20th CSEC – Past Year Paper Solution *2017-2018 Sem 1*
CZ 3002 – Advanced Software Engineering

Solver: Yew Shao Jie

1)

a) **Lack of automated source-code control**. Without automated source-code control, it exposes the project to needless risks. The developers have to coordinate their work manually. People develop new code to out-of-date interfaces and then have to redesign their code when they discover that they were using the wrong version of the interface. This results in unnecessary time-consumption. This can be resolved by using automated source-code control. This removes the need to manually intergrate different versions together. Moreover, the mistake made by the user can be prevented as the source-code control tracks the different versions and hence it is unlike that the developer will merge the wrong version. If the developer did merge the wrong versions, the cost in redoing the merging will not be as great as doing it manually.

b) The agile development method is a light weighted methodology that is suitable for small sized project in a changing environment because it delivers working software quickly and evolves quickly to meet changing requirement in a changing environment. Not possible for large projects as maintenance may be a problem due to agile characterizing that reduces the overheads in detailed documentation.

c) ISO 9126 is an international standard for the evaluation of software. The standard is divided into 4 parts which addresses, respectively, the following subjects: quality model; external metrics; internal metrics; and quality in use metrics. Using the ISO for derivation of system requirements brings clarity of definition of purpose and operating capability. This help designer to identify the tradeoffs between 2 or more characteristics when designing the system. These tradeoffs need to be identified, so that informed design decisions can be made.
The importance of having standards is that, it helps to adhere to best practice by avoiding repetition of past mistakes. Furthermore, have a framework for quality assurance processes.

d) External Quality attribute shows the usefulness of the system as perceived from outside. It provides customer value and meets the product owner's specifications. This quality can be measured through feature tests, QA and customer feedback. This is the quality that affects your client directly, as opposed to internal quality which affects them indirectly.
While an internal quality attribute is a measurement of how the system has been constructed. It is a much more granular measurement and considers things like clean code, complexity, duplication, component reuse. This quality can be measured through predefined standards, linting tools, unit tests etc. Internal quality affects your ability to manage and reason about the program.

2)

a)

i) MED ① login – 1 inquiry [Note that even though there are 3 different users, the same login page is used. Hence, only 1 inquiry instead of 3.]
LOW ② Create an account – 1 input, 1 logical file
HIGH ③ Create an account – 1 input, 1 logical file and social network system – 1 interface
LOW ④ Edit profiles – 1 input, 1 logical file

MED ⑤ Check-in on the event site using QR code – 1 input, 1 logical file, 1 interface
LOW ⑥ Search for events – 1 inquiry
LOW ⑦ View details of event – 1 inquiry
LOW ⑧ Register event – 1 input, 1 logical file
LOW ⑨ De-register event – 1 input, 1 logical file
LOW ⑩ Give comments – 1 input, 1 logical file
MED ⑪ Generate notification message – 1 output
MED ⑫ Generate QR code – 1 output
LOW ⑬ Create/update/delete – 3 input, 3 logical files
LOW ⑭ Create/update/delete – 3 input, 3 logical files
LOW ⑮ View participant list – 1 inquiry
LOW ⑯ View registered users – 1 inquiry
LOW ⑰ View event comments – 1 inquiry
LOW ⑱ Delete user account – 1 input, 1 logical file

Characteristics	**Low Complexity**	**Medium Complexity**	**High Complexity**
Inputs	2, 4, 8, 9, 10 ,13 (x3), 14 (x3), 18	5	3
Outputs		11, 12	
Inquiries	6, 7, 15, 16, 17	1	
Internal Files	2, 4, 8, 9, 10, 13 (x3), 14 (x3), 18	5	3
External Interfaces		5	3

ii)

Characteristics	**Low Complexity**	**Medium Complexity**	**High Complexity**
# Inputs	12 x 3 = 36	1 x 4 = 4	1 x 6 = 6
# Outputs	0 x 4 = 0	2 x 5 = 10	0 x 7 = 0
# Inquiries	5 x 3 = 15	1 x 4 = 4	0 x 6 = 0
# Internal Files	12 x 7 = 84	1 x 10 = 10	1 x 15 = 15
# External Interfaces	0 x 5 = 0	1 x 7 = 7	1 x 10 = 10
	135	35	31

iii) **Unadjusted Function Points** = 135+35+31
= **201**

All except end-user efficiency and distributed data processing are low – 1.

Influence Factors = Total Score * 0.01 + 0.65
= (1+1+1+1+5+0) * 0.01 + 0.65
= 0.74

Adjusted Function Point = Unadjusted FP * Influence Factors
= 201 * 0.74
= **148.74**

b)

i) Effort = (3.0) * (1.19*1.4*1.06) * $8^{1.12}$ = 54.395
Duration = 2.5 * $(54.395)^{0.35}$ = 10.125
Team Size = 54.395 / 10.125 = 5.3725 ≈ **6 people**

ii) (10.125 – 9) / 10.125 = 11%
Increase effort by 11%
Effort = 54.395 * 1.11 = 60.379
Team Size = 60.379 / 9 = 6.7088 ≈ **7 people**
The compression is not less than 75% and it would only require additional of 1 more member. Thus, it is reasonable to compress the time needed by 11%.

3)

a)

i) CVS, Git, SVN

ii) SVN, Git
- Commit, send changes from your working copy to the repository.
- Merge, apply the differences between 2 sources to a working copy path.
- Add, add file contents to the index

iii) SVN is a centralized revision control system, and GIT is a distributed revision control system (DVCS).
The act of cloning an entire repository gives distributed version control tools several advantages over centralized systems: Performing actions other than pushing and pulling changesets is extremely fast because the tool only needs to access the hard drive, not a remote server. Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once. Everything but pushing and pulling can be done without an internet connection. So you can work on a plane, and you won't be forced to commit several bugfixes as one big changeset. Since each programmer has a fully copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone.
There are only two major inherent disadvantages to using a distributed system: If your project contains many large, binary files that cannot be easily compressed, the space needed to store all versions of these files can accumulate quickly. If your project has a very long history (50,000 changesets or more), downloading the entire history can take an impractical amount of time and disk space.

b)

i) The Capability Maturity Model (CMM). The model's aim is to improve existing software development processes, but it can also be applied to other processes.
A maturity model can be viewed as a set of structured levels that describe how well the behaviors, practices and processes of an organization can reliably and sustainably produce required outcomes. A maturity model can be used as a benchmark for comparison and as an aid to understanding – for example, for comparative assessment of different organizations where there is something in common that can be used as a basis for comparison. In the case of the CMM, for example, the basis for comparison would be the organizations' software development processes.

ii) From level 1 maturity levels, the processes are unpredictable, poorly controlled and reactive. To move to level 2, where processes are managed, the processes need to be characterized for project and is often reactive. Then to level 3, where processes are defined, at this level, the processes need to be characterized for the organization and is proactive. Moreover, for each level, we must ensure that a particular set of key process areas are stable. To be considered as level 2, we need to ensure the following key process areas: Software configuration management, software quality assurance, software project tracking & oversight, software subcontract management, software project planning, requirements management. Whereas for level 3, we need to ensure the following on top of the key process areas in level 2: peer reviews, intergroup coordination, software product engineering, integrated software management, training program, organization process definition, organization process focus.

4)

a)

i) Regression testing is a type of software testing which verifies that software which was previously developed and tested still performs the same way after it was changed or interfaced with other software.
For example, the 2 pieces of function code below returns x*y result when both positive numbers. But if y is negative the result for the 2nd function would be different which is not intended. Thus, using regression testing, we can make use of the same test to identify any bugs/defects after a change.

```
XTimesY (x,y) {
   return x*y;
)
XTimesY (x,y) {
   int result = 0;
   for loop from 0 to y-1 {
         result = result + x;
   }
   return result;
}
```

ii) Integration testing is to test the integrations of many units together to make sure that components works when put together, including dependencies, databases and libraries. After all the integration is completed, previously used test is re-run. This is regression testing to ensure that further changes have not broken any units that were already tested. You can run your unit tests again and again for regression testing.

b) To support maintenance of legacy system. Firstly, we need to analyze and predict its maintainability, maintenance costs and changes needed. All these are important for making decision on maintaining a legacy system. Prediction can be done by analyzing the documentations, measurable metrics such as defect rate or how often changes are made.

When dealing with legacy system, we also need to understand the business value they have.

When it requires changes there are different ways, we could work around it.

1. If the system has high business value but low quality, we can choose to re-engineer the software or replace it with a new one.
2. If the system has high business value and high quality, we can just maintain the software as it is.
3. If the system has low business value and low quality, it is preferred to scrap the software and modify the processes.
4. If the system has low business value but high quality, we can choose to scrap like number 3 or maintain the system as it is like number 2.

--End of Answers--