CMPSC 280
Principles of Software Development
Fall 2015

Final Project Assignment: Implementing a Complete Programming Systems Product

Introduction

Throughout the semester, you have learned about the phases of the software development lifecycle and both the software tools and the non-cognitive skills that are needed to support the creation of high-quality software in each of these phases. Yet, with the exception of the previous laboratory assignment, none of the past work required you to deliver a programming systems product to an external customer. Moreover, even in this past assignment you did not proceed through the entire software development lifecycle to specify, architect, design, implement, test, document, and release a complete programming systems product that fully meets the needs of your external customer.

In this final project assignment, you will gain experience in eliciting requirements from an external customer. Then, using these requirements, you will specify, architect, and design a system that will meet the needs of this individual. While always keeping in direct contact with your client through discussions in the course's Slack team, in-person meetings, and short demonstrations, your team will then implement and test your software. Next, your team will produce comprehensive documentation that will ensure that your customer can easily use your program after this course has completed. Ultimately, your team must create a GitHub project for your system and, with the approval of the customer, release the system under a free and open-source license.

In addition, your team will use tools, such as JDepend, JavaNCSS, JaCoCo, and MAJOR, to characterize and evaluate the software system that you plan to implement and release. As you complete this assignment, you and your team members should always remember that, while your customer is an expert in the domain in which you must implement software, this individual is not a computer scientist or a software engineer who has studied the material in Computer Science 280. Therefore, you should present to your customer an intuitive explanation of what the results from these analysis tools say about the quality of the final system that you are delivering. Finally, as in the previous assignment, you and your team members will also explore both the issues related to effective technology transfer, the ways to ensure the quality of a software system's documentation, and the best practices in the licensing and release of free and open-source software.

Eliciting Requirements from the Customer

For this final project assignment, your team is tasked with specifying, designing, implementing, documenting, and releasing a programming systems product that will support the efficient and cost-effective scheduling of a shuttle van service provided by the Civic Engagement program at Allegheny. For the same of simplicity, this assignment sheet will refer to this product as `van-schedule` because it will be used by a member of the Civic Engagement staff in Allegheny College's Gateway to schedule van trips for students who volunteer off-campus. Each team is encouraged to develop their own name for their programming systems product that best represents their tool's key features.

In particular, your external customer for this project is Lee Scandinaro, a member of the Allegheny College Gateway who is a part of the AmeriCorps VISTA project. You should interact with your customer to learn more about the features that `van-schedule` should provide. Additionally, one of your team members should investigate existing tools or online discussion forums that outline the features that are common to software tools like `van-schedule`. With that said, it is important to note that the `van-schedule` system is intended to support and, if possible, replace the current approach that is completed in a manner that is largely manual in nature. The basic operation of the tool will proceed in the following manner: Lee Scandinaro will use the system when interacting with student service leaders who need regular transportation to a site where they perform service activities. This system should accommodate for the fact that there is only one shuttle van available for transportation and that this van has a limited number of seats available to students.

For this final project assignment, all of the students responsible for requirements elicitation and analysis will interact with the customer at a time that is convenient for this individual. It is important to note that these students should not assume that the customer is a computer scientist who understands many of the intricacies associated with the algorithms normally used to schedule scarce resources. So, when a team member learns something important about the features of the system, then this person should share these details with all the other team members through an appropriate private channel in our Slack team. Since the ultimate goal for your team is to release a final version of `van-schedule` by the end of this final project assignment, you and your team should carefully listen to and, whenever possible, meet the needs of the customer. Ultimately, the customer will decide which system, from the three competing teams, will be adopted for use during subsequent academic semesters. Please see the instructor if you have questions about this matter.

Designing the System

For this final project assignment, teams are encouraged to design and implement a system that uses whatever programming languages, development environments, and execution means that are most suited to the knowledge and skills of the team's members and the goals of the customer. In light of this freedom, the descriptions used in this section and the following sections will often (but not always) be "generic" and thus not, for instance, refer to language concepts (e.g., "object-oriented" or "Java class") that are specific to a single programming language or development environment. With that said, students should consider whether or not it is feasible to implement this system in the Java language as you are most likely the most familiar with development, testing, and analysis tools that support Java. Teams that do not pick Java as their implementation language should justify this choice and ensure that they still throughly analyze and test their final product.

Working with the members of your team and leveraging the content in the requirements document, you should create a design for your system. As you are finalizing the system's design, you should try to develop answers to relevant questions such as: How many components will you use? What will be the relationship between the components? What functions will the components have? What will be the inputs and outputs of the functions? Is the design easy to understand? Can you actually implement and test this design? After answering these questions, you should use Markdown to write a design document with text and diagrams that explain the system. Since the main focus of this assignment is now not design diagrams, students may use any reasonable tool to create their diagrams and, if necessary, include them as separate files in their team's repository. When clarification of the system's specification is necessary (e.g., due to writing ambiguities), the designers should interact with the individual(s) who elicited the requirements. Yet, your team should carefully ensure that the system's requirements and design are correctly in sync.

Implementing and Testing the System

The developer(s) and tester(s) on your team should take the requirements and design documents and start to think about how the system will be implemented. Your task is to ensure that the program faithfully adheres to the descriptions already produced by other members of your team. When an aspect of the requirements and design documents is not clear, the developer(s) must talk with the people who created these documents to resolve any outstanding concerns. The creators of these documents must quickly commit any changes in them to their repository so as to best ensure that the requirements and design of the system are in sync with its implementation.

As this programming systems product will be released to GitHub at the completion of the assignment, the implementors and testers should take care to create a system that is well-documented through comments and other relevant documentation. Whenever it is possible to do so, these members of your team must also add (semi-)automated methods for verifying that the implementation adheres to the requirements that previously were elicited from the customer. For instance, if your team decides to implement `van-schedule` in the Java programming language, then you should test it with automated tests cases that you wrote in JUnit, the industry standard for automated testing in Java. Overall, as you are implementing and testing your system you should hold yourselves to a high standard under the assumption that other software engineers will review and use your code.

Evaluating Your System's Design and Implementation

In advance of releasing the `van-schedule` system, it is important for your team to evaluate the quality of its design. So, your team could use a tool called JDepend; as in a previous assignment, you can learn more about JDepend by visiting `http://clarkware.com/software/JDepend.html`. Before you start the next phase of this assignment please make sure that all of your team members fully understand all of the details about the design quality metrics that JDepend calculates. Your team can use these metrics to better understand the quality of `van-schedule`'s design and make suggestions, to your team members, for ways in which the design of your system could be improved. As one of the deliverables for this assignment, your team should use Markdown to prepare a document that contains intuitive definitions, suitable for your customer, that describe all of the design quality metrics and then comments on the relevant characteristics of your tool's design.

It is also essential for your team to calculate metrics that characterize the quality of `van-schedule`'s implementation. As you know from a previous laboratory assignment, JavaNCSS is a software tool that can automatically scan your Java source code and report information about the number of non-commented source code statements and the cyclomatic complexity. If you have not already done so, you can learn more about JavaNCSS by visiting `http://www.kclee.de/clemens/java/javancss/`. After reading the Web site for JavaNCSS, each member of your team should search for papers in the ACM Digital Library, available at `http://dl.acm.org/`, that formally describe the meaning of the metrics calculated by JavaNCSS, such as cyclomatic complexity. As one of the deliverables for this assignment, your team should use Markdown to write intuitive definitions, suitable for your customer, that fully explain all of the metrics calculated by JavaNCSS.

Once all of the members of your team understand the metrics calculated by JavaNCSS, you should download and install this tool. As you did with JDepend, now you must learn how to run JavaNCSS and then apply it to the source code of the system implemented by your external customers. What does the output of JavaNCSS tell you about the quality of `van-schedule`'s implementation? Does `van-schedule` exhibit good implementation characteristics? How can you use the values of these metrics to better understand and suggest modifications for the implementation of the `van-schedule` system? How much of `van-schedule`'s source code is for the program itself? How much of the source code is devoted to the test suite? Overall, how well is `van-schedule` documented? What are ways in which your team can improve `van-schedule`'s documentation?

Additionally, your external customer wants you to consider assessing the quality of the test suite(s) that you have implemented for `van-schedule`. For instance, your team can think about using a test coverage analysis tool like JaCoCo to calculate the code coverage of `van-schedule`'s test suite. Alternatively, you could use a tool like MAJOR to perform a mutation analysis of `van-schedule`'s tests; students who want to learn more about MAJOR and the concepts underlying mutation analysis are encouraged to read several of the papers in the "References" section of a past assignment and to visit MAJOR's Web site, available at `http://mutation-testing.org/`. At minimum, your team must document the baseline characteristics of the test suite. For instance, how many tests are in `van-schedule`? Do all of the test cases pass? Are there tests for all of `van-schedule`'s main components? Are you confident in your tool's correctness? Given the fact that the customer is, understandably, not an expert in software testing, your team should take every step possible to assess, guarantee, and intuitively characterize the quality of your system.

Along with running software tools, like JDepend and JavaNCSS, to automatically assess the quality of `van-schedule`'s design and implementation, you should now take time, as a team, to better explain and visualize the design of your team's tool. Following the outline of the unified modelling (UML) diagram in Figure 6.22 of SETP, your team should create a high-level package diagram for `van-schedule`. Next, your team should pick at least one major "sub-system" in `van-schedule` and create a UML class diagram, like Figures 6.18, 6.20, and 6.21, that shows the relationship among the classes in this sub-system. Please use a technical drawing tool to create these UML diagrams; students should consider using a tool like Mermaid, available at `https://github.com/knsv/mermaid/`, to program these diagrams and make them available in their repository. Finally, your team should prepare an explanation of at least one example of how `van-schedule` employs object-oriented principles (i.e., polymorphism, inheritance, and encapsulation); if you do not implement your final project in a language like Java then your team must still articulate the ways in which your code adheres to commonly-accepted principles for a high-quality design.

Preparing Your System for Public Release

You and your team members should prepare complete documentation, in the form of a `README.md` file that will display on the front page of your GitHub site, that explains how to use your programming systems product. Additionally, the `README.md` file should link to all of the other relevant documentation that analyzes your final project's design and implementation. One possibility is for your team to create a Bitbucket repository to support the initial stages of creating your system, with the intention of ultimately migrating to a GitHub repository before the submission of the final product. Alternatively, your team can agree to develop `van-schedule` is a more public fashion by creating a GitHub repository right away and then using this repository and, for instance, its issue tracking features to manage the creation of your tool. Ultimately, your customer should be provided with a link to a publicly accessible GitHub repository that contains all of the required deliverables; the customer should be able to follow the instructions on this GitHub site when attempting to use your tool. Please see the course instructor if you have questions about these important tasks.

Review the Textbooks for Key Ideas

To best ensure your team's success at completing this project, you should review all of the content in Chapters 1–6 of SETP, focusing on Chapter 3's relevant details about the process of developing software. Every member of the team should also review Chapters 1–4 of MMM, reconsidering topics such as the roles in a software development team and the techniques that help developers achieve conceptual integrity in a design. Refreshing your understanding of this content will best ensure that your team works well together when completing this assignment — and help you to prepare for the final project where you will deliver this large-scale system to an external customer.

Since the quality of your final product is critical to the customer's success in regularly using your program after its release, all of the members of your team should also read Chapters 8 and 9 of SETP. Since it focuses on the phases of the software testing process and the strategies for efficient and effective testing, this content explains the best practices that you can apply when working on this final project. In particular, you and your team members should pay careful attention to the development of test plans and the use of tools for automated testing and analysis.

Finally, since your team must deliver a working system to a customer, it is imperative that you read and understand the content in Chapter 10 of SETP. Your team may also wish to review SETP's material about software documentation (see Section 10.2) and technology transfer (see Section 14.2). Additionally, you may examine Chapter 15 of MMM to learn more about ways in which you can document software. Although this content in SETP and MMM reinforces concepts already introduced in past class and laboratory sessions, it will provide further context to support your release of a programming systems product to your external customer. Please see the course instructor if you have questions about any of these reading assignments.

Summary of the Required Deliverables

As mentioned previously, this assignment asks your team to deliver to your external customer, Lee Scandinaro, a full-featured programming systems product through a publicly accessible GitHub repository. This repository should explain every step that the customer must take to install and use your software to solve the stated problem. For the purposes of grading the final project, this assignment also invites your team to submit one printed and signed version of the following files:

1. A description of and justification for your team's chosen organization, roles, and tool support
2. A complete `README.md` file that fully explains how to use your programming systems product
3. A requirements document that describes the key features that your customer requested
4. A document that intuitively explains the meaning of the metrics for design and code quality
5. A full-featured written analysis of the values of all the relevant metrics for your system
6. Technical diagrams that correctly visualize the architecture and design of your system
7. A written explanation of how your system evidences the principles of high-quality design
8. A complete demonstration of your programming systems product to your customer
9. A separately submitted short review of your demonstration written by your customer
10. A separately submitted written evaluation of the work that you and your team completed