Solver: Jesslyn Chew

1)

a) "Silver bullet syndrome" implies that excessive reliance on the advertised benefits of previously unused technology and too little information about how well they would do in this particular development environment. This results in teams using those technologies and expecting to solve their schedule problems, only to be disappointed. One example would be a team may expect that using Django would drastically reduce the time required to complete the project due to its advertised benefits from others and its popularity. However, the team did not account for the time required to learn it and overestimated its usefulness and simplicity. This resulted in the predicted schedule to be inaccurate as it is too short.

b) The 2 internal attributes for managing portability requirement are the number of installation specific library routines used and the number of machine dependent calls used.
Portability is the capability of the software product to be transferred from one environment to another. The environment may include organisation, hardware or software environment. To ensure good portability, we must make sure that the software able to transfer to a different environment. This external attribute is affected by how dependent the software is to its current environment. The more reliant it is to the current hardware and software of the machine used, the less portable it is.
During project development, we must ensure that the 2 internal attributes are minimised. Constantly check that the installation specific library routines and the machine dependent calls are only used when really necessary to reduce the dependency on the software and hardware environment. By doing so, we can ensure good portability in the project.

c) In formal technical review meetings, the issues in the project are consolidated. However, the review is only complete when the issues found are corrected and verified. For example, in a formal technical review meeting, the team has found a issue with the login feature and plan who will fix that issue. At the end of the meeting, the review is not finished. It is only finished once the assigned team has fixed the login feature and the verification for the product quality is done.

d) Project development lifecycle is a portion within the project (management) lifecycle. The development lifecycle consists of the product or system code development. This phase is inside the "Execution" phase of the project (management) lifecycle. The project management lifecycle not only include the development, but the entire project from getting the contract or business case, forming the team, planning the schedule for the project development and completion.

2)

a)

i) Sentence 1: Staff → Log in to the system (1 Inquiry)
Sentence 2: Staff → update access permissions to financial data (1 Input, 1 Logical file)
Sentence 3: Calculate pay details based on data in the database, save pay details to database(1 Logical file), send an instruction to external system(1 External Interface), print pay slip(1 Output) – Due to the additional calculation, assume all characteristics in this

sentence to be of high complexity
Sentence 4: Update employee data based on input from external human resource system (1 Logical file, 1 external interface, 1 input)
Sentence 5: print monthly report containing the payment information of the month (1 output)
Assume that the characteristics are of medium complexity if not mentioned.

ii)

| Characteristics | Low Complexity | Medium Complexity | High Complexity |
|---|---|---|---|
| # Inputs | 0 x 3 = 0 | 2 x 4 = 8 | 0 x 6 = 0 |
| # Outputs | 0 x 4 = 0 | 1 x 5 = 5 | 1 x 7 = 7 |
| # Inquires | 0 x 3 = 0 | 1 x 4 = 4 | 0 x 6 = 0 |
| # Internal Files | 0 x 7 = 0 | 2 x 10 = 20 | 1 x 15 = 15 |
| # External Interfaces | 0 x 5 = 0 | 1 x 7 = 7 | 1 x 10 = 10 |

Total Unadjusted Functional points = 8 + 5 + 4 + 20 + 7 + 7 + 15 + 10 = 76

b) Function points = 500
1 Function points → 29 lines of code for C++
System has medium complexity but development team has no experience on such system development
System has very tight hardware, software and operation constraints
Find the estimated size of the team

Based on the development team experience and the tight hardware, software and operation constraints, the software project mode is embedded.
KLOC = 500 * 29 / 1000 = 14.5
Effort = a (KLOC)b = 3.6 (14.5)1.20 = 89.114
Duration = c (Effort)d = 2.5 (89.114)0.32 = 10.518
Team size = Effort / Duration = 89.114 / 10.518 = 8.47 $\approx$ 9

c) Budget = $3,500
Critical path: ABCFI

| Activity | Crash cost per week | Crash weeks available |
|---|---|---|
| **C** | **$400** | **2** |
| D | $1200 | 1 |
| E | $800 | 1 |
| **F** | **$500** | **2** |
| G | $500 | 1 |
| H | $900 | 1 |
| **I** | **$700** | **1** |

Activity C [Crash weeks available = 2] has the crash cost per week of $400 that is in the critical path.

- No Change in critical path
- Budget left = $3500 - $400 = $3100

Activity C [Crash weeks available = 1] has the crash cost per week of $400 that is in the critical path.

- Critical path: ABEFI and ABCFI
- Budget left = $3100 - $400 = $2700
- Crash weeks for activity C is now zero

| Activity | Crash cost per week | Crash weeks available |
|---|---|---|
| **C** | **$400** | **0** |
| D | $1200 | 1 |
| **E** | **$800** | **1** |
| **F** | **$500** | **1** |
| G | $500 | 1 |
| **H** | **$900** | **1** |
| **I** | **$700** | **0** |

Activity F [Crash weeks available = 2] has the crash cost per week of $500 that is in the critical path.

- Critical path: ABEFI, ABCFI, ABEHI and ABCHI
- Budget left = $2700 - $500 = $2200

To gain 1 week, either cut E and C (cannot) / cut F and H ($1400) / cut I ($700)
The cheapest is to cut I

- Critical path: ABEFI, ABCFI, ABEHI and ABCHI
- Budget left = $2200 - $700 = $1500
- Crash week for I is now 0

To gain 1 week, cut F and H ($1400)

- Critical path: ABEFI, ABCFI, ABEHI and ABCHI
- Budget = $1500 - $1400 = $100

Conclude steps:

- Cut C
- Cut C
- Cut F
- Cut I
- Cut F and H

Total weeks reduced is 5 weeks.

3)

a)

i) From the description, it is mentioned that one of the existing software tools that has a much-simplified prediction function being used is outdated and should be replaced with an advanced form of a market prediction software. This is a form of adaptive maintenance, under the enhancement category.

ii) We can use the defect rate and mean time to change as a metric to decide how to evolve. The defect rate tells us the ratio of mistakes based on the size of the software, while the mean time to change is an indirect measurement of maintainability as it tells us the amount of time for a change on the software is required. These metrics are measurements on the quality of the product. If the quality of the product is at an undesirable level, then there is a need for the maintenance.

iii) Based on the investigation, there can be 2 conclusions. The first conclusion is that there is a need for a change. The second conclusion is that the change is unnecessary.
If there is a need for a change, the following are the possible means of evolving:
Replace the system with a newer system
Re-engineering is not possible as there is a large change in functionality from the much-simplified prediction function to the advanced form. It is also assumed that it is impossible to scrap the system completely as the prediction function is an important functionality in a consultant company.
If there is no need for the change, the maintenance team would then have to continue maintaining the system.

b)

i) Git.

ii) Function 1 – Commit: This function commits changes in the working directory. Note that "git add" must be used first before this function.
Function 2 – Pull: This function allows the user to fetch and merge the repository from the remote server to his/her working directory.
Function 3 – Log: The team may have made multiple of commits and changes to the repository. This function allows the user to view the version history of the current branch.
Function 4 – Merge: In a repository, there may be multiple branches so that different people can focus on different parts of the project without worrying about overriding each other's work or the merging process until their portion is complete. This function enables the user to merge a specified branch into the current branch

iii) Function 1 – git commit -m "[Commit message]"
Function 2 – git pull [Repository link]
Function 3 – git log
Function 4 – git merge [branch name]

4)

a)

i) Capability Maturity Model Integration (CMMI)

ii) There are 2 main benefits for CMMI which are categorised as improvement and appraisal. <Refer to lecture on CMMI, page 17>

iii) In order to improve the maturity from level 1 to 2, the following key process areas need to be involved: Software configuration management, software quality assurance, software

project tracking & oversight, software subcontract management, software project planning, requirements management. In our lab project, only requirements management, project planning, software quality assurance and configuration management are included. Hence, the software processes should include software project tracking & oversight, software subcontract management.

b)

i) First, we have to identify an interface that you will need to test regression testing on. Regression testing is required when specific changes are made. In this example, we will take a bug fix as an example.

```
== OLD CODE ==
public void class myClass {
    public void divide (float x, float y) {
            return (x/y);
    }
}
```

Note that a bug was found in this case as y = 0 will result in a division error. Hence, the code is modified to fix the bug found. We will force the function to return 0 to avoid the division error.

```
== NEW CODE ==
public void class myClass {
    public void divide (float x, float y) {
            if (y == 0)
                    return 0;
            return (x/y);
    }
}
```

ii)
```
myClass c = new myClass();
// test 1
float result1 = c.divide(5,2);
if (result1 == 2.5)
    System.out.println(“Pass test 1.”);
else
    System.out.println(“Fail test case, test 1 output “ + result1 + “ instead of 2.5”);

// test 2
float result1 = c.divide(-5,2);
if (result1 == -2.5)
    System.out.println(“Pass test 2.”);
else
    System.out.println(“Fail test case, test 2 output “ + result1 + “ instead of -2.5”);
```

```
// test 3
float result1 = c.divide(5,0);
if (result1 == 0)
    System.out.println("Pass test 3.");
else
    System.out.println("Fail test case, test 3 output " + result1 + " instead of 0");

// test 4
float result1 = c.divide(0,2);
if (result1 == 0)
    System.out.println("Pass test 4.");
else
    System.out.println("Fail test case, test 4 output " + result1 + " instead of 0");
```

iii) The bug can be detected in 2 ways. First, it will detect if the code can compile and run. If the code fails to run, the integrated development environment will display a message. If the code can be executed, we will compare the test results with the actual results. This will check the correctness of the result. If the test result is different from the expected behaviour, a bug is found.

--End of Answers--