Informatics 111 / CSE 121: Software Tools and Methods
Fall Quarter 2007

Assignment 1

Due Tuesday, October 30th, 2007, at 11:59pm

Part 1. Effort Estimation (30 points)

Effort estimation is the practice of making accurate estimates of how long it will take to complete a piece of work. People in software are notoriously bad at this; many routine projects run over their schedules and budgets. The objective of this question is to build up your skill at making estimates of how long it takes you to complete a task. This question will be repeated on each assignment to ensure that you go into industry with a substantial data set about yourself. The basic plan for this task is to make an estimate of how long it will take to complete the assignment (in advance), record how long you spent on various activities, analyze this log, and calculate your error. This sequence of steps simulates the billing process in a consulting environment.

1.1 A priori estimate. (2 points) When you start working on this assignment, write down an estimate of how long it will take you to answer all of the questions. (You will not be penalized for an inaccurate estimate; the purpose of this exercise is for you to become better at effort estimation.) This estimate should be a single number (in hours).

1.2 Logging. (16 points) As you are working on the assignment, record what you are doing and how long you spent. For this question, quantity of data, i.e. number of entries, is important. As a rule of thumb, you should add a log entry every time you switch tasks. For example, if you do something for two hours straight, that can be one log entry. However, if you do two or three things in half an hour, you must have a log entry for each of them. You do not need to include time for logging itself, but should include the time spent answering the other sections of Part 1. (Try to avoid infinite recursion!)

Each task should be assigned a category. You can decide on your own categories. Here are some suggestions: Project Management; Planning; Research; Tool Learning; Programming; Debugging; Writing. You may add your own categories if your activities don't fit well into those suggested.

Here are some example log entries, showing the format of your log. Your logging is not limited to the sample entries. Add ones appropriate for your own tasks.

Items	Date	Part	Description	Time	Category
1.	10/10/06	1	Read assignment and create estimate	20 min	Project Management
2.	10/12/06	2	Study WebAnalyzer code	45 min	Planning
3.	10/13/06	2	Develop Junit tests	35 min	Programming
4.	10/15/06	2	Develop Junit tests	130 min	Programming
5.	10/16/06	3	Study refactoring handout	20 min	Research
6.	10/17/06	3	Refactor code	120 min	Programming
7.-15.					
16.	10/22/06	1	Analyze log entries and calculations	45 min	Project Management
17.	10/22/06	All	Create PDF and Submit assignment	40 min	Project Management

1.3 Analysis. (6 points) Tally up the time spent in each of the categories and on the entire assignment. Calculate summary statistics, make a table, and use Excel to plot a bar graph by category.

Here's what a table of summary statistics might look like.

Category	Time	Percentage of Total
Project Management	1.2 hrs	7.2%
Planning	3.5 hrs	22.8%
Programming	6.5 hrs	35.8%
(more)		
(more)		
Total	11.4 hrs	100.0%

1.4 Error Calculation. (2 points) If you had written a contract with your client based on your initial estimate, you'd want to determine at the whether you made a profit on this project. Calculate the percentage error of your initial calculation. (This is the number of hours, or minutes, that you were off, divided by the initial estimate. Your error should be negative if you came in under budget.)

1.5 Discussion. (4 points) What did you learn from this exercise? Were there surprises in how long various activities took? Are you good at estimation? Will you use the knowledge you've gained in planning future assignments?

Deliverables

Your deliverable for Part 1 is an Excel spreadsheet named Effort.xls, with three tabs (worksheets).

- An initial estimate of hours, a single number. (Tab 1)
- A log of your time spent on the assignment, with categories. (Tab 2)
- A table with the time totaled by category. (Tab 3)
- A bar graph showing time spent by category. (Tab 3)
- Comparison of estimate vs. actual. (Tab 1)
- Discussion of result. Put this in Report.doc or Report.pdf (see below).

Part 2. Creating Test Cases (30 points)

Parts 2, 3, and 4 of the assignment all involve the LunarLander code available at http://www.ics.uci.edu/~michele/INF111/Informatics.html (LunarLanderAssignment1.zip).

For Part 2 you will use JUnit and Eclipse to create, execute, and report test cases for the LunarLander system. At this point you shouldn't be changing any of the LunarLander source code, so if you find bugs report them but do not fix them. Our goal is to understand the current behavior of the code base.

To write a reasonably complete set of JUnit tests for even a small system like LunarLander is a substantial task. For this assignment, your guidelines are: have at least one JUnit test case for each method in every class (if possible); and focus your efforts on the Processor class. It is not required to test the Interfaces such as IObserver and ISubject.

As you write the test cases you will probably feel the need for guidance as to what is "correct" behavior. In general, assume that the code is working correctly. You are also welcome to ask in lecture or discussion. Finally, if you need to make an assumption, document it in your report.

Deliverables

Your deliverables for Part 2 will be a zip file named JUnit.zip containing the JUnit source code, and a report named Report.doc or Report.pdf. The report should provide:

- a description of the JUnit test cases you developed (explain why any methods lack test cases if that is the case),
- discussion of any test cases that LunarLander fails,
- report of any assumptions you made about the system requirements.

Part 3. Refactoring (35 points)

You've probably noticed that the code isn't a paragon of good practices.

For Part 3 you will refactor the code to improve its readability, maintainability, and conformance to object-oriented design principles. Remember that in refactoring you don't add or change functionality, or fix bugs.

The assignment is to make six to eight refactorings of the source code. You can use the built-in Eclipse refactor options, and/or you can refactor by hand following the approaches discussed in lecture. You can also research other refactoring techniques, or develop your own.

Deliverables

Your deliverables for Part 3 will be a zip file named Refactored.zip containing the refactored LunarLander source code, and a report named Report.doc or Report.pdf (same report file as in Part 2). The report should provide:
• a description of each refactoring you did, including the motivation,
• discussion of any issues or problems that arose.

Part 4. Regression testing (5 points)

It's possible that your refactoring inadvertently introduced some functional changes to the LunarLander system.

For Part 4 you will rerun the JUnit test cases you wrote for Part2, and ascertain that the code still functions in the same way. If you do find functional changes, note this in your report and go back to the refactoring stage to correct the refactoring.

If you changed class, method, or variable names or roles, it is possible that your JUnit test cases won't work with the refactored code. In this case you do not need to rewrite the test cases; just note the issue in your report.

Deliverables

Your deliverables for Part 4 will be further discussion in your Report.doc or Report.pdf. The report should provide:
• a list of any refactorings you redid or fixed after running regression testing,
• discussion of any test cases that could no longer be run with the refactored code.

Handing In Your Assignment

Your assignment must be submitted electronically to checkmate.ics.uci.edu. (CSE 121 students, please send in your assignment through Informatics 111.) You will submit four files.

1. Your reports in Report.doc or Report.pdf.
2. A zip file called JUnit.zip
3. A zip file called Refactored.zip.
4. A spreadsheet called Effort.xls.

Do not zip these four files into one big .zip file. Do not use .rar instead of .zip.