the document icon, document name, and document path. The document icon shows the system icon associated with the particular item type, providing the user with a quick and useful visual cue. The document

name column displays the title of the specific web page, document, or folder, and the document path column contains the local or internet path leading to the item.

Figure 2.13. The "Snapshot Details" datagrid.

As a user navigates the ProjectSnap interface, switching between different work processes and snapshots, the various display components update accordingly to reflect the user's actions. For example, when the user switches between work processes, the list of snapshots updates, along with the enlarged snapshot preview image and details data grid.

Other Functionality

In addition to the functionality surrounding work processes and snapshots, ProjectSnap has a handful of other features implemented to help streamline usability and round off the list of system requirements. While ProjectSnap is a tool that assists users with managing their work processes, the application itself is not actually *part* of a work process. For instance, if a user is conducting research on the Internet to facilitate the writing of a paper, ProjectSnap can help them save and recall this work process, but it is not an actual part of the work the user is conducting. With this in mind, the author decided that there needed to be a way to hide ProjectSnap from the context of an active

work process, while still making its features available. The solution was to make project

snap dock as a notify icon (i.e. one of the small icons visible near the system clock on the

taskbar) when the user minimized it. **Figure 2.14** shows the result of minimizing

ProjectSnap.
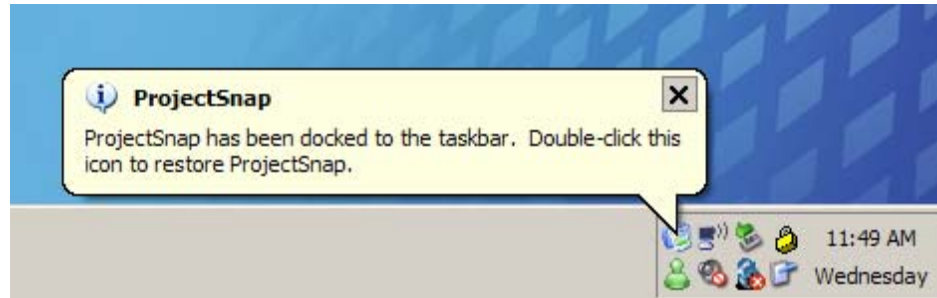
Figure 2.14. The ProjectSnap notify icon.

Docking ProjectSnap as a notify icon causes it to be removed from the taskbar,

effectively excluding it from the user's current work process. Once ProjectSnap is

docked, the user can access a few features by right-clicking the icon for a context menu

(see **Figure 2.15**).
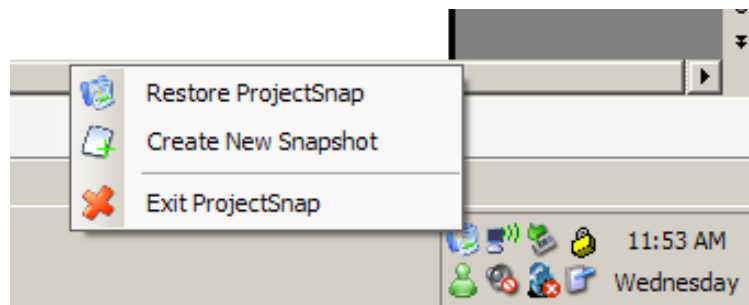
Figure 2.15. The notify icon context menu.

21

These features include restoring ProjectSnap (i.e. bringing it back into view), creating a new snapshot (which is placed in the last active work process), and exiting ProjectSnap. Creating a snapshot via the notify icon provides the user with suitable feedback (see **Figure 2.16**).
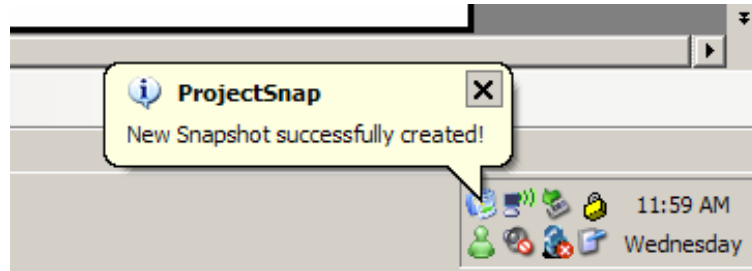
Figure 2.16. The pop-up balloon informing the user of the newly created snapshot.

Like the majority of Windows-based applications, ProjectSnap features a simple menu strip across the top of the interface that provides access to a handful of general functionality items. In this case, only the "File" item is available with just two options beneath it (see **Figure 2.17**).
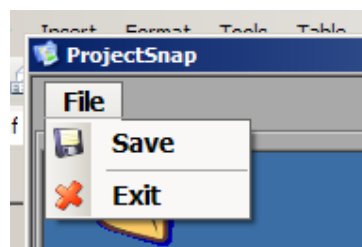
Figure 2.17. The ProjectSnap menu strip.

Users can select the "Save" item to effectively record the current work processes and snapshots that are in ProjectSnap. All work process and snapshot data is saved to an

22

external file that is loaded whenever ProjectSnap is opened. If the user tries to exit the application without explicitly saving their changes, they are prompted to see if they would still like to save (see **Figure 2.18**).
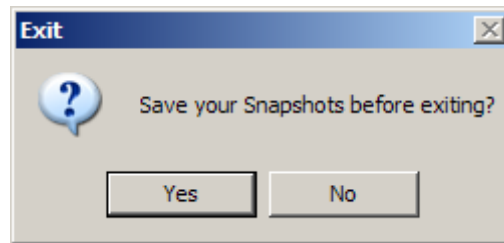
Figure 2.18. The save before exiting message box.

When the user saves their progress, a status bar along the bottom of the interface displays a simple message informing the user of this (see **Figure 2.19**). The status bar is also used to display a message when the user copies a snapshot from one work process to another (see **Figure 2.20**).
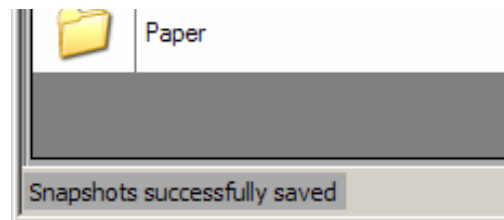
Figure 2.19. The status bar message when the user saves their work.

Paper · C:\Documents and Settings\erin.brimhall

Snapshot "Day 1 Research" successfully copied to Work Process, "New Work Process"
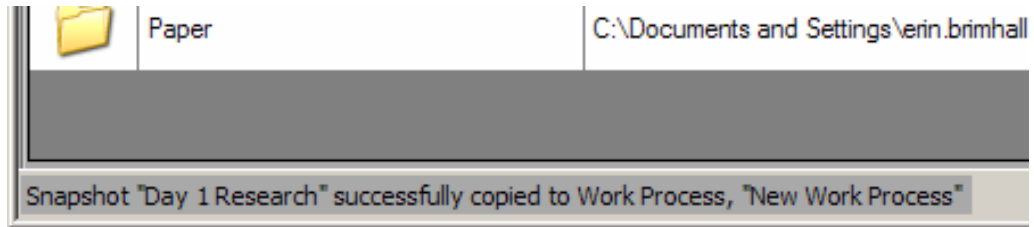
Figure 2.20. The status bar message when the user copies a snapshot.

ProjectSnap implements a variety of application controls (e.g. lists, thumbnails, data grids, etc) that, when combined in a single interface, present the user with numerous options and paths to access the underlying functions and features of the system. Next, Chapter 3 will examine the initial design efforts that proceeded ProjectSnap's implementation, and finish by detailing the underlying architecture that comprises the heart of the application.

24

CHAPTER 3

SOFTWARE DESIGN

Overview

As a Windows-based application, ProjectSnap relies heavily on the use of forms and form controls, including ListViews, Buttons, DataGridViews, MenuStrips, and PictureBoxes, to name a few. Behind the various forms and controls is a carefully conceived system architecture that accommodates the gathering and storing of various data, as well as the enforcement of any relationships between this data. ProjectSnap consists of twenty-one (21) classes arranged into a number of cohesive subsystems that accommodate everything from the creation and storage of snapshots, to the reading and writing of data to and from an external file.

One of the goals behind the architecture design was to facilitate extensions to the system, particularly in regard to what applications and documents can be "captured" by ProjectSnap. As it stands now, ProjectSnap can accommodate Windows Explorer (i.e. folder browsers), Internet Explorer, Microsoft Word, and Microsoft PowerPoint. Because of how the architecture has been broken up, it is a relatively simple task to incorporate the new functionality needed to capture other applications; a task left up to future programmers. In this chapter of the paper, we will discuss the system architecture of ProjectSnap, as well as detail a number of the subsystems and their classes in terms of their attributes and methods.

25

System Architecture

The system architecture for ProjectSnap was first created using IBM's Rational Rose, a unified modeling language (UML) development application. This diagram provided the starting point for development, though quickly became outdated as work progressed. The final architecture diagram was generated automatically by Microsoft Visual Studio, and then tweaked for consistency by the author. Because of its size, the diagram cannot be shown in its entirety. Instead, we will discuss the different subsystems and the relationships between them, starting with the main application form subsystem (**Figure 3.1**).

MainApplicationWindow Subsystem

The MainApplicationWindow class composes a large portion of the ProjectSnap application, specifically, the application window itself, the controls on the window, and all the routines that dictate the actions taken when a user interacts with the controls (i.e. Event Handlers). This class is "collapsed" in **Figure 3.1**, that is, the attributes and methods are not visible. This is done because of the sheer size of the class. At over one-hundred attributes and methods, showing MainApplicationWindow in an expanded view in this paper is simply not possible. Instead, we will focus on the different classes contained in and used by MainApplicationWindow.
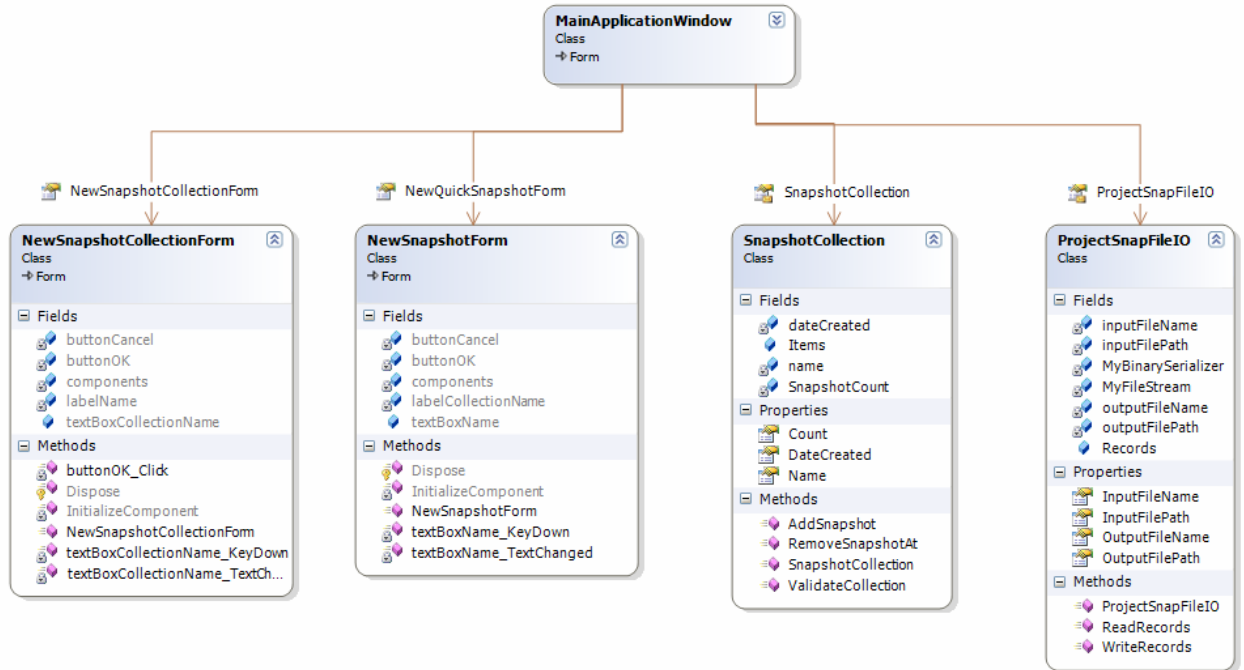
Figure 3.1. The main application form subsystem.

First is the NewSnapshotCollectionForm class. This class implements the form shown when the user clicks the "New Work Process" button (see **Figure 2.4**). Its attributes consist primarily of form controls such as the input text box and the "OK" and "Cancel" buttons. Its methods include event handlers for the actions taken when the user interacts with a control (e.g. what happens when the user clicks the "OK" button).

Next is the NewSnapshotForm class. This class implements the form shown when the user clicks the "New Snapshot" button (see **Figure 2.7**). This class is similar in many ways to the NewSnapshotCollectionForm, as they both offer essentially the same functionality.

27

Next is the SnapshotCollection class. This class implements the data structure responsible for storing snapshots (referred to as a "work process" in the application itself). A SnapshotCollection has a number of attributes, including the date it was created, the snapshots it contains (an ArrayList called, "Items"), the name of the snapshot collection, and the number of snapshots it holds. The methods implemented in this class revolve primarily around operations on snapshots, including adding and removing snapshots to and from the collection. The "ValidateCollection" method is responsible for verifying that the contents of each snapshot in the collection are error-free. Refer to **Figure 3.2** for a more detailed look at the SnapshotCollection class.
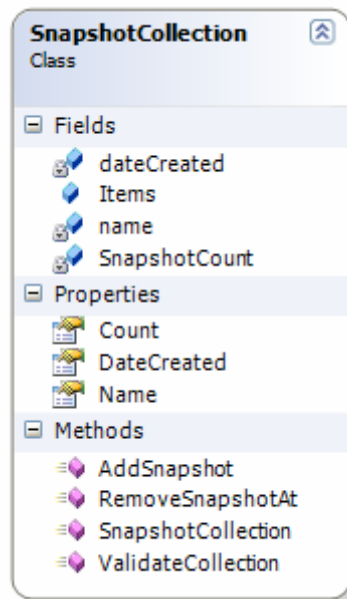
Figure 3.2. The SnapshotCollection class.

Attached to the MainApplicationWindow class is the ProjectSnapFileIO class. This class implements the attributes and methods needed to read and write work process and snapshot data to an external file. In a nutshell, the ProjectSnapFileIO class reads and

writes data using a method called "serialization." Serialization allows for entire objects (both system and user-defined) to be converted into pure binary form. Once the information is converted, it can then be written to an external file using a FileStream, which is a built-in time of the Microsoft .NET Framework. Information that has been "serialized" can also be "de-serialized," effectively decoding objects and their contents from pure binary. Generally speaking, ProjectSnap saves its data by "serializing" the work processes and snapshots and saving them to a file. Similarly, it opens its saved data by "de-serializing" the output file and storing the encoded work processes and snapshots to a local variable (in this case, an ArrayList). See **Figure 3.3** for a more detailed look at the ProjectSnapFileIO class.
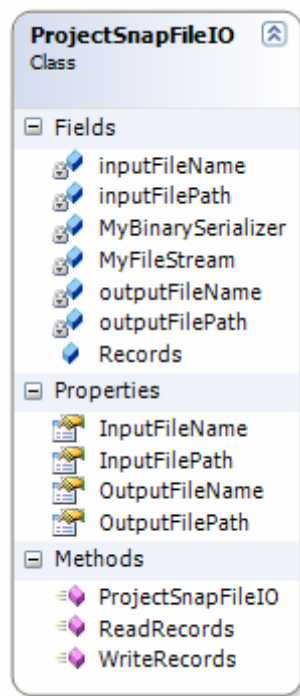
Figure 3.3. The ProjectSnapFileIO class.

Ultimately, the MainApplicationForm subsystems stores some number of SnapshotCollection objects, which in turn store some number of Snapshot objects. MainApplicationForm must *know* about the Snapshot class (in order to do things like populate the snapshot list with the correct information), but it has no idea about (and no access to) the data structures underlying the Snapshot class. This design was done specifically to reduce coupling between classes, and to ensure that changes in the lower-level classes would have little to no effect on the more high-level classes. Next, we will discuss the Snapshot subsystem.

Snapshot Subsystem

The Snapshot subsystem is comprised of six primary classes: the Snapshot, ScreenGrabber, ApplicationObject, ApplicationObjectCreator, ApplicationDocument, and ApplicationDocumentCreator classes. Together, these data structures provide the basis for creating, storing, and manipulating snapshots and their contents. **Figure 3.4** gives a general view of the Snapshot subsystem. Next, we will discuss each of these classes in more detail.
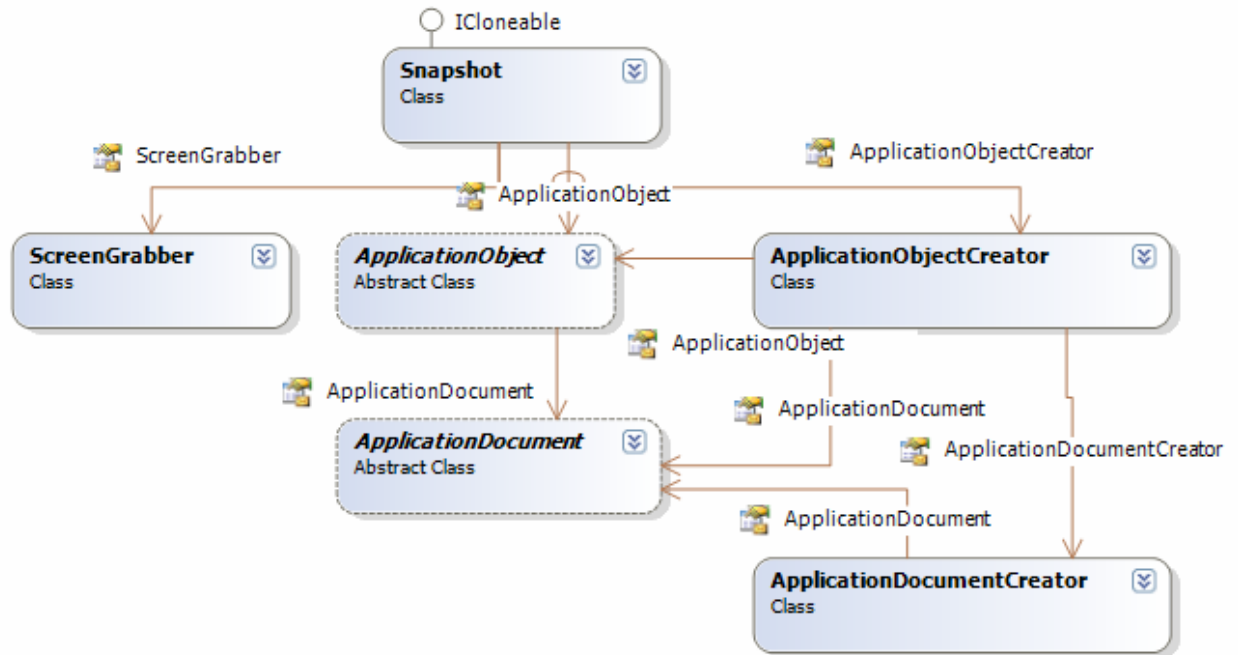
Figure 3.4. The Snapshot subsystem.

The Snapshot class implements the attributes and methods responsible for storing and accessing general information about a snapshot, such as its name, the data and time it was created, as well as the methods that initiate the process of "capturing" the running applications and documents. **Figure 3.5** shows an expanded view of the Snapshot class. Like SnapshotCollection, the Snapshot class also implements a validation method ("ValidateSnapshot") that assists in verifying the applications and documents of a snapshot.

As we saw in Chapter 2, each snapshot includes an image of the computer screen to provide a visual cue to the user. In order to create this image, the Snapshot class uses methods contained within the ScreenGrabber class. ScreenGrabber is actually a small subsystem of its own, as it uses wrapper functions from two external libraries

31

(specifically, the GDI32 and User32 DLLs), though we will not examine this subsystem in detail.   See **Figure 3.6** for a diagram of the ScreenGrabber subsystem.   The ScreenGrabber subsystem independently creates and stores a bitmap image of the desktop that the Snapshot class later uses.

In general terms, SnapshotCollections contain Snapshots, Snapshots contain ApplicationObjects, and ApplicationObjects contain ApplicationDocuments. ApplicationObject is an abstract class (as denoted by its dashed border line in **Figure 3.4**), meaning that it acts as the base class for an inheritance hierarchy, as well as specifies the signature of one or more abstract functions that are actually implemented in each class that is derived from ApplicationObject. The purpose of ApplicationObject and its derived classes (which we will discuss shortly) is to store information about the applications that are "captured" when a snapshot is taken, such as the name and path. We will discuss the ApplicationObject subsystem in more detail later.
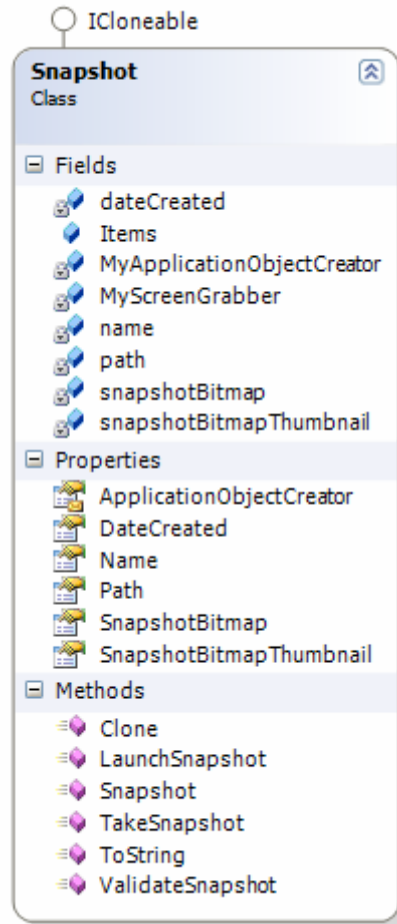
Figure 3.5. The Snapshot class.

As ApplicationObjects are created, so are ApplicationDocuments. ApplicationDocument is another abstract class that provides the interface for creating more specific class definitions. In general terms, the ApplicationDocument class and its derived classes are responsible for storing information about specific documents that were "captured" when a snapshot is created. These classes also implement a number of important methods, including ones that validate the document, check if a document is already open, and ones that actually open the document (i.e. open it for the user to see). We will discuss the ApplicationDocument subsystem in more detail later.

ApplicationObject and ApplicationDocument each have their own object factories (i.e. ApplicationObjectCreator and ApplicationDocumentCreator), which each implement a single function that creates and returns a reference to its respective object type.

When a snapshot is created, the general process flow of the Snapshot subsystem is as follows: the Snapshot class uses its TakeSnapshot method, which starts by calling the CreateApplicationObject method in the ApplicationObjectCreator class. The CreateApplicationObject method cycles through the various types supported by ProjectSnap (i.e. Word, Internet Explorer, PowerPoint, etc), determining which applications are open and then creating the appropriate ApplicationObjects. CreateApplicationObject then calls the CreateApplicationDocument method in the ApplicationDocumentCreator class. Based on its input, CreateApplicationDocument creates the specific ApplicationDocument object, sets its attributes, and returns the reference to the object. CreateApplicationObject then takes the ApplicationDocument reference and stores it in an ArrayList that is part of the ApplicationObject being created at that particular moment. This is repeated until all the documents for the specific ApplicationObject type have been captured and saved. Once CreateApplicationObject has looped through all the open applications and documents, creating some number of ApplicationObjects and ApplicationDocuments, these new objects are stored in an ArrayList that is then stored in the Snapshot object itself. The TakeSnapshot method also utilizes the ScreenGrabber class routines to create a bitmap image of the system desktop, which is then stored in an attribute of the Snapshot class. Next, we will discuss the ApplicationObject and ApplicationDocument subsystems.
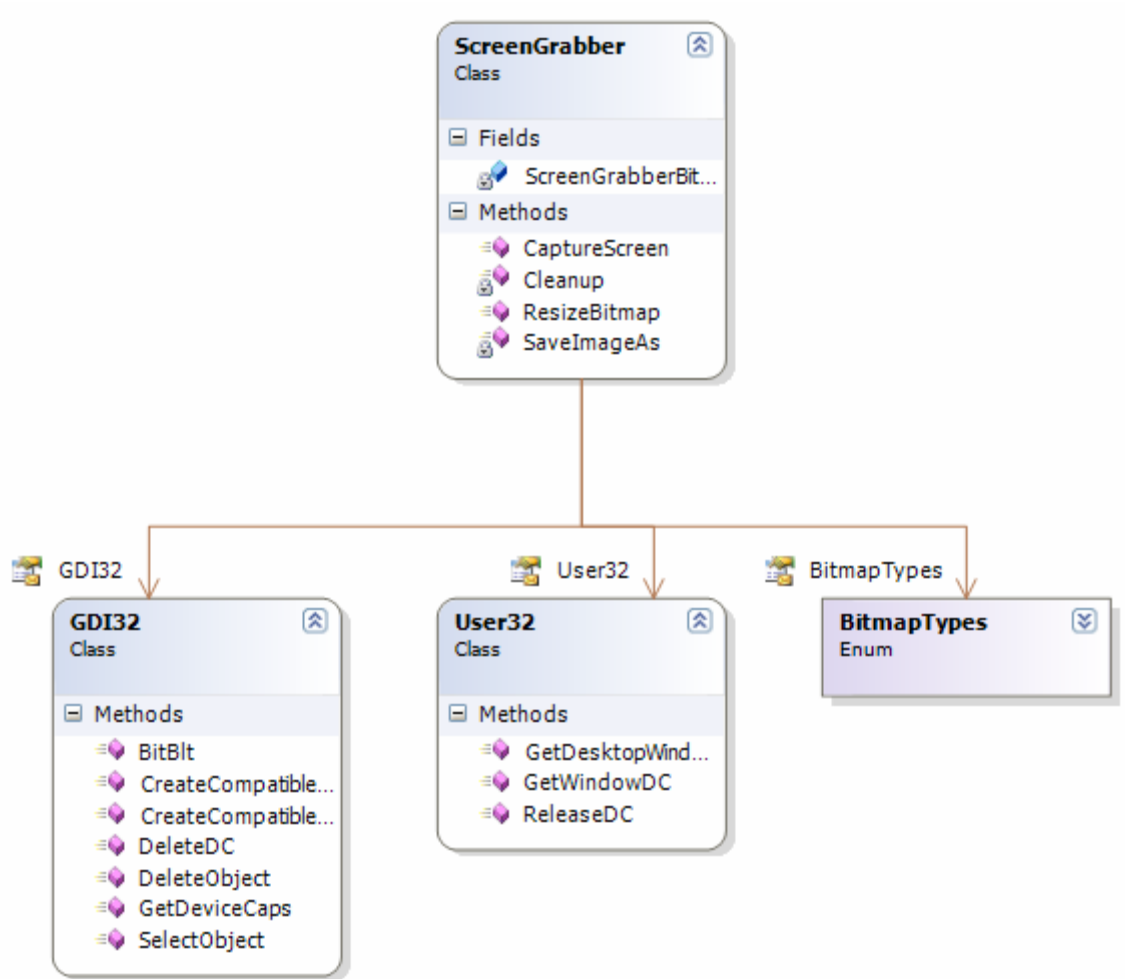
Figure 3.6. The ScreenGrabber subsystem.

ApplicationObject Subsystem

The ApplicationObject subsystem implements the inheritance hierarchy responsible for storing information about specific applications that are captured with a snapshot. It also implements a handful of methods for launching the applications and

their related documents. **Figure 3.7** provides a high-level view of the ApplicationObject subsystem.
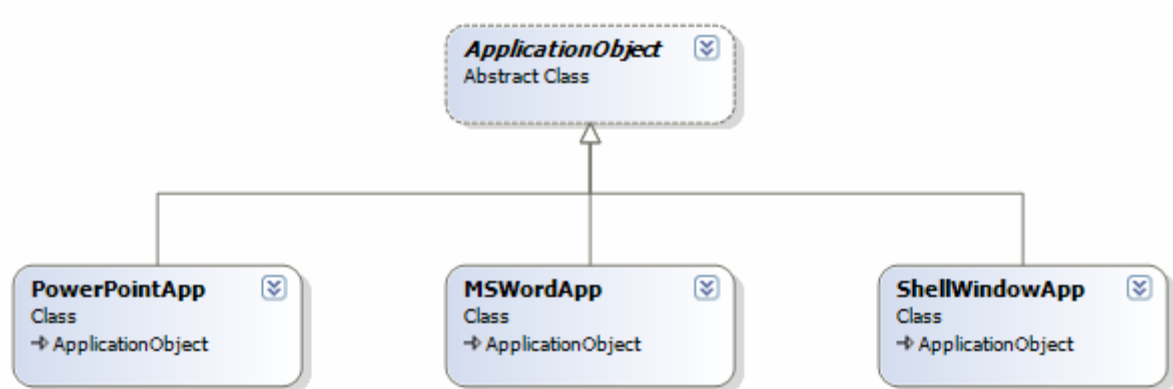
Figure 3.7. The ApplicationObject subsystem.

As mentioned earlier, ProjectSnap supports the capture of several application types: Word, PowerPoint, Windows Explorer, and Internet Explorer. In **Figure 3.7**, we see that ApplicationObject provides the base class for the inheritance hierarchy, with PowerPointApp, MSWordApp, and ShellWindowApp as classes that are derived from it. An interesting fact that was encountered during development was that the operating system did not delineate between Windows Explorer windows and Internet Explorer windows: it treated them both as "Shell Window Objects." Because of this, only one derived class, ShellWindowApp, was needed to handle what first appeared to be two completely different objects.

As the implementation stands now, there is actually very little difference between the four classes shown in **Figure 3.7**. In other words, ProjectSnap would function properly by only using the ApplicationObject class and abandoning the inheritance

36

hierarchy all together. However, the author chose to leave this structure in place with the expectation that future development of ProjectSnap might require it, as well as the fact that this inheritance hierarchy makes the system architecture more understandable.

ApplicationDocument Subsystem

The ApplicationDocument subsystem implements the inheritance hierarchy responsible for storing information about the documents and windows that are captured with a snapshot. It also implements a variety of functions for launching and validating the documents. **Figure 3.8** shows a high-level view of this subsystem.
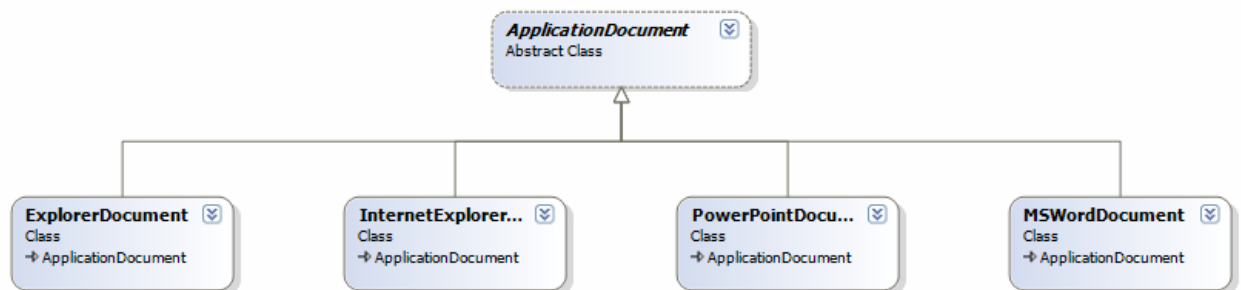
Figure 3.8. The ApplicationDocument subsystem.

Each of the four derived classes from ApplicationDocument have specific implementations of key functions that were defined as abstract in the base class. The first function is OpenDocument. Because the four document types are so different, they each require their own specific way of being opened. For example, a Word document is launched by programmatically creating a new instance of a Word application, creating a new instance of a Word document object using information from the MSWordDocument

class, and then launching the document object with the application object. ExplorerDocument, on the other hand, requires a completely different method for opening itself that revolves around the use of shell windows. The second function is the ValidateDocument function. When a snapshot is loaded or launched, ProjectSnap first verifies that its contents are valid, i.e. that they can be found according to their respective names and paths. The ValidateDocument method in each of the derived classes follows specific steps to determine if a document is reachable. For example, a MSWordDocument is validated by programmatically creating a File object using the path and file name of the particular Word document. If this fails, we know that something about the Word document has changed (e.g. it was relocated or renamed).

Validating if an InternetExplorerDocument (i.e. a web page) is reachable was a totally different matter. Since a web page is not a local file (like a Word document) ProjectSnap uses a WebClient object that tries to browse to the address stored in the InternetExplorerDocument object. The problem with this was that the WebClient object would cause ProjectSnap to "hang" for several minutes as it tried to connect to an unreachable web address. The author's solution was to use a multi-threaded approach. The actual call to test the web address using the WebClient is made in a thread separate from ProjectSnap's. It also times out after 5 seconds. The end result is a way of testing a connection to a web page that doesn't interrupt the operating of ProjectSnap. If the web address cannot be reached after 5 seconds, it is assumed it cannot be reached at all.

Validation & Launching Snapshots

As mentioned in the previous sections, the SnapshotCollection, Snapshot, ApplicationObject, and ApplicationDocument classes all have a validation function of some kind. The validation process starts from the top down, beginning at the SnapshotCollection class. The ValidateCollection function loops through all the snapshots in the collection and calls each one's ValidateSnapshot function. The ValidateSnapshot method then loops through all the ApplicationObjects stored in the snapshot and calls each one's ValidateObject method. The ValidateObject method then loops through all the ApplicationDocuments stored in the particular ApplicationObject and calls each one's ValidateDocument method. It is at this level where the actual validation occurs. Recall that at the ApplicationDocument level, the ValidateDocument method is abstract and is actually implemented in each of the four derived classes. Through the power of polymorphism, ProjetSnap is able to determine which of the four ValidateDocument functions is called based on the underlying class type of ApplicationDocument. As errors are discovered, they are propagated back up to the Snapshot level, where the user is given the opportunity to deal with them (see **Figure 2.10** and **Figure 2.11**).

Launching a snapshot is achieved in a way similar to how it is validated. Starting at the Snapshot level, the LaunchSnapshot method is called, which loops through each ApplicationObject stored in the snapshot and call each one's LaunchAllDocuments method. The LaunchAllDocuments method loops through all the ApplicationDocuments stored in the ApplicationObject and calls each one's LaunchDocument method. It is at

39

this level where the actual code exists that opens the document on the user's computer. Again, using polymorphism, ProjectSnap can automatically determine which version of the LaunchDocument method to call based on the underlying document type (e.g. MSWordDocument, InternetExplorerDocument, etc).

Behind the scenes, ProjectSnap's features and appearance are implemented through numerous classes and other programming constructs, each with its own (often extensive) set of attributes, methods, and relationships. Chapter 3 has detailed this underlying design. In Chapter 4, the process and challenges involved in developing ProjectSnap will be discussed, including how technical risks related to system development were addressed, and the general order in which system components were designed and implemented.

CHAPTER 4

IMPLEMENTATION

Technical Risks

Before actual development began, the author faced a number of key technical challenges with ProjectSnap. These high-risk factors dictated whether or not key concepts behind ProjectSnap could even be implemented. The primary technical risk that was addressed was how to programmatically attach to running instances of applications and gather information about them. In order for a snapshot to "capture" application and document data, ProjectSnap needed to determine what applications were open, create some kind of hook for each one, and then read in information about them (e.g. the name of the application, where its executable is located, the name of the documents open in it, the path to those documents, etc). It was the author's initial hope that the Windows operating system maintained some general list of the running applications and documents, and that creating a snapshot would involve accessing and reading data from this list. The first few weeks were devoted to exploring this concept.

As work progressed, it became apparent that such a list of applications and documents did exist in the operating, but its uses were limited. The Running Objects Table (ROT) maintains a listing of the application objects actively executing on a computer, but what "objects" are registered in the table was unclear and inconsistent. For example, one instance of Microsoft Word could be running that wasn't registered with

the ROT, while another running instance was. Whether or not an object became registered depended on many things, and could not be counted on with any sort of confidence. Even after using the ROT to get a listing about an object, it wasn't possible to determine the local file path of the object's application file, let alone the information about the object's open documents. A different approach was needed.

The author soon found that each type of application (e.g. Word, PowerPoint, Internet Explorer, etc) required a specific approach for attaching to it. The MS Office Interop libraries provided key methods for attaching to running instances of Word and PowerPoint, while Windows Explorer and Internet Explorer required the use of special system libraries centered around shell window objects. The first bit of code written for ProjectSnap tested how the Word, PowerPoint, Windows Explorer, and Internet Explorer applications could be attached to. Once the author was satisfied that this approach was possible, development began on the ProjectSnap application itself.

ProjectSnap Development Process

Project development was completed almost exclusively on the author's research laptop, along with a lab desktop computer that was used to host the Visual Source Safe configuration management software and the actual ProjectSnap program files. Once the source controls were in place and the high-risk technical factors had been addressed, ProjectSnap was developed and tested over the course of approximately 11 weeks, bringing the total time to research, develop, and test the application to around 4 and a half months (18 weeks).

42

Development began based on the initial system architecture designed in Rational Rose. With the design diagram, the author identified the major subsystems (e.g. ApplicationObject subsystem, ApplicationDocument subsystem, etc.) and began coding from the top down. For example, with the ApplicationObject subsystem, the base class (ApplicationObject) was created first in order to specify the basis for the derived classes, which were implemented immediately afterwards. Each module was tested as it was finished, and then tested again as the modules were combined to form subsystems. A basic Windows form interface was needed in order to test aspects of certain subsystems and view the results. This basic testing form slowly evolved into the ProjectSnap application interface itself. As new subsystems were created, new controls (such as ListViews, Buttons, and DataGridViews) were added to the form to test the additional data structures. Each control on the application form went through a gradual process of changing the appearance and position of the control, as well as adding more functionality and behavior that was tested as development progressed.

Eventually, the design and layout of the main application window of ProjectSnap was finalized. By that time, the base functionality of work processes and snapshots had been implemented, though many details still needed attention. It wasn't actually until later in development that the ApplicationDocument subsystem design was finalized and implemented. This was due to the author's erroneous decision that the ApplicationObject subsystem would handle the majority of complex operations (e.g. opening and verifying data, capturing data, etc), when in reality these operations belonged in the

43

ApplicationDocument class. The ApplicationDocument class was unable to handle these operations alone, so it was converted to the base class for the inheritance hierarchy that now forms the ApplicationDocument subsystem.

As testing was conducted on the various functional components of ProjectSnap, new requirements began cropping up that had not been identified in the requirements development stage of the project. See **Appendix C** for the initial (and now outdated) list of system requirements. Two examples of unplanned system requirements are error-handling and data validation, both of which became key components of ProjectSnap's functionality and usability. Previously unanticipated usage scenarios were identified in the interface that caused many different unrecoverable errors, so programmatic safeguards were added to ensure that the effects of these errors was lessened, or that their occurrence was prevented entirely (i.e. via the different validation functions). Many of the late additions to ProjectSnap were purely superficial or only slightly improved the user's experience. Things like tooltips on controls, a status bar to display user feedback, and improved error messages added small but noticeable quality to the application.

Ultimately, the implementation of ProjectSnap followed a rather hybrid approach of both the spiral and iterative development processes. Work began by first identifying and addressing the high-risk areas of the project, then developing and testing on a per-subsystem basis, and then iterating through different builds of the application as new problems and solutions were discovered. This process provided the much-needed flexibility to adapt to the varying technical challenges that arose during development.

The overall impact of changing directions during development was alleviated by a solid system architecture that allowed for new system components to be added and modified with relative ease. The end result is a concise, well-tested, well-documented application that provides interesting and valuable functionality, as well opportunities for future extensions and improvements.

ProjectSnap Installation & Removal

ProjectSnap can be installed on a computer by launching the "setup.exe" file included with the installation package. This file launches the Project Snap setup wizard, which allows the user to specify a number of options, including where to install ProjectSnap and which users will have access to the application (see **Figure 4.1**). The default installation path of ProjectSnap is "C:\Program Files\University of Montana\ProjectSnap\"
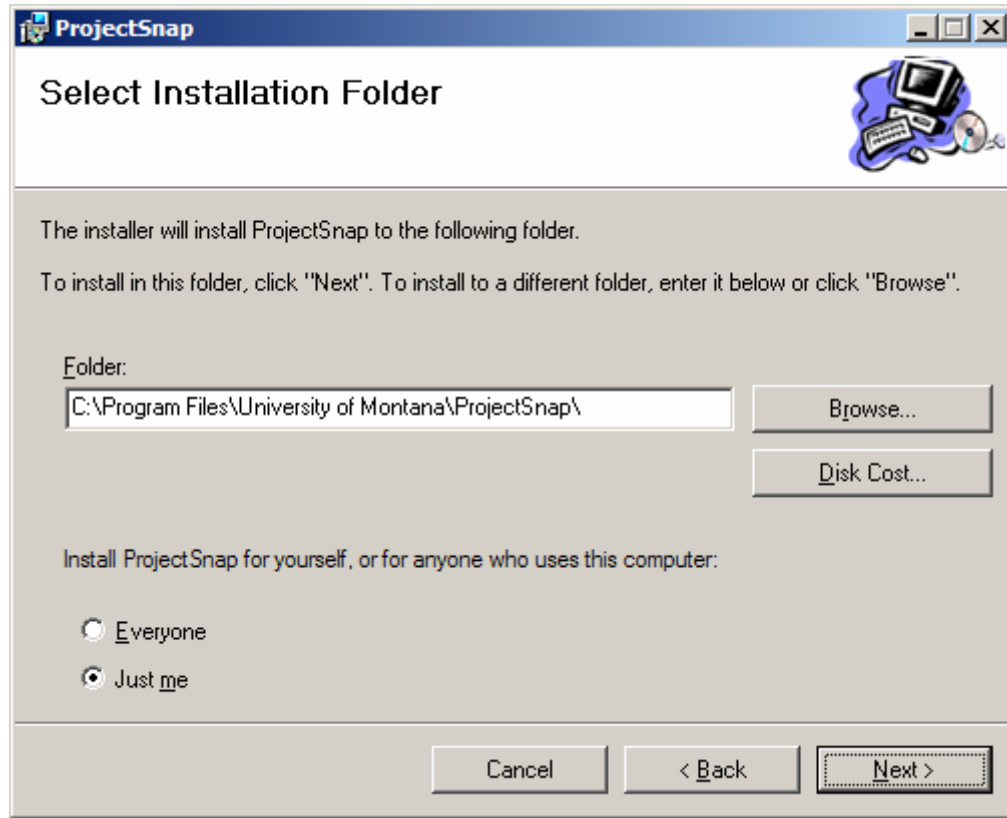
Figure 4.1. The ProjectSnap setup wizard.

The setup wizard creates a shortcut icon on the desktop that allows ProjectSnap to be quickly launched. Once installed, ProjectSnap can be removed from the system through the "Add/Remove Programs" menu under "Control Panel."

ProjectSnap's development posed many challenges and risks that were overcome or mitigated. Some issues were anticipated and addressed accordingly, while others sprang up unexpectedly and were remedied or diverted as quickly as possible. The next chapter will discuss the user testing conducted on ProjectSnap, including the goals, method, and materials used for the tests. It will conclude with an overview of the results and their implications for the system.

CHAPTER 5

USER TESTING

Introduction

Testing is a key activity in producing high quality software applications. From the beginning of development until the end, a system must be tested in order to ensure that defects are efficiently detected and fixed. Basic testing began early with ProjectSnap, starting at the unit and subsystem level, and continuing up to rudimentary system and acceptance testing. As a heavily user-centered application, the author felt it necessary that ProjectSnap undergo testing with real test subjects in order to evaluate the system's acceptability in terms of its features and usability, as well as to identify any errors or missing functionality. In this chapter of the paper we will discuss the user testing that was conducted on ProjectSnap, including the goals and method. We will conclude by looking at the testing results and their implications for the system.

Goals

Like most software applications, the purpose and use of ProjectSnap is well-understood by its creator, but not necessarily by the casual computer user. This gulf of understanding can stem from many things, including the inherent complexity of the application, a subtle or specialized problem area, or poor interface design and usability. One of the goals for ProjectSnap's testing was to evaluate the users' understanding of what exactly the application does and how it could be useful to them, if at all. Because

47

the typical computer user does not currently have tools similar to ProjectSnap available to them, it is important to determine whether the application is easy to learn and use. Users would quickly lose any sense of *why* they would want to use ProjectSnap if the application is too difficult to understand.

In addition to identifying any conceptual difficulties user might have with ProjectSnap, testing was also conducted to uncover any defects, usability issues, and missing functionality. In the next section we will discuss the materials and test environment, as well as how testing was conducted.

Method

In order to test the key functional components of ProjectSnap, the author began by developing a set of materials to help guide the user through a specific set of activities to perform on the system during the testing session. These activities were broken up according to a series of scenarios and tasks, with the scenarios describing a hypothetical situation in which the users should conduct one or more tasks. See **Appendix A** for a copy of the user testing scenarios and tasks handout. Test results were recorded by the author in the form of worksheet notes, separated according to the respective scenario and tasks. This worksheet also included three post-test interview questions. See **Appendix B** for a copy of the user testing worksheet. Since the testing involved human subjects (i.e. university students) the author sought and received written approval from the Institutional Review Board (IRB) at the University of Montana. This approval ensured that the testing was safe for the subjects to participate in and that the person administering the tests (i.e.

the author) had taken and passed the pertinent exams on ethical testing procedures and considerations.

Testing was conducted on the author's laptop computer in room SS402 of the Computer Science Department. The laptop included the latest version of ProjectSnap, an active Internet connection, and a desktop folder containing a number of files needed to complete the scenarios and tasks. The author informally recruited 4 test subjects, including 3 graduate students from the Computer Science Department and 1 graduate chemistry student.

Testing sessions lasted an average of 45-minutes and included a brief introduction of the system by the author, followed by the user working through the list of scenarios and tasks. Users were told that even though the author would be present during testing, he would not be able to answer their questions (at that time) or assist them. Test subjects were encouraged to think out loud as they worked and ask questions that could be answered at the end of the test session. Once the user finished the scenarios and tasks, the author interviewed them briefly on their experience (see the end of **Appendix B** for the short list of interview questions).

Results

User response to ProjectSnap was notably positive. Test subjects appreciated ProjectSnap for a variety of reasons, especially its functionality which they felt was new, interesting, and useful. The layout of the interface was generally well-liked, with

49

subjects saying that the color scheme and button icons were appropriate and intuitive. The different form components seemed clearly related to most subjects, though one subject reported that the relationship between a snapshot and the datagrid contents was not immediately obvious to him. He suggested that this confusion could be alleviated by somehow reorganizing the form controls or even adding new labels.

There were a handful of functional obstacles that each test subject ran into. The first was the absence of double-click functionality on snapshots. When a test subject went to launch a snapshot, they always first tried to double-click the snapshot item in the list. At testing time, the only way to launch a snapshot was by using the "Launch Snapshot" button; double-clicking the list item had no effect. Another unimplemented feature that users consistently attempted to find was a context menu on the snapshots. When test subjects were asked to copy a snapshot they always first tried to right-click the snapshot, looking for a context menu with a "Copy" item on it. Of course, no such context menu exists on the snapshots, causing users to stumble briefly before eventually creating the copy by dragging and dropping a snapshot with the mouse.

Test subjects consistently looked to the "File" menu strip item for functionality, such as for adding new work processes and snapshots, and copying snapshots. The only two items under "File" are "Save" and "Exit." Users did not return to the "File" menu after seeing its items, but it is clear that certain redundancies should exist in order to provide users with more than one path to key functionality. Tables **5.1, 5.2,** and **5.3** detail the results from the ProjectSnap user testing. **Table 5.1** details the ideas for new

50

functionality, **Table 5.2** details ideas for modifying existing functionality or behavior, and **Table 5.3** details general issues or concerns with the application. Test results are numbered, categorized (either "New Functionality," "Modified Functionality," or "General Issue"), and given a short title and description.

#	Category:	Title:	Description:
1	New Functionality	Double-click Snapshots	Double-clicking a snapshot in the list should produce the same result as clicking the "Launch Snapshot" button, i.e. launching the snapshot.
2	New Functionality	Snapshot Context Menu	Right-clicking a snapshot in the list should bring up a context menu with options such as, "New Snapshot," "Cut," "Copy," "Paste," "Delete," "Rename," and "Launch."
3	New Functionality	Extended Datagrid Context Menu	A "Remove" option should be added to the datagrid context menu so users can easily omit items from a particular snapshot.
4a	New Functionality	Extend Menustrip	The "File" item on the menu strip should be expanded to include a "New" option that expands to include "New Work Process" and "New Snapshot."
4b	New Functionality	Extend Menustrip	An "Edit" item should be added to the menustrip that would include such options as, "Cut," "Copy," "Paste," "Delete," and "Rename." These items would only operate on the currently selected work process or snapshot.
4c	New Functionality	Extend Menustrip	A "Snapshot" item should be added to the menustrip that will include at least one option: "Launch Current Snapshot."
5	New Functionality	Support More Applications	While no user actually brought this up during testing, the author believes that adding support for more applications is key to making ProjectSnap more useful. Adobe PDF files should be the first item on the list of applications to add support for.

Table 5.1. Ideas for new ProjectSnap functionality.

#	Category:	Title:	Description:
1	Modified Functionality	Minimizing and Hiding ProjectSnap	Currently, when ProjectSnap is minimized, it is removed from the taskbar and docked as a "notify icon" near the system clock. This confused many users when it first happened. Instead, ProjectSnap should remain on the taskbar when it is minimized. If the "X" button on the application window is clicked, then ProjectSnap should dock as a notify icon. If the user wishes to actually exit the program entirely, they can do so through the "File" menu or by using the context menu on the notify icon. This minimizing/closing behavior would closely match that of Microsoft Messenger.
2	Modified Functionality	Left-clicking the Notify Icon	Currently, the notify icon does not respond to a single clicked from the left mouse button. The functionality of the icon's context menu should be modified so it displays with both a single right and left mouse click.

Table 5.2. Ideas for modifying existing ProjectSnap functionality.

#	Category:	Title:	Description:
1	General Issue	Connection Between Current Snapshot & Datagrid	A few users mentioned that it was not immediately clear what the relationship is between the currently selected snapshot and the datagrid contents, i.e. that the datagrid was displaying the "contents" of a snapshot.

Table 5.3. General user issues with ProjectSnap.

CHAPTER 6

PROJECT EVALUATION

Assessment

The creation of ProjectSnap was challenging on many levels, with obstacles and questions arising at practically every stage of development. The first issues arose as soon as the author posed the key concept behind ProjectSnap: how can you programmatically connect to and capture information from running applications? While there was no question this was possible, whether or not it was doable according to the author's technical background and project timeframe was another matter. The first 4 weeks of work was spent researching this question and developing test code to verify the information found. While the author enjoyed a considerable amount of success initially, especially when it came to programmatically connecting to Microsoft products, this progress quickly waned. Numerous difficulties arose with connecting to Adobe Acrobat Reader; the application which supports the PDF file type. Because of the project's time constraints, support for PDF files has been left for future enhancements.

Despite a variety of technical challenges the author feels he has accomplished a majority of his goals through the development of ProjectSnap, particularly to create an application that helps computer users manage their electronic work processes. One of the initial concerns with the project was not just *can this be done*, but also, *can this be useful*? The entire concept of ProjectSnap originated from the author and was not based

54

on existing studies with proven results. The risk of failure in either the implementation or in the value of the end product was very real. While the author held a good deal of confidence with the usability and features of ProjectSnap before user testing, the feedback and comments from the different test subjects helped solidify the usefulness and value of the system. ProjectSnap demonstrates a way for users to quickly capture, organize, and revisit basic electronic work processes centered around Microsoft products, i.e. the Office suite, Internet Explorer, and Windows Explorer. The application itself operates very efficiently in terms of the time it takes to capture and load snapshot data, and offers a high degree of robustness and stability due to the author's careful consideration of usage scenarios.

Still, even in its current form ProjectSnap has a number of areas that should be expanded or modified to improve usability and the general benefits offered by the application. A majority of these improvements fall under the category of functional redundancies, in other words, multiple interface paths to key system features. The addition of new menustrip items, context menus, and mouse-click behaviors (as described in **Tables 5.1** and **5.2**) would help to round out the set of accessibility options available to users. Overall, this project presented a challenging and innovative concept to address programmatically. The end result is a fully functional application that offers valuable features and interesting hints at future uses and extensions.

Future Work

As it stands now, ProjectSnap provides efficient means for computer users to manage and revisit their electronic work processes, all through a stable and user-friendly graphical interface. Still, there is much work left to be done to improve and expand the application. Besides the new interface and usability additions listed in **Tables 5.1** and **5.2**, the primary area that the author feels deserves the most attention is the support of new application types in snapshots. PDF files are number one on the list of new documents to support due to their overwhelming usage, especially by students in higher education.

The architecture of ProjectSnap was designed with expansions in mind, especially allowing new application and document types to be added with relative ease. For example, to add support for a new application/document type, a number of changes would need to take place. First off, a new class derived from ApplicationObject would need to be created, e.g. AcrobatApp. The virtual functions defined in the ApplicationObject class would need to be implemented specifically for the new derived class. The CreateApplicationObject method of the ApplicationObjectCreator class would need to be modified to handle the new AcrobatApp sub-type. Next, a new class derived from ApplicationDocument would need to be created, e.g. AcrobatDocument. Its ValidateDocument and OpenDocument methods would need to be implemented specifically for the PDF file type. ApplicationDocumentCreator would need to be modified slightly to account for the new document type (e.g. PDF files) so it would know how to parse the input data and create an AcrobatDocument. Finally, at the Snapshot

56

level, the TakeSnapshot method would need to be modified very slightly to include Acrobat files (PDFs) in snapshots.

While the description above might make it sound like a lot of work to add support for PDF files, we have to appreciate the fact that a majority of the code needed to accomplish this can be borrowed and reused from existing classes and methods. The other major factor is that none of the user interface, screen capturing, or file I/O components of ProjectSnap need to be modified to account for the new document type. Again, one of the major goals with ProjectSnap was to reduce coupling between classes and allow the addition of new functionality without needing to completely overhaul existing components.

ProjectSnap was designed to be very modular in terms of how the system components can be separated, combined, and used independently of one another. As a result, it is possible to take and use the entire application or remove any number of its subsystems for separate use. For example, if another researcher was interested in capturing application data, they might copy the ApplicationObject and ApplicationDocument subsystems into their own software. These subsystems would provide the capturing functionality without any "snapshots" or "work processes" being involved. Similarly, if a researcher wanted the screen grabbing or file I/O functionality, they could easily isolate and copy these subsystems. While it would be unreasonable to try to isolate the user interface component of ProjectSnap for separate use, the vast

57

number of methods contained within it could certainly provide an extensive code reference for future researchers.

One exciting prospect for ProjectSnap was its possible inclusion in an eNotebook system being developed by the author's research advisor, Dr. Yolanda Reimer. The purpose of the eNotebook is to assist students with the way they store, organize, access, combine, and use electronic information related to their academic tasks. A work process is just another piece of information that students might benefit from having more control over, so ProjectSnap's functionality seemed like an obvious complement to the features of the eNotebook. The eNotebook could incorporate ProjectSnap in a number of ways. One would be to simply launch ProjectSnap from the eNotebook. This approach would be the quickest and easiest, though the eNotebook might benefit from having ProjectSnap functionality embedded directly inside it. This approach would involve copying most all the different subsystems of ProjectSnap, except the user interface. The eNotebook developers would then have to decide on a new interface for accessing the functionality of the ProjectSnap subsystems and displaying the captured data. Again, the ProjectSnap user interface could serve as a valuable reference and source of code to copy from.

58

BIBLIOGRAPHY

Adar, E., Karger, D., and Stein, L.A. (1999). Haystack: per-user information environments. *In Proceedings of CIKM. ACM Press*, 413-422.

Bellotti, V. and Smith, I. (2000). Informing the design of an information management system with iterative fieldwork. *Proceedings of DIS*, 227-237.

Bergman, O., Beyth-Marom, R., and Nachmias, R. (2006). The project fragmentation problem in pers onal information management. *Proc. CHI 2006 – Personal Information Management.*

Bush, V. (1945). As we may think. *The Atlantic Monthly, 176(1)*, 101-108.

Dumais, S., Cutrell, E., Cadiz, J., Jancke, G., Sarin, R. and Robbins, D.C. (2003). Stuff I've seen: a system for personal information retrieval and re-use*. Proc. SIGIR,* 72-79.

Gemmell, J., Bell, G., Lueder, R., Drucker, S., and Wong., C. (2002). MyLifeBits: fulfilling the Memex vision. *Proceedings of the tenth ACM international conference on multimedia, ACM Press: Juan-les-Pins, France.* 235-238.

Kaptelinin, V. (2003). UMEA: translating interaction histories into project contexts. *Proceedings of CHI,* 343-360.

Teevan, J., Jones, W., and Benderson, B. B. (2006). Personal information management. *Communications of the ACM, January. Volume 49, No. 1.*

APPENDICIES

Appendix A – User Testing Scenarios and Tasks Sheet

ProjectSnap is a simple application designed to help students in higher education manage their electronic work processes by providing ways of saving and recalling the open applications and documents on a computer. Two key terms associated with ProjectSnap are "Work Process" and "Snapshot." A Work Process can be thought of as a specific task, such as writing a paper for a class. A Snapshot represents the state of a Work Process at a specific moment in time. By completing the following scenarios and tasks you will help the author identify any problems with ProjectSnap, as well as potential new features. Please read the scenarios and tasks carefully, as I will not be able to offer you any assistance as you work through them. If you become completely stuck or an error occurs with ProjectSnap then I will help. Feel free to ask any questions as you work and I will be sure to answer them at the end of this session.

Thank you for agreeing to help with this testing and good luck!

Scenario #1:

You are a college student at the University of Montana who is currently enrolled in HIS101, Intro to World History. You have an assignment due next week and want to begin researching and writing it. You recently installed the ProjectSnap application to assist with tracking this and other academic work processes.

Task #1:

Find and open the folder, "History 101" on the desktop of the computer. Open the "History Assignment1.doc" file that is inside the folder. Type "History 101 Paper" in the MS Word document and save it. Open the "History Lecture1.ppt" file that is inside the "History 101" folder. Open Internet Explorer and browse to "www.google.com" Type, "History" in the Google search field and hit the "Enter" key. Using the shortcut on the desktop, launch "ProjectSnap."

Task #2:

Inside ProjectSnap, create a new Work Process and name it, "HIS101 Assignment1." Now, create a new Snapshot and name it, "Research Day 1." Close all the currently open windows on the computer, *including* ProjectSnap. If ProjectSnap asks if you want to save your Snapshots, click "Yes."

Task #3:

Reopen ProjectSnap. Revisit the work you were doing on the history paper by launching the Snapshot you took earlier. Now, close all the open windows on the computer *except* ProjectSnap. Using ProjectSnap, open only the "History Assignment1.doc" file. Create a new Snapshot called "Research Paper" and save the changes. Close ProjectSnap and all other open windows on the computer.

Scenario #2:

You are feeling ambitious about your school work and decide to begin work early on the second assignment for your "History 101" class. You realize that you can reuse some of the resources from the first assignment and return to ProjectSnap for help.

Task #1:

Open ProjectSnap. Create a new Work Process called, "HIS101 Assignment2." Copy the "Research Day 1" Snapshot into the "HIS101 Assignment2" Work Process. Launch the copied version of the "Research Day 1" Snapshot.

Task #2:

Open a *new* web browser window and navigate to "www.wikipedia.org" Open the "History Lecture 2.ppt" file in the "History 101" folder on the desktop. Minimize ProjectSnap. Without restoring ProjectSnap, use the appropriate icon near the system clock to create a new Snapshot called "Test Snapshot". Now, restore ProjectSnap, save your changes, and then close the application and all other open windows.

Task #3:

Open ProjectSnap. Inside the "HIS101 Assignment2" Work Process, delete the "Research Day 1" and "Test Snapshot" Snapshots. Delete the "HIS101 Assignment2" Workprocess. Save the changes and close ProjectSnap and all other open windows.

Task #4:

Experiment freely with ProjectSnap. Create and delete Work Processes and Snapshots. Do activities that will help clarify any aspects of ProjectSnap. Even do things that you think might break the application. Feel free to ask any questions as you work.

Appendix B - User Testing Work Sheet

Subject #:_____

Scenario #1:

Task #1:

Find and open the folder, "History 101" on the desktop of the computer. Open the "History Assignment1.doc" file that is inside the folder. Type "History 101 Paper" in the MS Word document and save it. Open the "History Lecture1.ppt" file that is inside the "History 101" folder. Open Internet Explorer and browse to "www.google.com" Type, "History" in the Google search field and hit the "Enter" key. Using the shortcut on the desktop, launch "ProjectSnap."

Task #2:

Inside ProjectSnap, create a new Work Process and name it, "HIS101 Assignment1." Now, create a new Snapshot and name it, "Research Day 1." Close all the currently open windows on the computer, *including* ProjectSnap. If ProjectSnap asks if you want to save your Snapshots, click "Yes."

Task #3:

Reopen ProjectSnap. Revisit the work you were doing on the history paper by launching the Snapshot you took earlier. Now, close all the open windows on the computer *except* ProjectSnap. Using ProjectSnap, open only the "History Assignment1.doc" file. Create a new Snapshot called "Research Paper" and save the changes. Close ProjectSnap and all other open windows on the computer.

Scenario #2:

Task #1:

Open ProjectSnap. Create a new Work Process called, "HIS101 Assignment2." Copy the "Research Day 1" Snapshot into the "HIS101 Assignment2" Work Process. Launch the copied version of the "Research Day 1" Snapshot.

Task #2:

Open a new web browser window and navigate to "www.wikipedia.org" Open the "History Lecture 2.ppt" file in the "History 101" folder on the desktop. Minimize ProjectSnap. Without restoring ProjectSnap, create a new Snapshot called "Test Snapshot". Now, restore ProjectSnap, save your changes, and then close the application and all other open windows.

63

Task #3:

Open ProjectSnap. Inside the "HIS101 Assignment2" Work Process, delete the "Research Day 1" and "Test Snapshot" Snapshots. Delete the "HIS101 Assignment2" Workprocess. Save the changes and close ProjectSnap and all other open windows.

Task #4:

Experiment freely with ProjectSnap. Create and delete Work Processes and Snapshots. Do activities that will help clarify any aspects of ProjectSnap. Even do things that you think might break the application. Feel free to ask any questions as you work.

Post-Test Questions

1. What are some of your general thoughts about ProjectSnap? What did you like? What didn't you like?

2. What are features or functionality you would like to see added to ProjectSnap?

3. Is ProjectSnap an application you could see yourself using? Why or why not?

Appendix C - Initial System Requirements

System features are prioritized according to their perceived value and complexity. Priorities appear at the ends of sentences in parentheses, e.g. (MEDIUM). High priority items are scheduled for implementation, while only some Medium priority features will be implemented.

1. **Users will be able to create snapshots of their immediate application workspace.**

    1.1. Snapshots will be taken using either the "snapshot" button in the user interface, or under the "Snapshots" menu. (HIGH)

    1.2. Users will chose between a regular snapshot and a "quick" snapshot.

        1.2.1. A regular snapshot will allow them to individually select which items they want included in the snapshot. (HIGH)

        1.2.2. A "quick" snapshot bypasses the selection process and automatically creates a snapshot that includes all open applications/documents. (MEDIUM)

    1.3. The user will go through a three-step process when creating a regular snapshot:

        1.3.1. Step one is to specify the applications/documents they want to include in the snapshot (see Requirement 2). (HIGH)

        1.3.2. Step two is to specify where the snapshot will be saved. (HIGH)

        1.3.3. Step three is to give the snapshot a name. (HIGH)

2. **Users will be able to specify which applications and documents they want to include in their snapshot.**

    2.1. Once a snapshot is taken, the user will see a list of all applications, documents, and web pages that are open and can be included in the snapshot. (HIGH)

    2.2. Each item in the list will have an associated checkbox that indicates whether or not the application/document will be included in the final snapshot. (HIGH)

        2.2.1. Checking a checkbox means the item will be included in the snapshot; unchecking a checkbox means the item will be excluded from the snapshot.

    2.3. Each item in the list will have details accompanying it, including what application it belongs to, the name of the file or web page, and the date created (if applicable). (HIGH)

66

       2.3.1. This "metadata" will be displayed in a label object that shows the extra details for the item currently selected in the list.

3. Users will be able to name their snapshots.

3.1. In the snapshot creation wizard, the user will be able to type a name for their snapshot. (HIGH)

3.2. If the user is creating a quick snapshot, the name will default to "Snapshot xx-xx-xx" where the x's are the current system date. (MEDIUM)

4. Users will be able indicate where they want to save their snapshots.

4.1. In the snapshot creation wizard, a folder-selector dialog will enable the user to select the directory to save their snapshot to. (HIGH)

4.2. If the user is creating a quick snapshot, it will be automatically saved to a "ProjectSnap" folder in the user's "MyDocuments" folder. (MEDIUM)

5. Users will be able to choose where they save their snapshots.

5.1. After taking a snapshot, a dialog box will appear to allow the user to select the destination folder where the snapshot reference will be saved. (HIGH)

6. Users will be able to open snapshots that they have taken.

6.1. Snapshots will exist as .pss files that contain XML entries representing the application and document information captured at the time the snapshot was created. (HIGH)

6.2. Users can open snapshots in two different ways:

       6.2.1. Double-clicking a snapshot file will open *ProjectSnap* and display the list of applications and documents that were captured previously. (MEDIUM)

       6.2.2. Open a snapshot file from within the *ProjectSnap* application by using the "Open Snapshot" button or menu option. (HIGH)

6.3. Only one snapshot can be open in *ProjectSnap* at a time.

7. Users will be able to "launch" snapshots.

7.1. Launching a snapshot will open all applications, documents, and web pages associated with the particular snapshot. These applications will be opened in the

67

operating system and not in a separate "shell" or in the *ProjectSnap* interface. (HIGH)

7.2. To launch a snapshot, the user will need to open a snapshot in *ProjectSnap*, and then select the "Launch Snapshot" button or menu item. (HIGH)

7.3. If applications and documents are open from previous snapshots, any new snapshot applications and documents will be opened along with them.

8. Users will be able to modify snapshots.

8.1. The user will need to have opened a snapshot using either of the methods described above. (MEDIUM)

8.2. The user will then select either the "Modify Snapshot" button or menu item. (MEDIUM)

8.2.1. *ProjectSnap* will compare the currently open applications and documents to those captured in the snapshot, and then list all the applications and documents from the original snapshot, as well as those currently open that do not match the snapshot.

8.2.2. Each item listed will have an indication as to whether it is an old application/document that is currently opened, an old application/document that is currently unopened, or an application/document that is not part of the original snapshot.

8.2.3. Using the checkbox feature described earlier, the user will be able to select and unselect which applications/documents they want to update the snapshot with.

8.2.4. Clicking the "Update" button will save the snapshot with the new application/document data as specified by the user.