**Software Design**

**Final Paper**

**24 June 2020**

****

**Submitted by**

**Rizwan Mukhtiar**

MS (Computer Science)

Registration ID: 5952

**Assigned by**

**Dr. Fazal-E-Malik**

**Department of Computer Science**

**Faculty of Engineering and Applied Sciences**

**Iqra National University, Peshawar**

|  |  |  |
| --- | --- | --- |
| **Q.1** |  | **Is there any relationship among Client (Customer, sponsor), Developer and End User? If yes then explain.** |

Yes, there is a big relationship among client, supporting developers and the end user who operate the actual system in any organization. User involvement is the key concept in the development of useful and usable systems and has positive effects on system success and user satisfaction. Organizational information systems cannot produce any positive outcome unless the end users accept, adopt and use it. If any organization needs the systems to operate for their company this will extend the involvement through the end user who gives their input that how the system have the required functionalities and how the system will behave user friendly and accurately. End user involvement is necessary to manage the system so then the developer team after analysis developed the system for the organization on the basis of end user need. Software Development requires a large number of technical resources as well as stakeholders. Those are software developers on the technical side, and potential users on the market side. The success factor of software heavily depends on whether the software solution is able to fulfill the expectations of the addressed users. Software design experience teaches us that it gets more and more expensive to remove conceptual flaws the later the process of software development gets.

End-user centered design (UCD) methodologies for the development of software systems have received a lot of attention in recent times with high emphasis on user participation in the system development life cycle (SDLC). The direct participation of management and other end-users in SDLC contributes to the acceptability and usability of software systems. It also guides the software developers in the development and implementation of software systems that respond to both functional and non-functional requirements of the users.

End-user participation has been extensively studied in literature. However, there has been limited research on end-user participation in the developing world. Most of the information systems development and acceptance studies were carried out in developed countries where the culture could be described as associative, as against developing countries (especially in Africa) whose cultures could be described as abstractive where perception and behavior may not follow a clearly predictive pattern. End-user participation in software development could be highly contextual. The levels of and factors affecting end-user’s participation could vary across organizations and across countries. Heeks (2002) points to the high failure rate of information systems (IS) projects in the developing world, which may have in part resulted from low end-user participation.

End-user participation has not been extensively studied in developing countries when compared with those of developing countries. However, a number of end-user participation studies have been carried out, with emphasis on finding system development methodologies that would enhance end-user participation. A number of the studies attempt to identify factors that influence end-user participation in system development. End-user participation could be a perception issue. While end-users may believe that there is need for more participation, software developers on the other hand, may perceive adequate end-user participation based on the contextual elements of the project.

So the client or the sponsor usually prefer the end user requirements which will be friendly and operable to use for their organization and cost efficiently so then the sponsor makes the system accordingly on the basis of their business effective stage. Sometime end users have the a lot of those requirements which could be unnecessary useless for a company so then client just skip the requirement and move forward to make easier, efficient and cost effective system.

|  |  |  |
| --- | --- | --- |
| **Q.2** |  | **Explain the design “Trades-Offs” between the following:**   1. **Cost vs. Robustness** 2. **Cost vs. Reusability** 3. **Backward compatibility vs. Readability** |

A trade-off (or tradeoff) is a situational decision that involves diminishing or losing one quality, quantity or property of a set or design in return for gains in other aspects. In simple terms, a tradeoff is where one thing increases and another must decrease. During software development, tradeoffs are made on a daily basis by the people participating in the development project. Different roles in the project have to handle different tradeoffs. Some examples are that managers distribute work to developers and while doing so they have to balance the workload between the developers and deciding how many people that should be assigned to a particular task. If more people are assigned to a task then the task will be completed faster, but adding more people past a certain point only serves to increase the overhead of the group and in turn increases the time it takes to complete the task. Developers in turn make decisions regarding design and implementation details. An example is when software architects try to balance the quality attributes of the system. A balance of functional as well as quality requirements has to be achieved so that the intended users of the system will find it useful.

1. **Cost vs. Robustness**

Designing a production process normally is involved with some important constraints such as uncertainty, trade-off between production costs and quality, customer’s expectations and production tolerances. Nowadays, most engineering design methods try to assist decision makers for optimizing the processes and achieving the highest quality with minimum costs. The process of finding the accurate design parameters is stated as an optimization. Typically, any optimization technique needs to consider design constraints. It is the engineer’s duty to choose the design parameters according to an (or some) objective function. Process optimization is one of the intensive aspects of product development. During the optimization process, we need to maximize one or more parameters, while keeping all others within their constraints. The main goal is to reach a desired performance for the process that manufactures some products, by minimizing the cost of operation in a production process, or the variability of a quality characteristics by maximizing the yield of the production process. Furthermore, due to noisy data and/or uncertainty affecting some parameters of the model, achieving robust performance plays an essential role for engineering design problems.

1. **Cost vs. Reusability**

In most engineering disciplines, systems are designed by composing existing components that have been used in other systems. Software engineering has been more focused on original development but it is now recognized that to achieve better software, more quickly and at lower cost, we need a design process that is based on systematic software reuse. Information systems development is typically acknowledged as an expensive and lengthy process, often producing code that is of uneven quality and difficult to maintain. Software reuse has been advocated as a means of revolutionizing this process. The claimed benefits from software reuse are reduction in development cost and time, improvement in software quality, increase in programmer productivity, and improvement in maintainability. Software reuse does incur undeniable costs of creating, populating, and maintaining a library of reusable components. There is anecdotal evidence to suggest that some organizations benefit from reuse. However, many software developers practicing reuse claim these benefits without formal demonstration thereof.

1. **Backward compatibility vs. Readability**

Backward compatibility (sometimes backwards compatibility) is a property of a system, product, or technology that allows for [interoperability](https://en.wikipedia.org/wiki/Interoperability) with an older [legacy system](https://en.wikipedia.org/wiki/Legacy_system), or with [input](https://en.wikipedia.org/wiki/Input/output) designed for such a system, especially in [telecommunications](https://en.wikipedia.org/wiki/Telecommunications) and [computing](https://en.wikipedia.org/wiki/Computing). Backward compatibility is sometimes also called downward compatibility. Modifying a system in a way that does not allow backward compatibility is sometimes called "[breaking](https://en.wiktionary.org/wiki/breaking_change)" backward compatibility whereas the readability in software programming can be defined by the ease with which the software is read and understood. However, making software more readable helps in reviewing and maintaining it over the course of its life. Simplicity in logic, conditional statements, and the structure of the code all help with readability

|  |  |  |
| --- | --- | --- |
| **Q.3** |  | **What is the outcome of the software design? Explain in detail.** |

Software development is the process of conceiving, specifying, designing, [programming](https://en.wikipedia.org/wiki/Computer_programming), [documenting](https://en.wikipedia.org/wiki/Software_documentation), [testing](https://en.wikipedia.org/wiki/Software_testing), and [bug fixing](https://en.wikipedia.org/wiki/Software_bugs) involved in creating and maintaining [applications](https://en.wikipedia.org/wiki/Application_software), [frameworks](https://en.wikipedia.org/wiki/Software_framework), or other software components. Software development is a process of writing and [maintaining](https://en.wikipedia.org/wiki/Software_maintenance) the [source code](https://en.wikipedia.org/wiki/Source_code), but in a broader sense, it includes all that is involved between the conception of the desired software through to the final manifestation of the software, sometimes in a planned and [structured](https://en.wikipedia.org/wiki/Software_development_process) process. Therefore, software development may include research, new development, prototyping, modification, reuse, re-engineering, maintenance, or any other activities that result in software products.

* Upon successful completion of the program, you should be able to:
* Apply information technology principles and practices to real-world solutions
* Demonstrate effective use of written, verbal, and non-verbal communication, employing relevant knowledge, skills, and judgment in a business setting
* Manage a simple project and be able to contribute to a more complex project as a team member
* Program using one of at least two software languages to develop and evaluate software, hardware infrastructure, and network solutions to meet desired client outcomes
* Work as a professional maintaining high standards of practice, making ethical/legal judgments and decisions, and sustaining a professional standing through a commitment to life-long learning
* Develop and apply personal management and team member skills as a professional software developer
* Demonstrate employability skills and a commitment to professionalism

The period between the newly-developed project and the operational one is called a software development lifecycle, or SDLC for short. SDLC consists of a series of steps, or phases, that design a model for the [custom software development](https://qarea.com/services/custom-software-development) and its lifecycle management and brings it to the product release. Generally, software development cycle presupposes 7 following steps:

1. Planning
2. Feasibility analysis
3. Product design
4. Coding
5. Implementation and Integration
6. Software Testing
7. Installation and Maintenance

The [development of app](https://qarea.com/services/application-development), website, or software is a complex process, and a wrong step in any stage of software development will cause the inevitable outcomes both for the quality of product and the entire business. It involves hard work, dedication, and expertise in software development. Software development process is lengthy and needs step-by-step techniques following. So let us engage in detailed consideration of every stage of software development process to once and for all understand this important IT fundamental.

**1. Planning**

It is the phase of brainstorming when specialists gather requirements and analyze all the aspects of a future software product. The developers should understand the clients’ requirements, namely, what exactly they want and what issues can occur in the development process. This stage involves communication between stakeholders, project team, and users.

**2. Feasibility analysis**

At this step, the project team defines the entire project in details and checks the project’s feasibility. The team divides the workflow into small tasks, so that developers, testers, designers, and project managers can evaluate their tasks. They define whether it’s feasible in terms such of cost, time, functioning, and reliability.

**3. Software Design**

The software design is the major aspect of [software development services](https://qarea.com/services) cycle. Design should be creative and clear. It involves overall product design along with data structure and database design. Software designing uses many different strategies. We care about the graphic interface of each product we work with.

**4. Programming**

This is the critical phase of SDLC. A lot of brains work for coding and deliver the desired software. Usually, a company assigns a team of programmers for a particular project. The tasks are subdivided into sub-phases called Task Allocation, so every coder has their own task.

**5. Implementation and Integration**

Normally software contains a great number of programs, which require careful implementation and step-by-step integration of the software product. During this software stage, the project team checks whether the software product runs on various systems. In case of bugs, testers fix them.

**6. Software Testing**

After completing of coding, the software is sent to the testing department. The work of testers plays the crucial role for the quality of software and its performance. Quality Analysts test software using various test cases.  Before the launch, a product needs verification which includes [software testing](https://testfort.com/) and debugging done by testers. When testing department ensured that software is error-free, it goes to the next stage.

**7. Installation and Maintenance**

Finally, the software is handed over to the clients to be installed on their devices. After the installation, if the client needs any modification, the product is to come under the maintenance process.

The featured stages of software development procedure are followed by the majority of IT companies in order to provide high-quality services in the development of all sorts of software.

|  |  |  |
| --- | --- | --- |
| **Q.4** |  | **What is ADL (Architectural Descriptive Language)? How many ADLs are there? Explain one of them.** |

**What is ADL (Architectural Description Language)?**

It is a modeling notation to support architecture-based development used to define and model system architecture prior to detailed design and implementation. ADLs are formal languages for describing the architecture of a software system. Each ADL defines a notation with precise syntax and semantics in which architecture models can be expressed, and provides a corresponding toolkit for working with the language. ADLs include general purpose languages like xADL and ACME, and [domain-specific languages](https://www.sciencedirect.com/topics/computer-science/domain-specific-language) (DSLs). ADLs support the description of structural and selected behavioral aspects. An ADL describes a system at the component and connector abstraction level. A system is a configuration of components and connectors. Components are units of computation and data stores. Connectors describe interactions between components and the rules that govern these interactions. The supported behavioral aspects are different for each ADL. There are number of ADLs are available, some of them are as follows:

**1) ACME** **Architectural Description Language**

Numerous architectural description languages (ADLs) have been developed, each providing complementary capabilities for architectural development and analysis. Unfortunately, each ADL and supporting toolset operates in isolation, making it difficult to integrate those tools and share architectural descriptions. Acme is being developed as a joint effort of the software architecture research community as a common interchange format for architecture design tools. Acme provides a structural framework for characterizing architectures, together with annotation facilities for additional ADL specific information. This scheme permits subsets of ADL tools to share architectural information that is jointly understood, while tolerating the presence of information that falls outside their common vocabulary.

**2) SADL** **Architectural Description Language**

The Simulation Architecture Description Language (SADL) developed at the National Aeronautics and Space Administration’s Marshall Space Flight Center to support the real-time simulation of advanced avionics systems. SADL is a graphical, domain-specific Architecture Description Language that facilitates the high level specification of both the software and hardware aspects of hard real-time, hardware in the loop, avionics system simulations targeted for execution on diverse hardware architectures including multiprocessor systems. It supports the hierarchical expression of the architecture of an entire simulation at various levels of abstraction. Real-time computation occurs when the computer’s internal concept of time is closely synchronized with actual time. One area in which real-time computation is required is simulation of advanced avionics systems. This is a very demanding type of computation requiring complex calculations within small time frames. It requires state of the art hardware to provide the computational speed required and special system software that is designed to minimize the overhead associated with program execution. Real-time simulation plays an important role in the evaluation and testing of avionics systems. It provides a means of analyzing a design before it is built to uncover design flaws and test proof-of-concept early in the design process. Also, real-time simulation can facilitate the testing of existing hardware subsystems by imitating the real time operation of subsystems interconnected to the hardware. In this way, the test article can be fully evaluated without the presence of the subsystem hardware. Finally, real time simulation can aid in project planning by providing virtual hardware.

**3) UML Architectural Description Language**

Architecture Description Languages (ADLs) are specialized formal languages supporting modeling and reasoning on software architectures. Although number of ADLs counts in the tens, their popularity and usage by practitioners is very low. The object oriented Unified Modeling Language (UML), which has become the OMG standard, offers a great variety of concepts for the definition of the structure and the expected behavior of a software system. Unified Modeling Language is de facto industrial standard, however not fully qualified ADL. It has the potential to replace many previously used software architecture description language. Compared with other ADLs, UML has the main drawback that its module concept is continuously changing from version to version without reaching a well-defined state. It is the purpose of this contribution to revisit the development of the UML module concept, to criticize its current form, and to present a compact and precise definition of its visibility rules.

**4) AESOP Architectural Description Language**

The Aesop System provides a toolkit for rapidly building software architecture design environments specialized for domain specific architectural styles. An open tool integration framework that supports cooperation between Aesop and your favorite tools. A "Style-aware" repository for storing, retrieving, and reusing architectural design elements. A persistent object-oriented database for storing and manipulating architectural designs. A customizable Tcl/Tk based graphical user interface. It helps you to exploit design and domain knowledge when producing domain specific architectures. Also helps in rapidly generate design environments for producing families of software systems. It is Store, retrieve, and organize libraries of "style-aware" architectural design elements and patterns with the Software Shelf.

Aesop provides a generic toolkit and communication infrastructure that users can customize with architectural style descriptions and a set of tools that they would like to use for architectural analysis. Example tools that we have integrated with our Aesop styles include: cycle detectors, type consistency verifiers, formal communication protocol analyzers, C-code generators, compilers, structured language editors, and rate-monotonic analysis tools.

**5) Rapide Architectural Description Language**

A language and toolset for simulation of distributed systems by partial orderings of events The Rapide Language effort focuses on developing a new technology for building large-scale, distributed multi-language systems. This technology is based upon a new generation of computer languages, called Executable Architecture Definition Languages (EADLs), and an innovative toolset supporting the use of EADLs in evolutionary development and rigorous analysis of large-scale systems. Rapide is designed to support component-based development of large, multi-language systems by utilizing architecture definitions as the development framework. Rapide™ adopts a new event-based execution model of distributed, time-sensitive systems. the “timed poset model.” posets provide the most detailed formal basis to date for constructing early life cycle prototyping tools, and later life cycle tools for correctness and performance analysis of distributed time-sensitive systems.

|  |  |  |
| --- | --- | --- |
| **Q.5** |  | **What is the Architectural Style? Explain components of a style?** |

**What is the Architectural Style?**

An architecture style (also known as an “architecture pattern”) abstracts the common properties of a family of similar designs. Define a family of systems in terms of a pattern of its structural organization. An [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) is a general, reusable solution to a commonly occurring problem in [software architecture](https://en.wikipedia.org/wiki/Software_architecture) within a given context. Architectural patterns are often documented as software [design patterns](https://en.wikipedia.org/wiki/Design_pattern_(computer_science)). In software engineering, an Architectural Pattern is a general and reusable solution to an occurring problem in a particular context. It is a recurring solution to a recurring problem. The purpose of Architectural Patterns is to understand how the major parts of the system fit together and how messages and data flow through the system.

Every software requires a proper plan and detailed blueprint before stepping into the development. Software architecture is the high level structure used for creating software systems and is actually a step-by-step blueprint of the entire software that is to be built. The purpose of the software and its specific functionalities are defined by the software's architectural style and pattern used. The architectural style is a very specific solution to a particular software which typically focuses on how to organize the code created for the software. It is the granularity of highest level that focuses on creating the layers and modules of the software and allowing an appropriate interaction between the various modules for giving the right results upon implementation.

**Components of Style**

Elements/components of style is that perform functions required by a system. Basic components or elements are as follows:

**Connectors**

Connectors enable communication, coordination, and cooperation among elements. Simply a software connectors perform transfer of control and data among components. Connectors can also provide services, such as persistence, invocation, messaging, and transactions, that are largely independent of the interacting components' functionalities

**Constraints**

Constraints define how elements can be integrated to form the system. Constraints are a very important feature in a relational model. In fact, the relational model supports the well-defined theory of constraints on attributes or tables. Constraints are useful because they allow a designer to specify the semantics of data in the database. *Constraints* are the rules that force DBMSs to check that data satisfies the semantics

**Attributes**

In [computing](https://en.wikipedia.org/wiki/Computing), an attribute is a specification that defines a [property](https://en.wikipedia.org/wiki/Property_(programming)) of an [object](https://en.wikipedia.org/wiki/Object_(computer_science)), element, or file. It may also refer to or set the specific [value](https://en.wikipedia.org/wiki/Value_(computer_science)) for a given instance of such. For clarity, attributes should more correctly be considered [metadata](https://en.wikipedia.org/wiki/Metadata). Attributes describe the advantages and disadvantages of the chosen structure.

**Elements of an Architectural Design**

The objective of architecture design is to determine how the software components of the information system will be assigned to the hardware devices of the system. In this section, we first discuss the major functions of the software to understand how the software can be divided into different parts. Then we briefly discuss the major types of hardware onto which the software can be placed. Although there are numerous ways in which the software components can be placed on the hardware components, the most common architecture is the client–server architecture, so we focus on it here.

The major architectural components of any system are the software and the hardware. The major software components of the system being developed have to be identified and then allocated to the various hardware components on which the system will operate. Each of these components can be combined in a variety of different ways.

All software systems can be divided into four basic functions. The first is data storage. Most information systems require data to be stored and retrieved, whether a small file, such as a list of lawn chemicals that are no longer authorized for residential applications, or a large database that stores an organization's human resources records. These are the data entities documented in ERDs. The second function is the data access logic: the processing required to access data, often meaning database queries in Structured Query Language (SQL). The third function is the application logic: the logic documented in the DFDs, use cases, and functional requirements. The fourth function is the presentation logic: the display of information to the user and the acceptance of the user's commands (the user interface). These four functions (data storage, data access logic, application logic, and presentation logic) are the basic building blocks of any information system.

The three primary hardware components of a system are client computers, servers, and the network that connects them. Client computers are the input–output devices employed by the user and are usually desktop or laptop computers, but can also be handheld devices, smartphones, special-purpose terminals, and so on. Servers typically are larger multi-user computers used to store software and data that can be accessed by anyone who has permission. The network that connects the computers can vary in speed from slow cell phones or modem connections that must be dialed, to medium-speed always-on frame relay networks, to fast always-on broadband connections such as cable modem, DSL, or T1 circuits, to high-speed always-on Ethernet, T3, or ATM circuits.