Question 1

Benefits of software re-use

- **Systems can be developed more quickly**
  - Reusing software can speed up system production because both development and validation time may be reduced
- **Effective use of specialists**
  - Application specialists can develop reusable software that encapsulates the work same work that was done previously
- **Increased dependability**
  - Reuse software which has been bee tried and tested in working systems should be more dependable than new software. Its design and implementation faults should have been found and fixed
- **Lower development costs**
  - Reusing means less software(lines of code) have to be written and that may result in hiring less developers
- **Standard compliance**
  - Reusable software can make standards easy to implement, software components like user interface can improve dependability

Problems associated with this approach

- **Creating, maintaining and using a component library**
  - Populating a reusable component library and ensuring the software developers can use this library can be expensive
- **Finding, understanding and adapting reusable components**
  - The cost of time spent in looking for software to reuse and assessing whether or not it meets the needs
- **Increased maintenance costs**
  - If the source code of a reused software system or component is not available, then maintenance costs may be higher because the reused elements of the system may become incompatible with changes made to the system
- **Lack of tool support**
  - Some software tools do not support development with reuse, it may be difficult or impossible to integrate these tools with a component library.
- **The cost of adapting and configuring the reusable software components**
  - Integrating reusable software with each other system might be challenging and as a result may take more time to implement

Question 2

Why it is important to make a distinction between developing the user requirements and developing system requirements in the requirement engineering process?

- **The user requirements**
  - Are intended to describe the system's functions and features from a user's perspective
  - It is important that users understands these requirement
  - They are statements which are expressed in natural language and diagrams
  - The people involved in the process must be able to understand the user's environment and application domain
- **The system requirements**
  - Are more detailed descriptions of the system's functions, services and operational constrains
  - The system requirement document should define exactly what is to be implemented
  - This may be part of the contract between the system buyer and the software developers
- There is a **fundamental difference** between the user and the system requirements as depicted above and that suggests they should be considered separately

Question 3

- Three (3) types of software maintenance
  - **Fault repair to fix bugs and vulnerabilities**
    - This consist of coding errors, design errors and requirements errors and they are sometimes called corrective maintenance
  - **Environmental adaptation to adapt the software to new platforms and environments**
    - This type of maintenance is required when some aspect of a system's environment, such as the hardware, the platform operating system or other support software changes.
  - **Functionality addition to add new features and to support new requirements**
    - This type of maintenance is necessary when system requirements change in response to organizational or business change. The scale of the changes required to the software is often much greater than for the other types of maintenance.
- It is sometimes difficult to distinguish between them
  - Because in practice, **there is no clear distinction between them**, when you adapt a system to a new environment, you may add functionality to take advantage of new environmental features
  - Different people give them **different names**, different people sometimes give them different names. "Corrective maintenance" is universally used to refer to

maintenance for fault repair. However, "adaptive maintenance" sometimes means adapting to a new environment and sometimes means adapting the software to new requirements. "Perfective maintenance" sometimes means perfecting the software by implementing new requirements; in other cases, it means maintaining the functionality of the system but improving its structure and its performance

Question 4

- Six (6) other problems and challenges that software engineering is likely to face in the 21st century?
    1. There is a huge need to reduce the carbon footprint across the globe and this means software engineers have to develop systems that are **energy efficient**
    2. Software engineers have to build systems that are for **multicultural use**
    3. Development of systems that can **adapt quickly** to new business needs
    4. Developing systems for **outsourced development**
    5. Development of systems that are **resistant to attack**
    6. Development of systems that can be adapted and **configured by end-users**

Question 5

- Why design conflict might arise when designing an architecture for which both availability and security requirements are the most important non-functional requirement?
    - By definition Availability means the architecture should be **designed to include redundant components** so that it is possible to replace and update components without stopping the system, while Security refers to **a layered structure** for the architecture should be used, with the most critical assets protected in the innermost layers and a high level of security validation applied to these layers
    - While designing an the architecture of a system, conflict may arise because
        - In security requirement, the architecture must be layered structure
        - In availability requirements, the architecture must be considered to terminate components
    - So the conflict might arise whether a layered structure or redundant components are used when designing an architecture.

Question 6

- **Release testing**
    - This is where a separate testing team tests a complete version of the system before it is released to users.
- **User testing**

- This is where the users or potential users of a system test the system in their own environment

- **Difference between release testing and user testing**
  - Release testing aim to check that the system meets requirements of the system stakeholders while user testing aim to check if the software can be marketed, released and sold.

Question 7

- Key stages in the process of system construction by composition?
  - **Formulate outline workflow**
    - Initial stage of service design where requirements for the composite service are used as a basis for creating an ideal service design
  - **Discover services**
    - Look for existing services to include in the composition
  - **Select services**
    - Select possible services that can implement workflow activities
  - **Refine workflow**
    - Adding detail to the abstract description and perhaps adding or removing workflow activities
  - **Create workflow program**
    - During this stage, the abstract workflow design is transformed to an executable program and the service interface is defined.
  - **Test service**
    - Test the completed service/application

Question 8

8.1 Four (4) levels of software reuse

1. **Abstraction level**
   - At this level, you don't reuse software directly but rather use knowledge of successful abstractions in the design of your software
2. **Object level**
   - At this level, you directly reuse the objects from a library rather than writing the code yourself
3. **Component level**
   - At this level you adapt to an external component by adding some code of your own
4. **System level**
   - At this level, you reuse the entire application systems

8.2 Bayersdorfer suggests that companies managing projects that use open source should:

1. **Establish a system for maintaining information about open-source components that are downloaded and used** – Create a knowledge base, where information lice licenses and versions are accessible
2. **Be aware of the different types of licenses and understand how a component is licensed before it is used** – Make sure you fully understand the licensing agreement of the component
3. **Be aware of evolution pathways for components** – Follow the components projects on their platform like GitHub for example
4. **Educate people about open source** – Make sure that your developers are well equipped with the knowledge around open source
5. **Have auditing systems in place** - Make sure your developers comply with the open source agreement'
6. **Participate in open source community** – Make sure you become a community member of the open source, you help and participate