Computer Sciences  
University of Technology
Q1.

**System software**: It is a collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities).

**Software Engineering**: the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

**Personal Software Process (PSP)**: is a structured set of process descriptions, measurements, and methods that can help engineers to improve their personal performance.

**Risk refinement**: Process of restating the risks as a set of more detailed risks that will be easier to mitigate, monitor, and manage.

**Software Quality**: Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

Q2. **The RAD model**

*Rapid application development* (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a “high-speed” adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time periods (e.g., 60 to 90 days) [MAR91]. Used primarily for information systems applications, the RAD approach encompasses the following phases [KER94]:

**Business modeling.** The information flow among business functions is modeled in a way that answers the following questions: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?

**Data modeling.** The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics (called *attributes*) of each object are identified and the relationships between these object defined.

**Process modeling.** The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

**Application generation.** RAD assumes the use of fourth generation techniques. Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

**Testing and turnover.** Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised.
The RAD process model is illustrated in Figure 2.6. Obviously, the time constraints imposed on a RAD project demand "scalable scope." If a business application can be modularized in a way that enables each major function to be completed in less than three months (using the approach described previously), it is a candidate for RAD. Each major function can be addressed by a separate RAD team and then integrated to form a whole.

Like all process models, the RAD approach has drawbacks:

• For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.

• RAD requires developers and customers who are committed to the rapid-fire activities necessary to get a system complete in a much abbreviated time frame. If commitment is lacking from either constituency, RAD projects will fail.

• Not all types of applications are appropriate for RAD. If a system cannot be properly modularized, building the components necessary for RAD will be problematic. If high performance is an issue and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.

• RAD is not appropriate when technical risks are high. This occurs when a new application makes heavy use of new technology or when the new software requires a high degree of interoperability with existing computer programs.

Q3.

Figure (1.4) depicts failure rate as a function of time for hardware. The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level (ideally, quite low) for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out. Software is not susceptible to the environmental maladies that cause hardware to wear out. In theory, therefore, the failure rate curve for software should take the form of the "idealized curve" shown in Figure 1.5.

Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected (ideally, without introducing other errors) and the curve flattens as shown. The idealized curve is a gross oversimplification of actual failure models for software. However, the implication is clear - software doesn't wear out. But it does deteriorate! This seeming contradiction can best be explained by considering the "actual curve" shown in Figure 1.5. During its life, software will undergo change (maintenance). As changes are made, it is likely that some new defects will be introduced, causing the failure rate curve to spike as shown in Figure 1.5. Before the curve can return to the original steady state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise the software is deteriorating due to change.
Another aspect of wear illustrates the difference between hardware and software. When a hardware component wears out, it is replaced by a spare part. There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, software maintenance involves considerably more complexity than hardware maintenance.

![Figure (1.5): Idealized and actual failure curves for software](image)

Q4.

a. A number of basic principles guide software project scheduling (developing a schedule):

1. **Compartmentalization** - the product and process must be decomposed into a manageable number of activities and tasks.

2. **Interdependency** - tasks that can be completed in parallel must be separated from those that must completed serially, other activities can occur independently.

3. **Time allocation** - every task has start and completion dates that take the task interdependencies into account. And each task to be scheduled must be allocated some number of work units (e.g., person-days of effort).

4. **Effort validation** – every project has a defined number of staff members. As time allocation occurs, the project manager must ensure that on any given day there are enough staff members assigned to completed the tasks within the time estimated in the project plan.

5. **Defined Responsibilities** - every scheduled task needs to be assigned to a specific team member.

6. **Defined outcomes** - every task in the schedule needs to have a defined outcome (usually a work product or deliverable, for example the design of a module).
7. **Defined milestones** – every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products from an engineering task have passed quality review.

b. **The characteristic of software engineer**

1- Good programmer and fluent in one or more programming language.  
2- Well versed data structure and approaches.  
3- Familiar with several designs approaches.  
4- Be able to translate vague (not clear) requirements and desires into precise specification.  
5-Be able to converse with the user of the system in terms of application not in “computer”.  
6- Able to a build a model. The model is used to answer questions about the system behavior and its performance.  
7- Communication skills and interpersonal skills.  

Q5.  

a) **Software inspection**: Software inspections are reviews of the code with the purpose of detecting defects. In an inspection someone other than the programmer reads a program unit of limited size to determine whether it satisfies the requirements and specification. A formal process and checklist are used to ensure that no aspects are forgotten.  
 **Software Testing**: Testing each unit founded in the software, followed by testing software integration.  

b) **Essential view**: presents the functions to be accomplished and the information to be processed without regard to implementation.  
 **Implementation view**: presents the real world manifestation of processing functions and information structures.  

c) **Quality of design**: the characteristics that designers specify for an item. It includes: the grade of materials (requirements), tolerance, performance specifications, and design of the system.  
 **Quality of conformance**: the degree to which the design specification are followed during manufacturing. It focuses on implementation based on the design.