Take-home Final Exam

Mark Randles
CS464 Software Development
2006-12-12

Question 1

One of the major aspects of the CS464 Software Development class was a group project, in which a single group would develop independently a piece of software for a selected client. Assignments to groups were random selection done by the instructor Dr. Chao in secret outside of the class room.

As assigned, my group was to continue the development of a on-line survey tool. During the initial stage of the project we held an initial planning meeting involving the entire group and our client, Dr. Dunning the CS Department chair. At this meeting we took delivery of the original code and project and gathered from Dr. Dunning the user stories he wished us to accomplish or at least try to.

Our original, stated goals, were to do at least three things: 1) to port the system from it's current home on the `voyager.cs.bgsu.edu` system to the newer `voyager-new.cs.bgsu.edu` system; 2) make improvements to the existing user-interface to not only improve it's readability but it's usefulness; 3) to expand many of it's core system to be more functional then currently implemented.

Since we were working with an already existing system, it was necessary to do a code review to see if the modification desired could be accomplished with the existing code base or if it would require heavy modification, would it just be easier to re-write the tool, incorporating the changes as we reimplemented the system.

The code assessment was done on an individual basis, and after a weekend of review, we came together as a group and decided that the best approach was to re-write the system instead of trying to modify the original source code. The original source was written in a very procedural structure and did not take advantage of many of the new facilities of the PHP scripting language such as object-orientation.

For our new implementation we used an object-orientated (OO) method to develop code that would be both easy to extend and maintain. The problem was broken down as such. The survey would be composed of questions, each question having a single type. Each question type would have an associated class which would take care of the rendering of the question (input or output controls), how the question would be stored in the database, and how the question would be created. The results set would be broken down in a similar manner, with each question having a subset of all the answers to the question. The actual web page container, which would facilitate code reuse within the project, was also implemented as a class which could be polymorphed into the different page styles and types. To encapsulate the database from our application a database class was also used. Additionally, a class to handle authentication of users was used.

Websites, especially PHP based sites, are especially well suited for OO back-end handlers, such as question and survey classes, and procedural front end scripts to operate on these back-end classes. Our project was no different, with all of the pages that users would open in their web browsers written in a procedural form, using the facilities created with the OO back-end to assemble and render whatever function that page would perform.

The entire project, however, was not only re-implementing the old system with a different design method. Key new features were added to the system to increase it's usefulness. First and foremost, the user pages have become more

usable due to things such as controls not scrolling off the bottom of the pages and consistent page organization and themes. The question types have also been expanded by allowing the user a variety of new control styles such as multi-line text controls and improvements of existing types such as the rating question.

During the life of the project, improvements were also made to the user pages. Things such as the dynamic addition of answers to multiple-choice questions were wanted and needed by the client. New user pages were also added to the system to improve it's usefulness to a wide range of users. Pages to retrieve/reset passwords and manage administrator users were added.

Now that the project is complete, it becomes clear just how far it has come. At it's beginning it was a sub-par system, with many visual and systemic bugs. Now, though not bug free, it is a much easier system to maintain and modify. Our critics can say that we did not add any major new features, however, we've done more then that. We've laid the foundation for future persons to extend our work without the need to re-implement the project.

Question 3

Object-oriented development (OOD) is a methodology for developing systems by breaking them apart into independent objects, and then assembling the system back out of the objects. To make a real-world analogy, it's much like building a spaceship out of Lego bricks. Each section of the ship can be built out of smaller parts such as individual bricks, some specialized and some not. The whole ship is built out of these smaller sub-sections. One can think of each of the bricks as a attribute to an object and the smaller sub-sections as individual objects. They all fit together to build the complete system, where none could do it alone.

Object-oriented development is not without it's pros and cons. Before a propper assessment of OOD can be done, it's important to note some of the weak points. First and foremost, OOD is easy to understand but can be very, very hard to do correctly in practice. OOD requires years worth of experience to do well, and by far and large, novices will do a poor job of it.

Another weak spot OOD has is there are certain systems which just can not be broken down into smaller parts. Generally these systems are extremely small, so there is a minimum of “guts” or system to work with. A related to this problem is the concern of some systems about performance. Many real-time systems require that any code run is as fast as possible. OOD tends to generate slower code depending on language and implementation style. These sort of systems may benefit from object-orientated design, but that design may need to be translated into some other implementation paradigm.

The benefits of good OOD are many. Most of, if not all, good OOD will result in a clean system, with a clear and concise logic about it. The actual design, in most cases, will lead easily to the actual system implementation with classes and interactions carrying over from the design to the implementation seamlessly. Good OOD will also lead to an ease of maintenance that stems from it's logical structure.

During the semester, we were required to do a software development project. The actual problem is too lengthy to be defined here, and is done in **Question 1** of this document. During our development, we did use limited OOD. During the initial development stage of our project, we laid out the various processes and objects that we would need to use to complete our system. We did not, however, generate any of the written documentation for this system, and rather did it on an ad hoc basis during implementation.

We chose to use OOD because of it's strengths. One of our project goals was to make the system as maintainable as possible, since not only were we given unmaintainable code originally, but we knew others in the future would be working on it. None of OOD's weaknesses applied to our project, so they were of no concern.

OOD is a powerful tool. It can be a great boon to any project where it is competently used. OOD can also be a bane to any project where it is misused or ill used. If you remove the negative aspects of OOD, it is a great benefit and a excellent development method.

Question 4

The rise of the Internet and the World Wide Web (WWW or web) has lead to a great burst of web-based tools for all things. Many of these tools are only useful for certain niche markets. However, tools such as blogs and wikis have become globally used and both are useful in software development.

A wiki is a collaborative website on which all of the content is both written and edited by the users of the website. Everybody who can access a wiki should be able to modify it's contents to fix spelling errors, cover gaps in it's knowledge base, and improve the overall quality of the wiki. The idea of a wiki has been around since the early 1990's and only after 2000 has it began to explode into many, many uses.

A blog, on the other hand, is a single publisher electronic journal that is open to the public. The word “blog” is derived from the word “weblog.” Many people, especially the young, maintain blogs on many diverse sites from their own personal web spaces to sites specifically set up for blogging. There is very little collaborative element to a blog, though most allow for other users to make comments on a specific entry.

Each of these has it's own application to the world of collaborative software development. Primarily the focus of this course was on Extreme Programming (XP) and it's techniques, therefor for the rest of this essay, I will focus on the use of blogs and wikis in XP.

Within the XP methodology there are a variety of different types of journals and documents that need to be kept up to date throughout the entire development cycle by the entire development team. Documents such as the daily journal and project plan are the responsibility of the entire team to keep up to date and then publish with full disclosure to the customer. Wikis and blogs are perfectly suited for this application, each more suited for a particular application then the other.

One of the most important documents in an XP team is the daily journal. Generally there is one specific person who's job is to keep it up-to-date with the doing and goings on of the day, but the entire team has the responsibility and capability to respond to entries and further comment. A wiki could do this, but since a wiki is a free-form document system, it lacks the chronological structure that a blog has. Blogs are a perfect tool to maintain a XP group's journal. Blogs being built upon the same premise of a journal, are integrally tied in with the chronological structure that the XP journal needs. Add to that facilities to comment on a given entry, blogs are a perfect tool to maintain the XP journal.

The most important document in an XP project is the project plan. Not only does it contain the sequence of stories that will eventually lead to the final product, it also contains a concise and focused description of the project at any given point in time. A wiki is very well suited to this task. Not only could a wiki document the user stories that go into a XP project, but it could also tie them into the project plan and additional technical documents like APIs. Since the wiki also allows anybody to modify a page, it's the perfect tool for collaborative documents such as the project plan.

Wikis and blogs are innovative new tools, created out of the collaborative environment bread on the World Wide Web. Each tool can fill a need for

collaborative teams, many times doing something that previously was difficult or time consuming to do. Both wikis and blogs, used in the right situation, will further advance software development and ease some of the burden on the developer, so they can focus on writing better software.