Solver: Rain Chua Qin Lei

1)

a) Wishful thinking. "They thought that maybe if everyone worked hard, and nothing went wrong, and they got a few luck breaks, they just might be able to pull it off". The team is hoping that nothing will go wrong and they might be able to complete the project. Life is not a bed of roses. Wake up! It is highly unlikely that everything will go smoothly during the development of the project. [given in the reference notes]

b) Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the heath, safety and welfare of the public, software engineers shall adhere to the following principles.
PUBLIC. Software engineers shall act consistently with public interest. Software engineers shall accept full responsibility for their own work and approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy, or harm the environment.
PROFESSION. Software engineers shall advance the integrity and reputation if the profession consistent with public interest. Software engineers shall help develop an organizational environment favorable to acting ethically and promote public knowledge of software engineering.

c) What type of system is being developed?
- Plan-driven approaches may be required for system that require a lot of analysis before implementation (e.g. real-time system with complex timing requirements)

What is the expected system lifetime?
- Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team.

What technologies are available to support system development?
- Agile methods rely on good tools to keep track of an evolving design.

How is the development team organized?
- If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.

Are there cultural or organizational issues that may affect the system development?
- Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.

How good are the designers and programmers in the development team?
- It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code.

Is the system subject to external regulation?
- If a system has to be approved by an external regulator (e.g. the FAA approve software that is critical to the operation of an aircraft) then you will probably be required to produce

detailed documentation as part of the system safety case.

d)

i) Reliability is the capability of the software product to maintain its level of performance under stated conditions for a stated period of time. Consideration of reliability include maturity, fault tolerance level and recoverability. For example, have most of the faults in the software been eliminated over time and is the software capable of handling errors. Maturity of system include frequency of system failure and presence of faults/software bugs. Fault tolerance capability of the system include the ability to maintain a specified level of performance or continue functioning in the event of software fault(s) and the system's response to invalid input data.

ii) Internal attributes to measure reliability include cyclomatic complexity, programme size in line of code and number of error messages. Cyclomatic complexity is a software metric used to indicate the complexity of a program. Cyclomatic complexity can be computed using control flow graph of the program. McCabe number (cyclomatic complexity) with a threshold of below 10 are considered reliable. Programme size is thought to be reflective of complexity, development effort and reliability. LOC or KLOC is an intuitive initial approach to measuring software size. The higher the number of error messages, the more reliable a software will be as the more errors are more likely to be contained.

2)

a)

i) ① Display + retrieve – 1 inquiry, external system – 1 interface
② Create, update, delete – 3 inputs, 3 logical files
③ Input record data – 1 input, 1 logical file
④ View details – 1 inquiry
⑤ Give and save feedback (based on patient's current status) – 1 inquiry, 1 logical file
[Vetter's note: Not too sure for ⑤, it is either inquiry or input. In this case, the solver assumes that the patient's current status is retrieved from the system]
⑥ Get statistical report – 1 output
⑦ Get summary report – 1 output

Characteristics	**Low**	**Medium**	**High**
Inputs		2 (x3), 3	
Outputs			6, 7
Inquiries	1, 4	5	
Logical Files		2 (x3), 3, 5	
Interfaces		1	

ii)

Characteristics	**Low Complexity**	**Medium Complexity**	**High Complexity**
# Inputs	$0 \times 3 = 0$	$4 \times 4 = 16$	$0 \times 6 = 0$
# Outputs	$0 \times 4 = 0$	$0 \times 5 = 0$	$2 \times 7 = 14$
# Inquiries	$2 \times 3 = 6$	$1 \times 4 = 4$	$0 \times 6 = 0$
# Internal Files	$0 \times 7 = 0$	$5 \times 10 = 50$	$0 \times 15 = 0$
# External Interfaces	$0 \times 5 = 0$	$1 \times 7 = 7$	$0 \times 10 = 0$

TOTAL:	6	77	14

Unadjusted Function Points = 6 + 77 + 14
= **97**

iii) Data communication [little]: 1
End-user efficiency [average]: 3
On-line update [frequent]: 4
Operation [easy]: 4
Influence Factors = Total Score * 0.01 + 0.65
= (1+3+4+4) * 0.01 + 0.65
= 0.77
Adjusted Function Point = Unadjusted FP * Influence Factors
= 97 * 0.77
= **74.69**

iv) $E = 2.94 * EAF * (KLOC)^{1.0997}$
$E = 2.94 * (1.46 * 1.13 * 0.86) * (74.69 * 53)^{1.0997} / 1000$
E = **37.713** PM
* Do note that it is KLOC and not LOC

b) Initial critical path: ABDCFJ

Activity to be cut	Crash cost per week	Critical path	# of weeks reduced
B	$ 400	ABDCFJ, AGDCFJ	1
F	$ 500	ABDCFJ, AGDCFJ, ABDCIJ, AGDCIJ	2
C	$ 1 200	ABDCFJ, AGDCFJ, ABDCIJ, AGDCIJ	3
C	$ 1 200	ABDCFJ, AGDCFJ, ABDCIJ, AGDCIJ	4

3)

a)

Well-defined interfaces	A well specified description of the service that permits consumers to invoke it in a standard way
Loose coupling	Service consumer is independent of the implementation specifics of service provider
Logical and physical separation of business logic from presentation logic	Service functionality is independent of user interface aspects
Highly reusable services	Services are designed in such a way that they are consumable by multiple applications
Coarse-grained granularity	Services are business-centric, i.e. reflect a meaningful business service not implementation internals

[Choose 2 is enough]

b)

i) Level 5 Optimizing: Focus on process improvement
Level 4 Quantitatively Managed: Processes measured and controlled
Level 3 Defined: Processes characterized for the organization and is proactive (Projects tailor

their processes from organization's standards)
Level 2 Managed: Processes characterized for projects and is often reactive
Level 1 Initial: Processes unpredictable, poorly controlled and reactive

ii) **CMMI**

LEVEL	CHARACTERISTIC	KEY PROCESS AREA	RESULTS
5 Optimizing	Improvement fed back into process	Process change management Technology innovation Defect prevention	Productivity and quality
4 Managed	(Quantitative) Measured process	Quality management Process measurement & analysis	
3 Defined	(Qualitative) Process defined and institutionalized	Peer reviews Intergroup coordination Software product engineering Integrated software management Training program Organization process definition Organization process focus	
2 Repeatable	(Intuitive) Process dependent on individuals	Software configuration management Software quality assurance Software project tracking & oversight Software subcontract management Software project planning Requirements management	
1 Initial	(Ad hoc/chaotic)	Survival	Risk

iii) **Wiring-by-Configuration** [From lecture notes]

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
   <!DOCTYPE struts-config PUBLIC


   "-//Apache Software Foundation//DTD Struts Configuration
1.3//EN"
   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
 <struts-config>
   <form-beans>
      <form-bean name="logonForm" type="app.LogonForm"/>
   </form-beans>
   <action-mappings>
      <action path="/Welcome" forward="/pages/Welcome.jsp"/>
      <action path="/Logon" forward-"/pages/Logon.jsp"/>
      <action path="/LogonSubmit" type="app.LogonAction"
name="IogonForm"
            scope="request" validate="true"
input="/pages/Logon.jsp">
            <forward name="success"
```

```
path="/pages/Welcome.jsp"/>
                    <forward name="failure" path="/pages/Logon.jsp"/>
                </action>
                <action path="/Logoff" type="app.LogoffAction">
                    <forward name="success"
path="/pages/Logoff.jsp"/>
                </action>
            </action-mappings>
            <message-resources parameter="resources.application"/>
      </struts-config>
```

4)

a)

i) Git.

git clone /path/to/repository

- Create a working copy of a local repository by running the command

git add <filename>

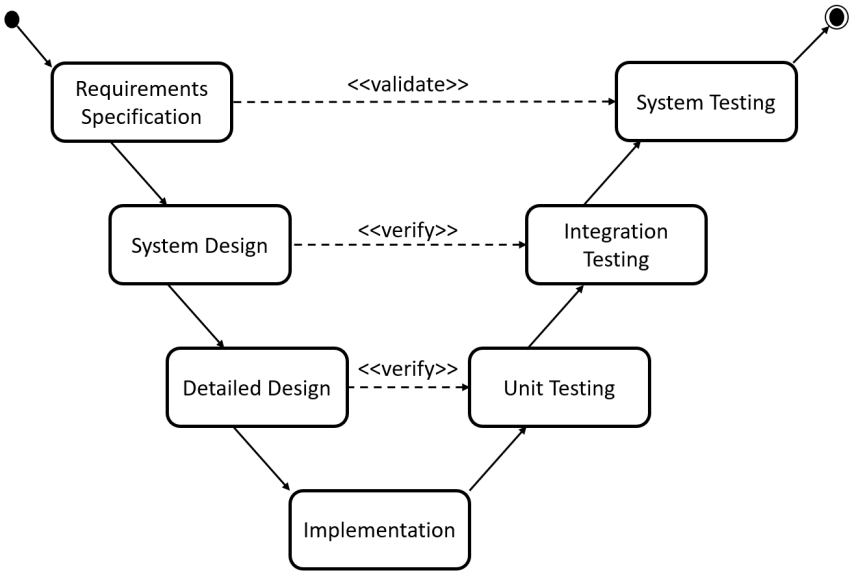
git add *

- These commands update the index using the current content found in the working tree, to prepare the content staged for the next commit.

ii) Refer to http://www.inanzzz.com/index.php/post/cs32/working-with-git-release-branches for Git implementation for the release.

(1) Create release branch
(2) Check out release branch
(3) Build and test release branch
(4) Create release distribution file
(5) Test distribution file contents
(6) Tag release
(7) Hand-off distribution file to QA

b) **Test Targets and Test Levels**



c) **Corrective**: Diagnosing and fixing errors, possibly ones found by users.
Preventive: Increasing software maintainability, or reliability to prevent problems in the future.
Adaptive: Modifying the system to cope with changes in the software environment.
Perfective: Implementing new or changed user requirements which concern functional enhancement to the software.
Perfective maintenance take the highest percentage of effort as shown in the graph in lecture slides. [65% - Functionality addition or modification, 17% - Faulty repairs, 18% - Software adaptation]

--End of Answers--