cs3307a – Object oriented analysis and design

System Enhancement Project

Project Overview

Computing systems grow over time, especially if they are useful to the stakeholders. (I say, "Systems that aren't changing are probably dying".) This means that new requirements need to be implemented over previously implemented requirements. In turn, this entails design and code enhancements, and testing.

So, let us assume that our banking system (Assignment 1 and Quiz) is growing. Specific enhancements are described below and form the central part of the enhancement project. For example, we introduce the use of a credit card for making purchases and have a monthly payment scheme. These features are added on top of the base banking system we already have. In addition, there is a separate vendor's point-of-sale system that communicates with the bank on the purchases made and for validating the use of the card. These enhancements will be designed and implemented.

In addition, software engineering theories indicate that software "inspections" complement software "testing" in developing quality software. Whereas testing works on "code" and "point-wise" analysis (i.e., through specific test cases) using testing tools, inspection works on (not only code but also) higher-level artefacts (e.g., design, architecture and requirements) using humans' cognitive analysis of a broad range of issues in development. In this project, we are concerned with OO "design" inspection. For example, does design satisfactorily implement the requirements? Is design easy to understand? Is design reusable? Etc. We will use a design inspection "instrument" (i.e., design questions) to analyse our design and gather feedback. Such feedback can be helpful in improving the system's design.

The concern for quality does not end with design inspections. There is a school of thought that says that design is not only "qualitatively" analysed and improved (e.g., through inspections) but that design can also be measured "quantitatively" using suitable metric tools. Example OO design metrics are: amount of coupling between classes, amount of inheritance, lack of cohesion in classes, number of methods per class, etc. In this project, we will use a metric tool (CCCC) to measure the quality of our banking system design, which we will then interpret as feed into design improvement.

Finally, we will examine the enhanced design of the banking system and the vendor system and introduce a design pattern in this context.

In summary, we will enhance the banking system and create a vendor system, conduct design inspections, measure design quality, and introduce a design pattern in the context of the new

systems. This project is thus aimed at contributing towards a number of learning outcomes described in the course description:

- Experience with the programming language C++.
- An understanding of the object model during programming, design and analysis phases.
- Experience with how to classify objects from given problem descriptions and how to make choices considering quality attributes.
- Experience with design patterns.
- Experience with OO metrics, use of metric tools, interpretation of measures, and the role of metrics in software design.
- Experience with design inspections.

Enhancement Description

The banking system resultant from Assignment 1 and Quiz is to be enhanced with some new features. In particular, each client with a chequing account is given a credit card by the bank for making purchases from a vendor. There is a client-dependent credit limit on the credit card. Bills are paid monthly from the chequing account. Also, there is a separate point-of-sale system operated by the vendor. Purchases are notified to the bank. The following describes the bank and vendor system requirements.

The banking system:

- Client-specific credit limit is decided, and is assumed to be preset, by the bank manager in bank's records.
- Triggered by the "end of the month" as an externally generated event, the credit card amount owing to the bank, by each client, is paid from the chequing account according to the specifications in the client records (i.e., pay full amount or pay minimal amount (10% of the full amount)). Unpaid portion accrues an interest of 2% per month which is added on to the amount owing after deduction of the minimal amount. Insufficient funds in the chequing account for paying the credit card bill results in: (i) a notification to the manager via a file of failed payments that the manager can view upon logging into the system, and (ii) freezing the credit card from further purchases until the debt (according to the specification in client record) is paid off through one or more deposits at which time the credit card is automatically unfrozen.
- A client is able to view his/her purchases made in the current month through ATM. This is itemised chronologically and the total of all purchases is also given. If the total amount of all purchases is more than 75% of the balance in the chequing account, the client is notified at viewing time. This helps in moderating his/her purchases in the month and in ensuring that there are sufficient funds to pay the credit card balance at the end of the month.

The vendor system:

- This is an independent point-of-sale system where purchases are made with the use of a valid (unfrozen) credit card. Frozen credit cards are rejected with an appropriate message to the client (display screen). We assume only one vendor.
- Credit card purchases at the vendor:
    - Use of a PIN code for authentication, which is the same as for accessing an ATM.
    - A random number generator produces an amount between $1 to $100 per sale. This is deemed to be the price of the item purchased, including all taxes.
    - Client accepts or cancels the sale. In either case, the user is logged out of the point-of-sale system. In the case of "accept", however, the amount is recorded in the vendor's database (for internal purposes) and also sent to the bank where the client's purchase-records are kept. The vendor shares with the bank the purchase data for a client through a shared database. Currently, the credit limit on a card is not enforced at the point-of-sale.

Disclaimer:
Major gaps, if any, in the requirements are to be reported to the instructor, which may result in specification changes or workaround. Minor gaps, if any, are to be filled in by the project team which is deemed to have internal Quality Assurance processes to certify specification changes.

Tasks to be done

System enhancement [10%]: Taking the banking system design (class diagram) resultant from 2 (iii) of the Quiz – "**Improve design**: **From ....**", please modify this design with the enhancements described above. Likewise, the program code from Assignment 1 is to be enhanced to implement the new features in C++. The vendor system is also to be designed and implemented in C++. This is independent of the banking system though it communicates with the banking system for purchases.

Deliverables:

[1] Enhanced design of the banking system and design of the vendor system.
[2] C++ code implementing [1].
[3] Explanation of the correspondence between code and design.
[4] Operational evidence (through execution traces) of the following scenarios:
- Purchase made at the vendor (valid credit card). Bank notified of transaction.
- Purchase made at the vendor (frozen credit card). Customer notified.
- Customer views current month's purchases at ATM.
- End-of-month event trigger:
    - Successful credit card payment.
    - Failed payment (credit card frozen)

Design inspection [10%]: Exchange (a) the design resultant from 2 (iii) of the Quiz – "**Improve design**: **From ....**" and (b) the program code from Assignment 1 with the "Inspector group". (The inspector group and the developer group pairs will be imminently announced).

Ideally, we want to include new design and code in the inspection process as well but there is a risk that some new design and/or code may not be available in time for inspection. Thus, for learning purposes, we limit ourselves to the design and code from the previous version of the banking system.

As "Inspector group" for the specified development group, you are asked to inspect their design using the given inspection instrument. Use the given program code as necessary for the purpose of assessing the design (e.g., association between two classes in the design can be verified by examining method calls across two classes; depth of inheritance can be examined in code; too many methods in a class can be similarly examined; etc.). The analysis recorded in the inspection sheet will be the basis for assessing the quality of an inspection. The inspection process and instrument will be discussed in the class.

Deliverables:

[5] Design inspection instrument filled out with analysis and findings.

Metrics and Interpretation [Assignment – 10%]: Here, the idea is to produce OO measures of the program developed. Note that we will have two releases of the banking system. One was created for Assignment 1 (let us call it "R1") and another, the enhanced version in the project (let us call it "R2"). You can ignore the vendor system for this quantitative analysis. Thus, we produce the measures for R1 and R2. See the lecture slides "CCCC Metric tool" for instruction on downloading the CCCC tool and producing the measures (i.e., generating the "**CCCC Software Metrics Report"**). We will also discuss this in the class if not already.

Following report generation, we depict historical measures for the two releases as shown in slide #19 of the "CCCC Metric tool" lecture notes. This is better created as a spreadsheet since it is unbounded.

Based on the measures obtained for the two releases, please identify for each of the OO metrics (WMC, DIT, NOC, CBO):

- *Danger*: Which classes in release R1 (respectively, R2) have values that exceed the threshold value defined as being dangerous for that measure?
    - Please highlight these class-metric cells in RED.
- *Caution:* Which classes in release R1 (respectively, R2) have values below the danger threshold but still above a second lower threshold which has been laid down to indicate cause for concern?
    - Please highlight these class-metric cells in YELLOW.
- *Trend:* Are there any emerging quality trends across the two releases for different classes and different metrics? For example:
    - Classes that were under quality control in release R1 but have now entered cause for concern or are in danger in release R2?

- Likewise, classes that were of concern or in danger in release R1 but have now improved and are now out of such concern or danger in release R2?
- In either of the above two cases, what can be inferred from the system design and code for release R1 and R2 that helps explain the trend?

Deliverables:

[6] "**CCCC Software Metrics Report"** for releases R1 and R2.

[7] A spreadsheet depicting historical OO measures for releases R1 and R2 with colour codings (RED and YELLOW) for danger and caution cases.

[8] Quality trend across releases R1 and R2.

Design Pattern in Context: [Assignment – 5%] Assignment 2 required creating your own scenarios and implementing design patterns in them. Here, we take the context of the enhanced banking project (together with the vendor system) and introduce a design pattern in the *design* document of these systems. Coding is not expected. Specific tasks are:

- Take the *design* of the enhanced banking system and the vendor system and introduce a design pattern in that context.
- You can introduce any design pattern you wish. However, the following illustrates a couple of possibilities:
  - *Adapter pattern:* The vendor and banking systems are each assumed to use different kinds of databases. In order to use the credit card for purchases, information must be sent and received between the vendor and banking systems. However, the information is stored differently within their respective databases.

    An adapter can be responsible for ensuring that information is in the format of the receiver. For example, the purchase details saved in the vendor's database must be converted into the format that is compatible with the banking system's database prior to being sent to the banking system.

  - *Strategy Pattern:* When it's time to pay the credit card amount due to the bank, the banking system can take different courses of action: pay full amount or pay minimum amount (10%) as per the client's preferences. Each course of action can be moved into its own strategy class that is responsible for appropriate actions. For example, in the case of minimum payment, the strategy class should calculate 2% interest of the owing amount after deducting the minimum. Yet exceptional handling would call for freezing the credit card until the credit card balance has been paid off.

Deliverables:

[9] A modified *design* of the banking and vendor systems, that includes the design

pattern.

[10] An explanation of the design pattern and how it works in the context of the banking and vendor systems.

Deadline for submission: 3rd December, 2014 at 23:59:59 hours.

p.s: If anything is not clear, please do not hesitate to ask!