Solver: Loo Zing Zai

1)

a) "Politics placed over substance" mistake is putting politics over results that is fatal to speed-oriented development. The 4 cases are:
Politicians: They specialized in "managing up", concentrating on relationships with their managers.
Researchers: They concentrate on scouting out and gathering information.
Isolationists: They keep to themselves, creating project boundaries that they keep closed to non-team members.
Generalists: They have a little of everything, they tend their relationships with their managers, perform research and scouting activities and coordinate with other teams.

b)

i) The waterfall lifecycle can be divided into 5 stages. They are:

Requirements Elicitation & Analysis: Understand what customer really need. Develop project plans, quality plans and other written documentations. In the long run, when employees leave the organization, new employees can refer to these plans to understand and continue to maintain the existing system. UML diagram, ER diagram and other high-level representations should also be documented.

Design: Develop software architectural design. Make use of design patterns and architectural patterns where applicable. Design and architectural patterns are abstract and do not contain code, so they are considered under "Design" rather than "Implementation". Keep in mind good design principles such as, loose coupling, high cohesion, modularity, reusability. By doing so, modules should be well-defined, conceptually simple and independent.

Implementation: Comment your code. It is the simplest way to take maintainability into consideration during implementation. When you comment your code, others maintaining the system/program will be able to understand.
Continuous integration using a Version Control Software can also be used where members integrate their work frequently. While this software development practice benefits when members are developing, it also helps during maintenance because every push require comments. We can trace back which improves maintainability.

Testing: Create and define test cases and test results in detail.

Maintenance: You can refactor your code to make it easier to understand. Can also improve the system/program structure, modularize components further and re-engineer data where possible.

Can also re-engineering or re-writing part of or all of a system without changing its

functionality with the goal of making systems/programs easier to maintain. More applicable to legacy systems.

ii) 1. **Number of procedure parameters:** The higher the number of parameters the more difficult it is to understand the procedure. It makes the procedure less modular and independent. Difficult to maintain when procedures are tightly coupled.
2. **Cyclomatic Complexity**: This is a measure of the control complexity of a program. This control complexity may be related to program understandability. A higher CC usually means that it is more difficult to maintain as the program is more complex and harder to understand.
3. **Programme size in lines of code**: The larger the size of a programme, the more difficult it is to trace through to resolve bugs and errors.

c) Quality culture is a system of shared core values and beliefs within an organization where quality is seen as everyone's responsibility. It resolves around quality management that guarantees a certain degree of quality being achieved in software products developed by the organization.
To have a better view of quality culture, we can compare the quality culture of a startup company and a pioneer organization in the software development industry for many years. A startup company has the freedom to develop applications. On the other hand, the organization will have defined and standardized procedure to follow. Defined and standardized procedures mean that procedures are process dependent instead of personnel dependent and assures a certain level of quality will always be achieved.

2)

a)

i)

	Low	**Medium**	**High**
Inputs		4 (x2)	
Outputs	5	6	7
Inquiries	1, 2, 3		
Logical Files		4 (x2)	
Interfaces	2, 3, 8 (x2)		

[For your information only, no need to present]
① [Low] login – 1 inquiry
② [Low] retrieve and display – 1 inquiry, from Hospital Patient System – 1 interface
③ [Low] Get list – 1 inquiry, from Recommendation Analysis System – 1 interface
④ [Medium] Create & update – 2 input, 2 logical files
⑤ [Low] Print records – 1 output
⑥ [Medium] Print report – 1 output
⑦ [High] Print report – 1 output, using a Data Analytics Software installed in the system – no interface as it is part of the system
⑧ [Low] 2 separate systems retrieve from this system – 2 interfaces

ii)

Characteristics	Low Complexity	Medium Complexity	High Complexity
# Inputs	0 x 3 = 0	2 x 4 = 8	0 x 6 = 0
# Outputs	1 x 4 = 4	1 x 5 = 5	1 x 7 = 7
# Inquiries	3 x 3 = 9	0 x 4 = 0	0 x 6 = 0
# Internal Files	0 x 7 = 0	2 x 10 = 20	0 x 15 = 0
# External Interfaces	4 x 5 = 20	0 x 7 = 0	0 x 10 = 0
	33	33	7

Unadjusted Function Points = 33 + 33 + 7
= **73**

iii) Complex processing [medium]: 3
Data communication [little]: 1
Transaction rate [little]: 1
Operation ease [essential]: 5
Influence Factors = Total Score * 0.01 + 0.65
= (5+1+1+3) * 0.01 + 0.65 *Essential = 5, Little = 1*
= 0.75
Adjusted Function Point = Unadjusted FP * Influence Factors
= 73 * 0.75
= **54.75**

b) COCOMO model takes into account team experiences and external drivers into the estimation to achieve a more accurate project estimation. For example, COCOMO Basic Model categories projects into 3 development modes, Organic, Semi-detached and Embedded. COCOMO Model II includes Effort Adjustment Factors derived from cost drivers and an Exponent value derived from the 5 scale drivers.

c) Agile software development encourages rapid response to change requests, early product delivery and continuous improvement. Code refactoring allows code to be understood, maintained and evolved easily. Defects can be identified and resolved immediately using refactoring during the continuous integration process. It lowers cyclomatic complexity and increases performance of the software.

3)

a) Github.
① **git clone "URL"**: Clones an existing Git repository.
② **git add .**: Add files in the local repository and prepares them for commit.
③ **git commit -m "COMMENT"**: Commits the tracked changes, includes deletion of files and prepares them to be pushed to a remote repository.
④ **git push origin master**: Pushes the changes in your local repository up to the remote repository you specified as the origin.

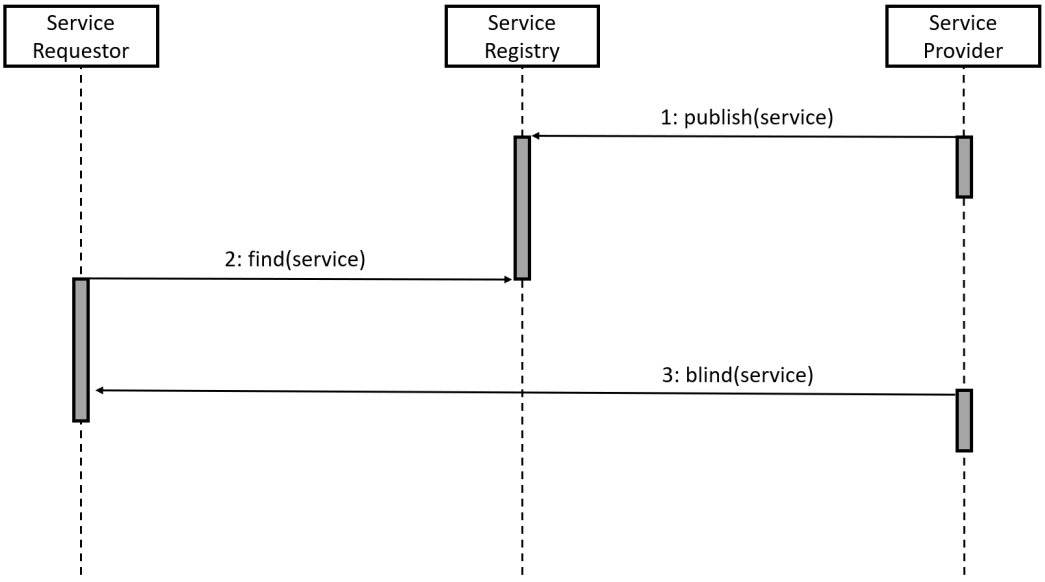
b)

i) Capability Maturity Model Integration (CMMI). Some benefits of the model are:
① Estimated VS Actual costs and schedule shown to be very accurate at CMMI Level 3.

② Productivity improves from 100% to 200%.
③ Post release defects reduced from 10% to 94%.
④ Work carried out according to a planned process.
⑤ Clearly defined roles and responsibilities.
⑥ Quantitative basis for judging quality, analyzing problems and making decisions.

ii) To demonstrate the maturity of the team, we can describe some areas. They are:
① Commitment to perform: Policies established (You can be more specific).
② Ability to perform: Training provided to all team members.
③ Activities performed: Reasonable plans are developed based on realistic estimates. Began with Software Project Proposal, Software Project Plan, Software Quality Plan, (include other plans you did). We have experience in software project tracking.
④ Measurement and analysis: Measurements taken to determine the status and effectiveness of the activities performed feature.
⑤ Verifying implementation: Reviews and audits by management and software quality assurance team.

c) *Not sure about this part. The description for the implementation not necessary for question.*
Infrastructure pattern implementation: **Amazon Web Services**, Amazon had every team decouple, define what resources they had and make them available through an API. Different teams developed their respective web services to allow others to gain access to their data. Different teams had to then interact using APIS, also known as web services. You can find more information from:
http://apievangelist.com/2012/01/12/the-secret-to-amazons-success-internal-apis/



4)

a) The 3 categories of mid-level design patterns are Broker design pattern, Generator design pattern and Reactor design pattern.

Broker design pattern can be used when interfaces are not compatible with each other. It coordinates communications between components. **Generator** design pattern can be used to create objects with ease. **Reactor** design pattern is used when components need to perform requests concurrently.

Advantages

Broker design pattern is to allow components to conform and interact with incompatible components. It is extremely useful in legacy systems which are difficult to maintain and have to interact with newer components & APIs.

Generator design pattern saves time and effort in creating and/or cloning objects. For complex objects that require many calls, generator design pattern can help us to clone new objects with ease.

Reactor design pattern separates components into modular, reusable parts. Take the Observer pattern as an example, we can keep adding concrete observer components to the Observer component and keep adding concrete observable components to the Observable component. It makes attaching and detaching components easy as they are independent and reusable.

b) The two categories are **Broker** and **Generator** design pattern.

Broker: **Adapter, Façade**	Generator: **Factory, Singleton**
Adapter: It can be used with a system that has an incompatible interface. The adapter wraps around the system and conforms to the required interface. The system can be reused with a new interface.	**Factory:** Creates multiple instances for client. For example, observers can be added using configuration files.
Façade: Defines an object that encapsulate how a set of objects interact.	**Singleton:** Creates 1 instances throughout the system.

c) The two testing techniques I chose were: **Acceptance testing** and **Alpha & Beta testing**

① **Acceptance testing:** Includes usability, functional and performance tests. I would focus on usability and human factors testing (it is under performance testing) where we are able to analyze how users behave to our current GUI. For example, what buttons they will press/whether text and widgets are too small for them to see. By performing these tests, we will be able to improve on the GUI

② **Alpha & Beta testing:** Alpha and beta testing are performed in different sites and environments. Alpha testing is performed at development site while beta testing is performed at a site similar/equal to production. This can be coupled with usability and human factors testing to see how users behave in the different environments.

--End of Answers--