Solver: Zhou Jingyuan

1)

a) Either the individual capabilities of the team members or their relationship as a team probably has the greatest influence on productivity (Boehm 1981, Lakhanpal 1993). Hiring from the bottom of the barrel will threaten a rapid development effort. In the case study, personnel selections were made with an eye toward who could be hired fastest instead of who would get the most work done over the life of the project. That practice gets the project off to a quick start but doesn't set it up for rapid completion.
① **Lack of capabilities**: when developing XXX, we have one team member who did not have enough coding background and does not finished his/her parts and hence delay the project.
② **Bad relationship:** when developing XXX, we have one team member who did not get along well with other members and slack off. As a result, the project was delayed.

b) A software property can be measured.
The relationship exists between what we can measure and what we want to know. We can only measure internal attributes but are often more interested in external software attributes.
A quality metric should be a predictor of product quality.

c) ① A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.
In our group project, we want Usability & Maintainability, so we define the number of error message and program size as our most significant quality attributes.
② The quality plan should define the quality assessment process.
We defined that the QA engineer will conduct code review and assessment for every five commits and for each version.
③ It should set out which organizational standards should be applied and, where necessary, define new standards to be used.
Instead of coming up with our own standards, we decided to follow strictly with the ISO/IEC 9126-1 Quality Model by the International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC).

d) There is no standard answer, below is one example.
Pros:
Many brains are better than one. Coders will often spot mistakes or question oddities in each other's code, thus improving the overall code robustness. Think of it as continuous peer code review.
Lowers risk of losing critical/valuable knowledge if a developer is hit by a bus or resigns. If a developer is on holiday for a week you don't have to wait for a week for a bug to be fixed, Or you don't have to explain to an angry coder why you messed with "his" code while his back was turned.
Cons:
Fights and religious wars over minor differences of opinion over coding style or implementation approaches

"Too many cooks spoil the broth" - sometimes the best way to keep a design clean and focused is to have only one person mastermind it. This doesn't necessarily mean that they do all the work though, as long as all the team members understand the vision and follow the architect's design. Code reviews can be very powerful here.

Not all programmers are created equal. It can be frustrating at times when your team is able to deliver good clean code, but perhaps one person isn't able to reach the quality the rest of the team delivers. This can breed discontent and divide a team as they can feel that their wonderful creation is being corrupted.

2)

a)

i)

	Low	**Medium**	**High**
Inputs	1	3, 4	
Outputs	7		6, 8
Inquiries	2, 5		
Logical Files	1	3	
Interfaces	2, 6		

Notes (for your understanding only, not necessary in exam):

Detailed definition adopted from: https://goo.gl/o6suwQ

1: Count as logical files as system need to store user information (debatable)

2: The system we are developing is a sub-system, hence the Clinic System is regarded as an external system. It is an inquiry as we assume display does not require additional processing.

3: Count as logical files as system need to store the plan.

4: Same plan as 3, only count once

5: Count as inquiries because it does not contain addition process before sending the data out. i.e. the system does not need to derive/compute any additional information before sending the data. Progress should be an attribute can directly obtain from the system.

6: As the word generate suggest, the existing data cannot be directly used, but need to be processed into certain format and/or compress, hence an output.

7 & 8: Reports need additional formatting on top of the data before send out. It is an output since additional processing/formatting need to be done.

ii)

Characteristics	**Low Complexity**	**Medium Complexity**	**High Complexity**
# Inputs	1 x 3 = 3	2 x 4 = 8	0 x 6 = 0
# Outputs	1 x 4 = 4	0 x 5 = 0	2 x 7 = 14
# Inquiries	2 x 3 = 6	0 x 4 = 0	0 x 6 = 0
# Internal Files	1 x 7 = 7	1 x 10 = 10	0 x 15 = 0
# External Interfaces	2 x 5 = 10	0 x 7 = 0	0 x 10 = 0
	30	18	14

Unadjusted Function Points = 30 + 18 + 14
= **62**

Influence Factors = Total Score * 0.01 + 0.65
= (3+2) * 0.01 + 0.65

= 0.7

Although system complexity is not one influence factors in the sheet we get, but note that it is during the exam, and we don't have that sheet available. Thus, we can only assume that there is one influence factor called system complexity.

Adjusted Function Point = Unadjusted FP * Influence Factors
= 62 * 0.7
= **43.4**

b)

i) Initiation, planning, execution, completion

ii)

Phase	**Activities**
Initiation	Assign project manager Define acceptance
Planning	Approve project plan Obtaining tools
Execution	Track work against project plan Review progress
Completion	Obtain customer acceptance Final project review

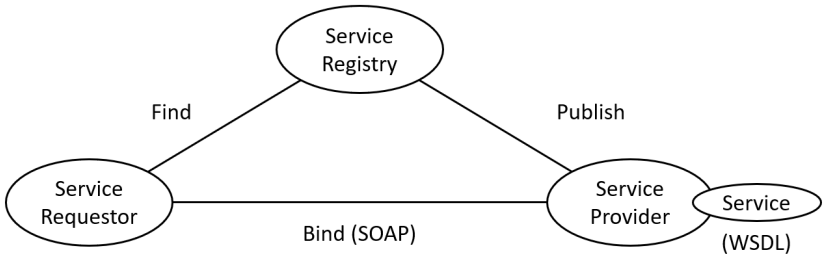
3)

a) Process-based quality management is to uses procedures to address the quality characteristics of the selected quality model (e.g. ISO 9126) at the various stages of the SDLC. CMMI defined the characteristics of good/effective Software Engineering Processes, i.e. good/effective methods to do software engineering, which is a model (management) to assess maturity and improve quality of processes.

b)

i) SOA main elements includes: 1) Well-defined interfaces 2) Loose coupling 3) Logical and physical separation of business logic from presentation logic 4) Highly reusable services 5) Coarse-grained granularity

One example of SOA will be Amazon Web Service (AWS). AWS (Service registry) manages a large number of physical machine (Service Provider). When I would like to rent a server, I (service requestor) send this request to AWS. AWS will provide me with one or more physical servers or a shared hardware for me. Thus, I will have my own server (Service)



ii) I will try to answer it as general as possible by choosing MDP as example:
In MDP, we need to build a robot that can explore a maze. We have 4 parties, robot hardware, pc, tablet and RPi. RPi acts as a service registry which published and available for invocation for different services.

Pseudo-code in Python
Robot hardware has the following interface:

- Move (step)
- Rotate (degree)

which is registered in the RPi.
To display the robot location in tablet, the tablet also implemented robot interface

- Move (step)
- Rotate (degree)

When PC want to move the robot forward, the PC send message to RPi to invoke move(1) method. RPi will broadcast this message to all devices that published this interface.
For robot in registeredRobot:
robot.move(1)
This implementation enables loose coupling between devices.

c) 1: Maintenance is inevitable: A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment. To maintain the value of these assets to the business, they must be changed and updated.
2: The number of changes and maintenance efforts needed are expected to increase as the system grows.
3: We might need experts who are the original developer or have insight into the system: where and how the changes should be made.
Defending the idea of maintaining: assume it is a relatively new software, the cost of maintaining the software outdone the cost of develop a new one.

4)

a) Take git as an example
git merge -m "Merge work in mybranch" mybranch
can merge two variation
git repack the public repository. This establishes a big pack that contains the initial set of objects as the baseline, and possibly **git prune** if the transport used for pulling from your repository supports packed repositories.
git checkout mybranch can checkout to let it be a local version so any changes you made on your local won't mess with the remote version unless you choose to commit and push

b) Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts
Model - It is the lowest level of the pattern responsible for maintaining data.
View - It is responsible for displaying all or a portion of the data to the user.
Controller - It is a software Code that controls the interactions between the Model and View.
Almost all web framework used MVC. [Take one of the most popular one as example, but you could choose whatever you familiar to explain.]
AngularJS is a JavaScript framework that implements MVC architecture. it isolates the application logic from the user interface layer (HTML) and supports separation of concerns. The controller handles all requests for the application and then works with the model to prepare any

data needed by the HTML. Practical issue: the binding of the model data and the view is done by dirty checking which need manual syncing if modification to the data is done outside the framework.

c) TDD is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only. This is opposed to software development that allows software to be added that is not proven to meet requirements.
Say a project to write a calculator, I will start by writing tests for testing addition. Then implement the addition functionality. Thereafter writing tests for subtraction, so on and so forth.
Benefit includes
1. Maintainable, Flexible, Easily Extensible
2. Unparalleled Test Coverage & Streamlined Codebase
3. Clean Interface
4. Refactoring Encourages improvements.

However, the drawbacks cannot be neglect:
1. It is hard to ascertain the range of the test: high level tests (integrate tests)? Or low-level test (unit test)? If high level tests, you cannot pinpoint the problem. If low level, it costs you twice the time to maintain your code once requirement changes (very common).
2. Focus on test rather design. TDD encourages runnable code but good code.
3. Extensive use of Mock & Stub. When interact with external system, you need many mocks to simulate external function and interaction, making the tests difficult to write. On top of that, you need to make many assumptions and maintain those mock. Sometimes your mock could even be wrong.
4. Time consuming. Some features are not easy to test, for instance parallel processing. In addition, before you write tests, you need to understanding requirements. If you failed to do that, all the tests and code go in vain. If not TDD, you only need to update the code

--End of Answers--