**CMPSC 290**
**Principles of Software Development**
**Fall 2013**

**Laboratory Assignment Five: Specifying and Implementing a Next Release Planner**

## Introduction

In the previous laboratory assignments, you have learned about the tools that we will use during the remainder of this semester to specify, design, implement, test, and document Java programs. You have also had several experiences with working in progressively larger teams to complete the phases of the software life cycle. Moreover, our past class sessions have introduced you to the key concepts associated with software engineering, with a recent focus on the elicitation of software requirements. In this assignment, you and your team will follow the phases of the software development life cycle and employ the concepts that we have studied in class to implement and test a program that can be used to efficiently determine how to release the next version of a software product. One week from now, you will describe and demonstrate your next release planner in a formal presentation.

## Next Release Planning

Managers often use a software process in order to make decisions about when to release an application. If you were a manager and you were charged with determining which features to include in the next release of a program you could use a next release planner to efficiently make an intelligent decision. For this laboratory assignment, you should assume that a next release planner accepts as input for each potential requirement $R_j$, (i) $C_j$, the cost associated with implementing the requirement and (ii) $B_j$, the monetary benefit for a program that contains this feature.

In order to determine which requirements will be part of the next release of a software application, a manager must choose from the requirements $R = \{R_1, \ldots, R_n\}$ and ensure that (i) the implementation tasks are completed at no more than the total fixed cost $C$ and (ii) the chosen requirements maximize the total monetary benefit that the company will see when it releases the software product. Given cost and benefit information for each $R_j$ and the fixed cost $C$, a next release planner will pick which requirements are included in the next release of the program.

Next release planning is equivalent to a well-known NP-hard combinatorial optimization problem. What is the equivalent problem? What are the heuristics for solving this problem?

## Observations About the Requirements

You are responsible for implementing a next release planner that should adhere to the requirements outlined in the previous section. Your planner should be implemented in Java and execute as a stand-alone program on the command-line. It must accept command-line arguments for (i) the set of requirements $R = \{R_1, \ldots, R_n\}$, (ii) the set of benefits $B = \{B_1, \ldots, B_n\}$, (iii) the set of costs $C = \{C_1, \ldots, C_n\}$, and (iv) the fixed cost $C$. Unless you decide otherwise, the planner may assume that all of the values in $R$, $B$, and $C$ are integers. All of the command-line arguments must be recognized, verified, and parsed using the JCommander tool available at `http://jcommander.org/`. The system must be able to efficiently solve large instances of the next release problem.

You must specify, as formally as is possible, the next release planner that your team will implement. Whenever it is justifiable to do so, your requirements document must adhere to the IEEE standard for software requirements specification. Through team discussions and interactions with your customer, you must define correct requirements that adhere to all of the characteristics of good requirements. For instance, beyond being correct and consistent, your requirements also must be testable and traceable. Using LaTeX, you should write a requirements document that fully explains the inputs, outputs, and behavior of the next release planner.

## Designing the System

Working with the members of your team and leveraging the content in the requirements document, you must create a design for your system. As you are finalizing the object-oriented design, you should try to develop answers to relevant questions such as: How many classes will you use? What will be the relationship between the classes? What methods will the classes have? What will be the inputs and outputs of the methods? Is the design testable? After answering these questions, you should use LaTeX to write a design document with text and diagrams that explain the system. Adhering to the JavaDoc standard, you should create extensive documentation for your classes.

## Implementing and Testing the Program

Using the requirements and design document, your team must implement and test the next release planner. You should focus on implementing a next release planner that is both correct and efficient. Just like in the previous laboratory assignments, your implementation must include the following:

1. A version control repository that contains all of the artifacts for the project
2. A build system that can build, clean, test, document, and run the program
3. A high-coverage test suite that effectively tests all of the classes in the program
4. A coverage report that was produced by the JaCoCo coverage monitoring tool
5. Fully documented Java source code that completely fulfills the requirements

Since you cannot exhaustively test this application, you must decide what types of inputs you will create in the test cases. You will also need to determine how you will know that the output of the next release planner is correct. Finally, make sure that your tests cover all of the requirements.

## Summary of the Required Deliverables

This assignment invites your team to submit one printed version of the following files:

1. A description of and justification for your team's chosen organization, roles, and tool support
2. A document that clearly specifies the inputs, outputs, and behavior of the next release planner
3. A document that explains the planner's design, with details about classes and methods
4. All of the implementation artifacts (e.g., build system, source code, and the coverage report)

You must also ensure that the instructor has read access to your Bitbucket repository that is named according to the convention `cs290F2013-lab5-team`*k*, with *k* representing the number of your assigned team. Your repository should contain all of the deliverables that you produced during the completion of this assignment. Please see the instructor if you have any questions.