Modern Methods of Software Engineering (ID2207) Final Project

Submission deadline: 2014-10-16

Problem Description ("Car Insurance Company")

Insurance company X processes claims which result from traffic accidents with cars where customers of X are involved in. Therefore, it uses the following procedure for the processing of the insurance claims: Every claim, reported by a customer, is registered by an employee of department CD (CD =Car Damages).
After the registration of the claim, the insurance claim is classified by a claim handler of A (high ranked officer) or B (low ranked officer) within Car Damage department. There are two categories: simple and complex claims. For simple claims two tasks need to be executed: check insurance and phone the garage (which repairs the car). These tasks are independent of each other. The complex claims require three tasks to be executed: check insurance, check damage history and phone garage (which repairs the car).

These tasks need to be executed sequentially in the order specified. Both for the simple and complex claims, the tasks are done by employees of department CD.
After executing the above mentioned claim handling steps, a decision is made. This decision is made by a claim handler A and has two possible outcomes: OK (positive) or NOK (negative). If the decision is positive, then insurance company X will pay. An employee of the finance department handles the payment. In any event, the insurance company sends a letter to the customer who sent the claim. An employee of the department CD writes this letter.

Insurance Claim Processing (in more detail)

When a claim is received, an employee first checks whether the claimant is insured by the organization or not. If not, the claimant is informed that the claim must be rejected. Otherwise, the severity of the claim is evaluated. [Based on combination of <u>price of car, cost of damage, previous history of accident</u> and maybe claim handler's personal opinion decide whether a claim is severe or not]. Based on the outcome (simple or complex claims), relevant forms are sent to the claimant. Once the forms are returned, they are checked for completeness. If the forms provide all relevant details, the claim is registered in the Claims Management system.

Otherwise, the claimant is asked to update the forms. Upon reception of the updated forms, they are checked again. (@original scenario developed by Prof.Wil van der Aalst , *Eindhoven University of Technology)*

In general the System should handle the followings:

1. You do not need a database. It is ok if data is stored in file or memory and disappears when the program is shut down.
2. You should have a user interface to be used by different employees of Insurance company. This means that you need to have kind of authentication and authorization mechanism to allow and control access of users to different parts of the system.
3. We would like to automate the entire processes of Car Insurance as much as possible. Specifically, the claim handling (both simple and complex types) need be automated. The interface to Garage and Customer (car owner) need to be automated by considering Web Form, SMS or E-Mail (You don't need to be implementing sending WebForm, SMS or E-Mail, just consider a simple interface that simulates them).
4. You need to automate the payment process of finance department to issue payment orders and keeps record of payments.
5. In the system, you have to keep record of all customer interactions with the company (i.e. personal records of customer, car insurance record and related documents, history of previous claims, etc).
6. Any employee of the system should be able to see current status of a claim at any stage.
7.… There could be number of other things which could be done, but we limit our problem to this description only...'

Project Description

In this project you are supposed to apply several elements of the Extreme Programming approach (XP) to solving the problem of implementing the system described above. Only a *subset* of XP practices/elements will be used, because of the difficulties with complete modeling an XP process in the framework of a small course project. For more details about XP we refer to the lecture notes and to http://www.extremeprogramming.org/start.html.

Remember that you must solve this problem in groups of two students. We will not accept any project done by only one student. XP practices depend on this to be meaningful.

Note: We are not doing Object Oriented Analysis and Design as we did in previous homeworks in this project. We follow XP approach.
The main elements of XP in this project will be:

1. Developing *user stories*,
2. *Release* planning,
3. *Iteration* planning,
4. Selecting a *system metaphor* that could be suitable for the problem solving,
5. Developing a system in a *test-driven* fashion for selected user stories,
6. *Refactoring* the programs,
7. *Pair programming.*
8. Daily Stand-up meeting

Project Task

Develop a set of user stories (remember that they are *not* use cases and they are much simpler). You should **write all user stories** for your system **(excluding User Interface user stories)**. <u>The stories should be meaningful and non-trivial, otherwise they will not be counted!</u>

1. Make an estimation of the time it takes to implement each user story.

2. Select a set of user stories for the first release, **roughly 60%** (at least 25). A release is a version of a system that is stable enough and has enough new features to be delivered to end users. The chosen user stories for the first release should include main functionality of your system.

3. Divide the development of the first release into at least 3 iterations. Select stories to be implemented in each of these iterations.

   Adjust the iteration duration to your current time constraints by starting from simpler user stories. BUT at the end whatever you have developed should be seen as integrated set of user stories which demonstrates full/partial functionality of your system (not islands of unrelated/far-away user-stories).

4. Write a metaphor.

5. Perform the planned iterations.

● It is very important that you actually finish each iteration. When iteration is finished there must be no “almost done” stories. The stories of the iteration must be written and successfully tested.

● Use test-driven programming. First write a test, then the implementation necessary to compile and run the test. You only have to write tests for the model, not for the user interface. **You are not allowed to use any tools to write the tests and perform unit testing on behalf of you. You SHOULD write the test cases by yourself.** *You are free to choose your programming language***.**

If there would be a piece of code in your deliverables project code which do not have corresponding and appropriate test-case, We conclude that you have not followed test-driven programming approach !

● Do at least 3 refactoring in each iteration.

● Use pair programming.

● Have daily meeting to discuss the project progress
It roughly takes 2 full days, for two skilled programmers to develop the system, for some it may take longer!

Deliverables

You should deliver the following:

1. A set of developed stories with time estimates.

2. A set of selected stories for the first release. Explain your selection by considering importance and risk factors.

3. Iteration plan that is a list of which stories you implement in which iteration.

4. The metaphor.

5. Description of your test-driven pair programming process and applied refactoring. Also describe how well you managed to estimate what should be done in each iteration. Write the truth! In order to pass you must show that you can draw conclusions of your mistakes, not that you have done everything perfectly.

6. The source code (**including source code for Test Cases**), and a *readme.txt* file that explains what is needed to compile and run the program.

7. Write acceptance tests for two of the user stories you implement. You must choose stories that take input and give output through the user interface. The chosen acceptance test should address the main functionality of your system.

8. Report of daily stand-up meetings!

• **Deadline: 16 October 2011 - 11:59 PM**

• You need to present your project. Presentation date will be announced later!

• **Send your deliverables with subject "*MMSE14- Project*" to** siskos.filotas@gmail.com, niksta@kth.se and misha@kth.se. **Please avoid using different Subjects – Our email filters cannot match your intelligence.** The project must be done in groups of two students. We will not accept any project done by only one student.

Appendix

1- What is stand up meeting ?

(http://www.extremeprogramming.org/rules/standupmeeting.html)

You are having daily meeting (usually in the morning as the beginning of the working day) with all group member discussing the project goals, issues and progress." During a stand up meeting developers report at least three things; what was accomplished yesterday, what will be attempted today, and what problems are causing delays. The daily stand up meeting is not another meeting to waste people's time. It will replace many other meetings giving a net savings several times its own length".

2 -Example User Story: Login

At Startup & When the system starts, the user interface will present a login screen. The user can now log in using its user name and password. After verification the user is presented with the set of actions it can perform.
Time estimate: GUI 0.5 hour, User Account: 0.5 hour

3- What is Acceptance Test?

"At the very least, an acceptance test could consist of a script of user interface actions and expected results that a human can run. Acceptance tests can be automated, either using the unit testing framework, or a separate acceptance testing framework.", [John Brewer].For those who are coding in Java, this tool could be helpful： http://exactor.sourceforge.net/