

ASSIGNMENT 1
Individual
30%

CHAPTER /TOPIC: Design Engineering and Creating An Architectural Design

INSTRUCTION:

1. Report MUST contain minimum pages of 5 and maximum of 10.
2. Each student is required to submit an individual written report that consists of Introduction, Body and References
3. Mode: Individual
4. Submission in the form of written report (Hard Copy)
5. Font size 11, Font Type : Arial, Line Spacing 1.5 points
6. Date of Submission: **14 OCTOBER 2024.**
7. Use the attached cover paper printed on Yellow paper ONLY. Attach the rubrics for marking.

Provision for Submission

If you have any difficulties in submitting the assignment according to the date given, please come and discuss it with me.


COURSE OUTCOME(S):

CO1 explain and apply a concept of software design and architecture design individually.

1. Explain the concepts of diversification and convergence in the context of software design.
2. What is the role of requirements engineering and analysis modeling in the diversification process?
3. Why is the software design process described as iterative?
4. What is the purpose of a blueprint in software design?
5. What are the two dimensions in which the design model can be viewed?
6. How does the process dimension influence the design model?
7. What does the abstraction dimension represent in the design model?
8. How are UML diagrams used in both the analysis model and design model?
9. Why is software architecture important in the development of a computer-based system?
10. What is the purpose of data design in software development?

MANAGEMENT & SCIENCE UNIVERSITY

CSE20203**SOFTWARE ARCHITECTURE, DESIGN AND TESTING****SEPTEMBER 2024****ASSIGNMENT 1**

Name: Syasswin A/L Navaratnam
ID: 2022023020018


Given On	7th October 2024
Submission Date	14th October 2024
Submitted On	14th October 2024

Introduction

Effective design strategies are required in the field of software development to create high-quality, flexible, and maintainable systems. This session delves into a variety of fundamental concepts in software design, such as convergence and diversity, which foster creative thinking through experimentation and progress. We will also examine how requirements engineering and analytical modelling aid in the diversification process, emphasizing how software design is iterative and allows for continual improvement. The function of design models and blueprints, as well as their dimensions (process and abstraction dimensions), and the application of UML diagrams in analysis and design settings, will all be discussed. We will also examine the significance of data design and software architecture in developing dependable computer-based systems. Understanding these concepts is critical to developing software that is both usable and long-lasting.

1. Explain the concepts of diversification and convergence in the context of software design.

Diversification and convergence are two key ideas in software design that influence choices at every stage of the development process. Each contributes in a different way to maintaining software system quality, guaranteeing flexibility, and controlling complexity.

Software Design Diversification

The practice of developing variants or several alternate ways to handle various facets of a software system's functionality is known as diversification. It is a proactive strategy meant to adapt to changing demands from users, the environment, or requirements. The goal of diversity is to give the software product flexibility and adaptability, which can be essential in a rapidly evolving technical environment.

Important instances of diversity in software design are as follows:

- Platform independence refers to the ability of software, such as cross-platform apps, to function across various hardware architectures and operating systems.
- Modular Design: The system is divided into self-contained modules, enabling the development and upkeep of each element independently.

- **Algorithmic Decisions:** Applying various algorithms to the same functionality and allowing the user to choose between them in accordance with resource or performance limitations.

As it enables the exploration of multiple design alternatives that may satisfy the project's goals in diverse ways, diversification is essential in the early stages of design, particularly during requirements engineering and analytical modelling.

Software Design Convergence

Conversely, convergence describes the process of reducing a number of competing ideas or methods to a single, cohesive answer. It frequently comes after the diversification stage and is crucial to preserving the software system's simplicity, consistency, and focus. Designers and developers can unify disparate design routes through convergence to guarantee seamless system integration and optimal performance.

Here are a few instances of convergence in software design:

- **Standardisation** is the process of making sure that a system's modules or components all employ the same data formats, protocols, and architectural patterns.
- **Integration** is the process of combining various parts or services into a unified system and making sure that everything functions as a whole.
- **Technology Stack Alignment:** Using a standard set of technologies to align the software architecture reduces complexity and increases maintainability.

Later phases of development, especially the software blueprint phase and iterative design procedures, are when convergence frequently happens as the design approaches finalisation and refinement.

2. What is the role of requirements engineering and analysis modeling in the diversification process?

During the diversification stage of the software development lifecycle, requirements engineering and analysis modelling are essential components.

Eliciting, evaluating, recording, and validating the requirements that the program must meet are all part of requirements engineering. This procedure guarantees that the development team is aware of the business objectives and stakeholder needs. Requirements engineering provide the data needed to produce several possible solutions at the diversification stage. Developers can come up with a variety of designs, architectures, and methods to satisfy requirements when they are aware of both functional and non-functional requirements. Requirements also aid in identifying limitations that will affect the breadth of diversification, such as performance or security considerations.

Conversely, analysis modelling converts the requirements into organized representations that may be utilized to investigate various design options. These models, which can be entity-relationship models, use case diagrams, or data flow diagrams, operate as abstractions for the structure, behaviour, and data flow of software systems. Analysis models aid in the visualization of many system views during the diversification process, allowing the team to examine alternate approaches to reaching the system's objectives. These models are the starting point for investigating various architectural and design possibilities, which facilitates the creation of several workable solutions.

Requirements engineering essentially gives the "what" of the system, while analytical modelling provides the "how." Together, they set specific objectives and investigate several strategies to achieve them, which accelerates the diversification process.

3. Why is the software design process described as iterative?

Since there are constant cycles of design, testing, evaluation, and refinement, the software design process is referred to as iterative. Software design is not a linear, one-time process; rather, it is a process that changes across several iterations in order to consider user feedback, adjust to changing needs, and enhance the final product's quality.

Since early designs are rarely ideal, software design is inherently iterative. Early on in the process, developers might provide high-level designs or prototypes that are evaluated in relation to performance benchmarks, user needs, or other standards. In later stages, the design is refined and adjusted in response to the input received from these assessments.

There are various advantages to this iterative method:

- Flexibility to handle changes as company demands and user feedback evolve, software projects frequently encounter requirements modifications. Designers can make these modifications without completely halting the project by using an iterative process.
- Risk management where iteration makes it possible to identify technical problems or design faults early on, allowing for their resolution before they become expensive or difficult to fix.
- Progressive refinement in every iteration enables ongoing design enhancement, leading to a more polished and effective finished product.
- User-centered development by collecting feedback from stakeholders or end users at different phases, designers may ensure that the software meets user expectations and needs through iterations.

4. What is the purpose of a blueprint in software design?

In software design, a blueprint is a comprehensive plan or road map that directs the software system's development from idea to completion. A software blueprint offers a clear and organized picture of the system, outlining its parts, architecture, linkages, and behaviours, much like an architectural design for a building.

A blueprint serves a variety of purposes and is essential to the design process for the following reasons:

Clarity and communication where a software blueprint facilitate communication between developers, architects, business analysts, and project managers, among other stakeholders. It offers a common knowledge of the design and operation of the system. Through the use of models, diagrams, and other artefacts, team members can more easily visualise the system's design, which facilitates collaboration, identification of possible problems, and informed decision-making.

Structured design such as software design follows a set of rules that are enforced by blueprints. The development team first outlines the high-level architecture, components, data models, and interaction processes of the system before diving right into coding. This methodical technique guarantees that the system is coherently arranged and that the interdependencies among various elements are thoroughly comprehended.

Planning and resource allocation by defining the scope of the task, blueprints aid in project planning. Project managers can make more realistic estimates of costs, resources, and timelines when the design is thoroughly documented. The blueprint also assists developers in comprehending the interoperability of various components, which minimises misunderstandings and implementation inefficiencies.

Risk reduction where early on in the process, the team can spot possible obstacles or bottlenecks thanks to a clearly defined plan. Teams can address scalability, performance, security, and integration issues before they become major problems during development by outlining the architecture of the system beforehand. By being proactive, the likelihood of expensive design errors is reduced.

Code development guide during coding and implementation, the blueprint acts as a guide for developers. It provides a guide for interacting between various modules and components as well as for adhering to specific architectural styles or design patterns. This guarantees that the final code is in line with the overall system architecture and expedites the development process.

Upkeep and future improvements when the blueprint is also essential to the program's long-term upkeep. A well-documented architectural blueprint facilitates system modification and extension as software systems grow. Developers can consult the blueprint whenever they add new features or resolve issues to make that the integrity of the system is maintained.

5. Which Two Dimensions Are Available for Viewing the Design Model?

Two dimensions can be used to view the software development design model: the process dimension and the abstraction dimension.

First is Process Dimension where the way the design model changes over time is indicated by this. It shows how the design model evolved from the first concept to the finished structure of the software design process. It documents the phases or procedures that are needed to convert user requirements into a fully functional software architecture and design.

Secondly, the levels of abstraction or detail that are portrayed in the design are referred to as the abstraction dimension. Several degrees of abstraction are used as the design develops, beginning with high-level conceptual models and working down to more intricate, tangible implementations. This dimension makes sure that as the project moves forward, design starts off broadly and gets progressively more detailed.

6. How Does the Design Model Get Affected by the Process Dimension?

By describing the phases and processes the design goes through from inception to completion, the process dimension has an impact on the design model. It divides the design work into doable stages, each of which helps to create the system incrementally. For example, early phases could concentrate on high-level system design, whereas later phases would delve into particular parts and interfaces.

This dimension ensures that modifications and choices made in previous phases are reviewed and verified as the system design progresses, leading the development team through a systematic, iterative refinement of the software. It promotes iteration since each stage builds upon and improves upon earlier design aspects in light of user feedback, testing, and fresh perspectives.

7. In the Design Model, what does the Abstraction Dimension Stand for?

Various levels of detail in the software's design are represented by the abstraction dimension in the design model. Designers can concentrate on particular system levels, from broad, abstract notions to more specific, tangible software representations, thanks to abstraction. Usually, there are multiple layers of abstraction:

High-level Abstraction where this describes general notions like as modules, subsystems, or services without getting into implementation specifics. It encompasses the system's design and basic structure.

Low-level Abstraction refers to a more intricate design that includes particular implementation details as well as class structures, algorithms, and data structures.

Applying abstraction makes design easier to handle and enables a gradual transition from rough, conceptual concepts to clean, executable code.

8. How Are the Analysis and Design Models and Analysis Models Using UML Diagrams?

Diagrams created using the Unified Modelling Language (UML) are essential tools for software development analysis and design models.

The system's functionality and user interactions are represented using UML diagrams in the analysis model. Use case diagrams, for instance, illustrate how users will interact with the system to capture functional requirements, whereas activity diagrams and sequence diagrams describe processes and interactions between various system actors.

UML diagrams are used in the Design Model to describe the internal architecture and structure of the system. Class Diagrams define classes, characteristics, and relationships to provide specifics about object-oriented design. Additional information on how various software components are distributed across hardware resources and interact during runtime is provided by component and deployment diagrams.

UML diagrams thereby offer coherence between the analysis phase, which captures system requirements, and the design phase, which specifies how those requirements will be realised.

9. What is the significance of software architecture in the creation of computer-based systems?

Software architecture is essential to the creation of computer-based systems because it establishes the overall structure of the system and provides a blueprint for how its constituent parts will communicate and work as a unit. Software architecture is significant for the following main reasons:

Basis for Design and Development where It offers a methodical structure for disassembling the system into smaller, more manageable parts, assisting developers in the construction, upkeep, and evolution of the system.

Performance and Scalability where robust architecture guarantees that the system can grow with the addition of new features or an increase in demand.

Flexibility and Maintainability is a well-designed architecture makes it possible for the system to be readily expanded, changed, or maintained over time without the need for significant rewrites.

Risk mitigation is the process of making sure that the system's fundamental architecture permits reliable and effective development, which helps to detect and mitigate possible risks early on, such as performance bottlenecks or integration issues.

Software architecture, in its whole, offers a thorough development roadmap for the system, guaranteeing that the final result satisfies both functional and non-functional requirements.

10. In software development, what is the goal of data design?

Arranging and structuring the data that the software system will utilise and process is the goal of data design in software development. Making a thorough plan for the system's overall data storage, access, and manipulation is known as data design. Among the main goals of data design are:

Data design makes ensuring that the data is organised in a way that keeps it consistent throughout the system and guards against corruption.

Optimizing Data Access that outlines effective techniques for data manipulation and access, which enhances system performance, particularly in applications with huge amounts of data.

Supporting Scalability an appropriately constructed data structure ensures that the system can scale over time by allowing for the handling of growing data quantities without degrading performance.

Enabling Data Relationships where data design helps the system simulate intricate real-world relationships by determining how various data items link to one another, for example through entity-relationship diagrams (ERDs).

To sum up, data design is essential for building a strong base on which the software's data handling functions may be carried out effectively and dependably, guaranteeing that the system functions well and expands appropriately.

Conclusion

To sum up, the ideas of convergence, diversity, and iterative design procedures are crucial elements of successful software development. Teams can test a variety of options and make sure the end result meets user expectations by combining requirements engineering with analysis modelling. Serving as a fundamental roadmap, the blueprint encourages organisation and clarity throughout the design process. Furthermore, employing UML diagrams and comprehending the dimensions of design models improves planning and communication, which in turn helps to create software architectures that are resilient. Acknowledging the significance of data design also guarantees that systems are not only operational but also expandable and easily maintained. These ideas work together to create a coherent framework that facilitates the effective creation of computer-based systems and opens doors for creativity and flexibility in a rapidly changing technological environment.

Rubric for evaluating the assignment:

Criteria	Exemplary (4)	Proficient (3)	Basic (2)	Needs Improvement (1)	
1. Explanation of Diversification and Convergence	Thoroughly explains both concepts in the context of software design, clearly showing the role of each. Provides examples for clarity.	Explains both concepts accurately with some reference to their role in design.	Defines diversification and convergence but lacks context in software design.	Incomplete or unclear explanation of diversification and convergence.	5
2. Role of Requirements Engineering and Analysis Modeling	Clearly describes how these guide the diversification process and their importance in selecting design components.	Describes the role of requirements engineering and analysis modeling in the diversification process, though missing some depth.	Mentions requirements engineering and analysis modeling, but with vague explanations.	Fails to connect the role of requirements engineering and analysis modeling to the diversification process.	7.5
3. Iterative Nature of Software Design	Provides a detailed explanation of why software design is iterative, including continuous refinement and feedback.	Explains that design is iterative, with some reference to feedback or refinement.	Mentions iteration but lacks detail on why the process is continuous.	Does not explain why software design is iterative or provides a minimal answer.	10
4. Purpose of a Blueprint in Software Design	Clearly explains the purpose of a blueprint, detailing its role in guiding the construction of software.	Explains the purpose of a blueprint but lacks detail on its significance in the software construction process.	Defines a blueprint but does not relate it to guiding software construction.	Incomplete or unclear explanation of a blueprint in software design.	10
5. Two Dimensions of the Design Model	Accurately identifies and explains both the process and abstraction dimensions with examples.	Identifies both dimensions and provides some explanation, but lacks examples or depth.	Identifies the dimensions but offers minimal explanation.	Fails to accurately identify or explain both dimensions.	7.5
6. Influence of the Process Dimension	Thoroughly explains how the process dimension affects the design model, including its role in the evolution of the design.	Explains how the process dimension affects the design model with limited depth.	Mentions the process dimension but lacks clarity on its influence.	Does not explain how the process dimension influences the design model.	7.5
7. Abstraction Dimension Representation	Clearly describes the abstraction dimension and its significance in refining the design model with examples.	Describes the abstraction dimension but lacks examples or depth in explanation.	Mentions the abstraction dimension with minimal explanation of its significance.	Incomplete or unclear explanation of the abstraction dimension.	7.5
8. Use of UML Diagrams in Analysis and Design Models	Clearly explains how UML diagrams are used in both models and provides specific	Explains how UML diagrams are used in both models, but lacks specific examples.	Mentions UML diagrams but does not explain how they are	Fails to explain or inaccurately describes the use of UML diagrams in	10

	examples of refinement and elaboration.		used in both models.	both models.
9. Importance of Software Architecture	Provides a detailed explanation of the importance of software architecture in development, including its impact on stakeholders and success.	Explains the importance of software architecture, but lacks detail on its role in communication and success.	Mentions software architecture but provides minimal explanation of its importance.	Fails to explain why software architecture is important or provides an unclear response.
10. Purpose of Data Design in Software Development	Thoroughly explains the purpose of data design, focusing on data structure representation and the challenge of efficient data storage and retrieval.	Explains the purpose of data design, but lacks detail on the challenge of data storage and retrieval.	Mentions data design but does not fully explain its purpose or challenge.	Incomplete or unclear explanation of data design's purpose.

10

10

Total marks 100 (30)

85