Final

Note

This is an open-book exam. You may consult any book, paper, note, or on-line resource, but discussions with others (in person or via a network) are strictly forbidden.

How to Submit Your Answers

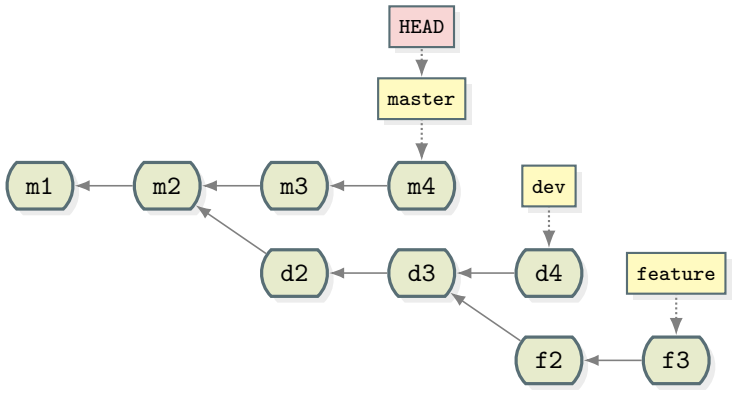
Please use a word processor or scan hand-written answers to produce a single PDF file. Name your file according to this pattern: "b067050xx-final". Upload the PDF file by the due time to the NTU COOL course site for Software Development Methods 2021.

Problems

1. (20 %) Consider a Git history with three branches, namely master (pointed to by HEAD), dev, and feature, as shown below. The Git history contains commits m's, d's, and f's. There are two assumptions:

   - There is no change in the working tree to be committed.
   - A commit remains after a Git command if the commit can still be reached from some branch. Otherwise, the commit is removed immediately.

   Write down the Git commands to incorporate changes from feature into dev and then from dev into master. At least one rebase command must be used. Draw the final Git history as well. You may use x' to denote a commit with the same changes as in x but with a different parent in the Git history.



2. Suppose you are working on a file manager application for a cloud storage service. The remote files work like local ones to the operating system, but the file contents are actually stored on the remote server and are transferred over the Internet when necessary.

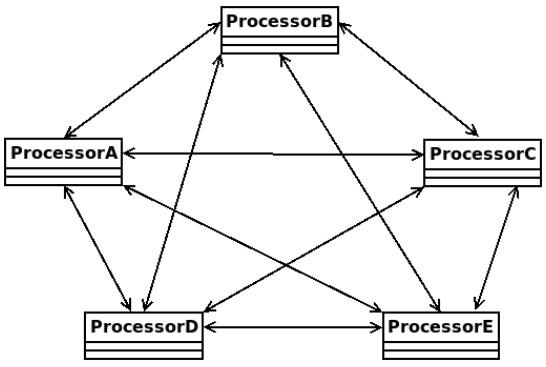
   Here is the interface of the remote file:

```
1  class RemoteFile {
2  public:
3    RemoteFile(const string& path);
4    virtual long Length() = 0;
5    virtual long Read(long offset, char* data, long size) = 0;
6    virtual long Write(long offset, char* data, long size) = 0;
7  };
8
9  class RemoteFileImpl : public RemoteFile {
10     (implementation of Length(), Read(), Write(), etc.)
11 };
```

(a) (4 %) Suppose you want to cache the file contents that were accessed previously to save network bandwidth. You want to introduce the cache without modifying every usage of RemoteFile in your code. What design pattern can be used to implement the cache by transparently controlling the access to the concrete RemoteFile in your code?

(b) (6 %) Please provide your design in a UML class diagram.

3. Suppose you are working on a data processing application. The application has a set of built-in data processor classes that transforms the input and then passes the output to the next processor depending on the output data. For example, input 1 is processed by processors A, B, and D consecutively, to get the final output. Input 2 is processed by processors B, A, D, C, E, and C, consecutively, to get the final output. The application uses the following design to make it possible to route the transformed data to the next processor:



Now you are adding more processor classes to the application but find this design inflexible: you need to modify all existing processors to connect the new processors to the existing ones. It's time to pay this technical debt and refactor the design.

(a) (4 %) Which design pattern can be applied to solve this problem, to make it easier for new processors to interact with existing processors?

(b) (6 %) Please provide your design in a UML class diagram.

Suppose now you are working on a design that allows the users to write their own custom processor class. Here is how the custom processor class is used:

```
1 class AbstractProcessor {
2 public:
3   virtual Data* Process(const Data* input) = 0;
4 };
```

You provide an implementation of the following class for creating your processor:

```
1 class ProcessorPlugin {
2 public:
3   virtual AbstractProcessor* CreateProcessor() = 0;
4 };
```

A custom processor is created during application startup as follows:

```
1 vector<ProcessorPlugin*> registered_plugins;
2
3 void LoadPlugin() {
4   for (ProcessorPlugin* plugin : registered_plugins) {
5     AbstractProcessor* custom_processor = plugin->CreateProcessor();
6     // Then add |custom_processor| to the core of the application
7     // to be invoked later.
8   }
9 }
```

(c) (2 %) Which design pattern is used in class ProcessorPlugin to let LoadPlugin() instantiate an AbstractProcessor implementation without knowing its concrete class?

Suppose you implemented a concrete AbstractProcessor class to anonymize sensitive user data. Then you added a concrete ProcessorPlugin class to instantiate the concrete AbstractProcessor class.

(d) (3 %) The processor is instantiated in the concrete ProcessorPlugin in LoadPlugin() during application startup. The Process() implementation of your processor is invoked in the core of the application later when necessary. Which design pattern is used here?

4. (10 %) Why do we need both verification and validation in an adequate software development process? Please explain the reasons using examples.

5. (20 %) Construct an abstract data model for the online order system of a frozen-food store that has to meet the following requirements:

   - The store carries several hundred kinds of frozen food.
   - Each order may include at most ten items; every pack/box of the same food counts as one item.

- Some food may be out of stock and hence cannot be ordered at times.
- A customer may request a delivery address different from her/his home address, for an order.
- A customer must have a full name, an email address, a phone number, and a home address.
- The store offers a membership program for customers.
- An expedite delivery may be requested with an added delivery charge, but it is free for members.

Please use the UML as much as possible when describing the model. State the assumptions, if any, you make for your design.

6. (5 %) Why is a white-box testing not necessarily static?

7. Please provide a precise description, using logical formulae, for each of the following requirements. The functions/constants and predicates you may use are: $+$, $-$, 0, 1, $<$, $=$, $\leq$, plus those introduced in the requirement statements. Make assumptions where you see necessary.

   (a) (10 %) The returned `result` is correct for a search function that is supposed to find the first occurrence of the value of `target` in an array $A[0..N-1]$ of integers; we stipulate that `result` $= -1$ if the value of `target` cannot be found in A.

   (b) (10 %) The array $B[0..2N-1]$ (of integers) is a merge, as in the merge sort, of the two halves of A, i.e., $A[0..N-1]$ and $A[N..2N-1]$, which have been sorted in increasing order.