Glossary of Terms for CENG350

June 26, 2024

Glossary

0.1 Chapter 1

Software Instructions (programs) that provide desired features, functions, and performance; data structures that enable programs to manipulate information; and documentation that describes the operation and use of programs.

System Software Software designed to provide a platform for other software. Examples include operating systems and utility software.

Application Software Software designed to help the user perform specific tasks. Examples include word processors, spreadsheets, and media players.

Engineering/Scientific Software Software used for scientific research and engineering applications, such as simulation software and scientific calculators.

Embedded Software Software written to control machines or devices that are not typically considered computers. Examples include software for automotive systems and industrial machines.

Product-line Software Software developed to be used in a family of products. Examples include software components shared across a range of products from a single company.

Web Applications (WebApps) Software that runs on web servers and can be accessed through web browsers.

AI Software Software that enables computers to perform tasks that typically require human intelligence, such as visual perception, speech recognition, and decision-making.

Open World Computing Pervasive, distributed computing environments.

Ubiquitous Computing Computing that is available anytime and anywhere through wireless networks.

Netsourcing Using the web as a computing engine.

Open Source Software Software with source code that is available for anyone to use, modify, and distribute.

Data Mining The process of discovering patterns and knowledge from large amounts of data.

Grid Computing Using a network of computers to work together to solve complex problems.

Cognitive Machines Machines that can mimic human cognitive functions such as learning and problem-solving.

Legacy Software Existing software that must be adapted, enhanced, extended, or re-architected to remain useful.

Concurrency The ability of a system to handle multiple tasks simultaneously.

Process Model A representation of the process used to develop software.

Quality Assurance (QA) Activities designed to ensure that the software meets certain standards of quality.

Configuration Management The process of managing changes to software to maintain its integrity over time.

Risk Management The process of identifying, assessing, and prioritizing risks, and then taking steps to minimize their impact.

0.2 Chapter 2

Software Process A structured set of activities required to develop a software system, including specification, design, implementation, validation, and evolution.

Specification The process of defining what a system should do.

Design and Implementation Defining the organization of a system and converting the specification into an executable system.

Validation Checking that the system does what the customer wants.

Evolution Changing the system in response to changing customer needs.

Plan-driven Processes Processes where all activities are planned in advance and progress is measured against the plan.

Agile Processes Processes with incremental planning and easier adaptation to changes.

Waterfall Model A plan-driven software process model with distinct phases of specification and development.

Incremental Development A process where specification, development, and validation activities are interleaved.

Integration and Configuration Developing a system by assembling existing configurable components.

Component-based Software Engineering (CBSE) A method of software development that involves assembling software from existing components.

Requirements Engineering The process of defining, documenting, and maintaining software requirements.

Verification and Validation (V&V) Activities that ensure a system meets its specifications and customer requirements.

System Testing Testing the system as a whole to ensure it meets its requirements.

Software Evolution The process of modifying software to adapt to new conditions or correct faults.

Prototyping Creating an initial version of a system to demonstrate concepts and try out design options.

Incremental Delivery Delivering a system in increments, each providing part of the required functionality.

Reuse-oriented Software Engineering Developing software by integrating and configuring existing components or systems.

Change Anticipation Including activities in the software process that anticipate possible changes.

Change Tolerance Designing the process to accommodate changes at a relatively low cost.

0.3 Leloudas Chapter 2

Dynamic Testing Testing that involves executing the code with test cases.

Static Testing Testing that involves checking the code and documents without executing the code.

Unit Testing Testing individual components.

Integration Testing Testing combined parts of an application to ensure they work together.

System Testing Testing the complete integrated system.

Acceptance Testing Testing the system's compliance with the business requirements.

Black-box Testing Testing without knowledge of the internal workings of the application.

White-box Testing Testing with knowledge of the internal workings of the application.

0.4 Chapter 3

Agile Development An approach to software development with interleaved specification, design, and implementation, focusing on rapid delivery, minimal documentation, and frequent releases.

Plan-driven Development A structured approach to software engineering with separate development stages and planned outputs for each stage.

Extreme Programming (XP) An agile method emphasizing customer satisfaction, frequent releases, continuous testing, and collaboration.

Incremental Delivery Delivering software in small, usable increments, allowing for frequent customer feedback and adjustments.

Customer Involvement Engaging customers throughout the development process to provide and prioritize requirements and evaluate iterations.

Refactoring The process of restructuring existing code without changing its external behavior to improve readability and reduce complexity.

Test-first Development Writing tests before the actual code to clarify requirements and ensure functionality from the beginning.

Pair Programming Two programmers working together at one workstation, collaborating on code development and review.

Continuous Integration Integrating and testing code frequently to catch issues early and ensure ongoing functionality.

Scrum An agile framework for managing iterative development, with roles such as Product Owner and ScrumMaster, and events like sprints and daily scrums.

Agile Manifesto A declaration of the core values and principles of agile software development, emphasizing individuals, working software, customer collaboration, and response to change.

Scaling Agile Adapting agile methods to handle large, complex projects and distributed teams while maintaining core agile principles.

Sustainable Pace Ensuring that the development team works at a pace that can be maintained indefinitely without burnout.

User Stories Simple descriptions of a software feature from an end-user perspective, used in agile development to capture requirements.

Automated Testing Using software tools to run tests on code automatically, ensuring consistency and reducing manual testing effort.

Incremental Planning Planning software development in small, manageable increments rather than a single, detailed upfront plan.

Simple Design Focusing on designing only what is necessary for current requirements, avoiding over-engineering.

0.5 Chapter 4

Requirements Engineering The process of establishing the services a customer requires from a system and the constraints under which it operates and is developed.

User Requirements Statements in natural language plus diagrams of the services the system provides and its operational constraints.

System Requirements A structured document setting out detailed descriptions of the system's functions, services, and operational constraints.

Functional Requirements Statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations.

Non-functional Requirements Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, and quality requirements (e.g., usability, performance).

System Stakeholders Individuals or organizations affected by the system in some way and who have a legitimate interest.

Requirements Elicitation The process of working with stakeholders to find out about the application domain, the services that the system should provide, and the system's operational constraints.

Requirements Specification The process of writing down the user and system requirements in a requirements document.

Requirements Validation The process of demonstrating that the requirements define the system that the customer really wants. It involves checks for validity, consistency, completeness, realism, and verifiability.

Requirements Management The process of managing changing requirements during the requirements engineering process and system development.

Requirements Document The official statement of what is required of the system developers. It includes both user and system requirements.

Requirements Change Management The process of deciding if a requirements change should be accepted, analyzing the change, and implementing it.

Requirements Review A systematic manual analysis of the requirements to ensure they meet the necessary criteria.

Ethnography A social science research method that involves observing and analyzing how people actually work to derive requirements.

Use Cases Scenarios that identify the actors in an interaction and describe the interaction itself.

Prototype An initial version of a system used to demonstrate concepts and try out design options.

Domain Requirements Constraints on the system from the domain of operation, such as legal or regulatory constraints.

Spiral Model An iterative model of the requirements engineering process that includes cycles of requirements discovery, classification, negotiation, and documentation.

0.6 Chapter 5: System Modeling

System Modeling The process of developing abstract models of a system to present different views or perspectives.

Unified Modeling Language (UML) A standardized modeling language used to specify, visualize, construct, and document the artifacts of software systems.

Context Models Models that illustrate the operational context of a system and show what lies outside its boundaries.

Interaction Models Models that highlight the interactions between a system and its environment or between system components.

Structural Models Models that display the organization of a system in terms of its components and their relationships.

Behavioral Models Models that show the dynamic behavior of a system and how it responds to events.

Activity Diagrams UML diagrams that show the activities involved in a process or data processing.

Use Case Diagrams UML diagrams that show the interactions between a system and its environment.

Sequence Diagrams UML diagrams that show interactions between actors and the system or between system components.

Class Diagrams UML diagrams that show the object classes in a system and the associations between these classes.

State Diagrams UML diagrams that show how the system reacts to internal and external events.

Model-driven Engineering (MDE) An approach to software development where models are the principal outputs, and programs are generated from these models.

0.7 Leloudas Chapter 5

Test Design Techniques Techniques for designing effective test cases.

Equivalence Partitioning Dividing input data into partitions and testing one value from each.

Boundary Value Analysis Testing at the boundaries between partitions.

Decision Table Testing Using decision tables to represent combinations of inputs and outputs.

State Transition Testing Testing state changes in the application.

0.8 Chapter 6: Architectural Design

Architectural Design The process of defining a structured solution that meets all the technical and operational requirements.

4+1 View Model A model of software architecture describing logical view, process view, development view, physical view, and scenarios.

Model-View-Controller (MVC) An architectural pattern that separates an application into three main components: Model, View, and Controller.

Layered Architecture Organizes the system into layers with related functionality, where each layer provides services to the layer above it.

Repository Pattern Manages all data in a system in a central repository accessible to all system components.

Client-Server An architecture where services are provided by servers and accessed by clients over a network.

Pipe and Filter An architecture where data processing components (filters) are connected in a sequence (pipes) to process data.

Transaction Processing Systems Interactive systems that allow information in a database to be remotely accessed and modified by multiple users.

Language Processing Systems Systems used to translate texts from one language to another or carry out instructions specified in an input language.

Architectural Patterns Stylized descriptions of good design practices.

Application Architectures Generic architectures for different types of software systems.

0.9 Chapter 7: Design and Implementation

Design and Implementation The stages in the software engineering process where an executable software system is developed, involving design activities and realizing the design as a program.

Object-oriented Design A design method where a system is viewed as a collection of interacting objects, each with a defined role and responsibility.

Design Patterns Reusable solutions to common software design problems.

Singleton Pattern A design pattern that restricts the instantiation of a class to one object.

Factory Pattern A design pattern that provides an interface for creating objects, allowing subclasses to alter the type of objects that will be created.

Observer Pattern A design pattern where an object maintains a list of dependents and notifies them of state changes.

Open Source Development A software development model where the source code is made publicly available for use, modification, and distribution.

Error Handling The process of responding to and managing errors that occur in a software system.

Optimizing for Performance Techniques used to improve the efficiency and speed of a software system.

Managing Dependencies The process of handling the interdependencies between software components.

0.10 Chapter 8: Software Testing

Software Testing The process of executing a program with the intent of finding errors and ensuring that the software functions as intended.

Validation Testing Testing to demonstrate that the software meets its requirements.

Defect Testing Testing to discover faults or defects in the software.

Development Testing Testing carried out by the development team during development.

Release Testing Testing a complete version of the system before release.

User Testing Testing by users or potential users in their own environment.

Unit Testing Testing individual components in isolation to ensure they function correctly.

Automated Testing Using test automation frameworks to write and run tests without manual intervention.

System Testing Integrating components and testing the system against its requirements.

Alpha Testing Users work with the development team to test the software.

Beta Testing Software is released to users to experiment and report problems.

Acceptance Testing Customers test the system to decide whether it is ready for deployment.

Test-driven Development (TDD) Writing tests before code and using tests to drive development.

Code Coverage Ensuring that all code written has at least one associated test.

Regression Testing Testing the system to check that changes have not broken previously working code.

Verification Checking that the software conforms to its specification.

Validation Checking that the software does what the user requires.

0.11 Chapter 9: Software Evolution

Software Evolution The process of developing, maintaining, and updating software systems over time.

Corrective Maintenance Fixing defects in the software.

Adaptive Maintenance Modifying the software to work in a new or changed environment.

Perfective Maintenance Enhancing the software to improve performance or maintainability.

Preventive Maintenance Making changes to prevent future problems.

Legacy Systems Older software systems that remain critical to an organization.

Phase-out The stage where the software is still used but no further changes are made.

Servicing Changes made only to keep the software operational.

Software Maintenance The process of modifying a software system after it has been delivered.

0.12 Chapter 10: Dependable Systems

Dependable Systems Systems that are reliable, available, secure, and safe, reflecting the user's degree of trust in the system.

Availability The probability that the system will be operational and deliver useful services.

Reliability The probability that the system will correctly deliver services as expected.

Safety The likelihood that the system will not cause harm to people or the environment.

Security The likelihood that the system can resist accidental or deliberate intrusions.

Resilience The ability of the system to maintain continuity of critical services in the presence of disruptive events.

Redundancy Keeping multiple versions of critical components to ensure availability in case of failure.

Diversity Providing the same functionality in different ways to avoid common-mode failures.

Sociotechnical Systems Systems that include hardware, software, and people, situated within an organization.

Formal Methods Approaches to software development based on mathematical representation and analysis of software.

0.13 Chapter 13: Security Engineering

Confidentiality Ensuring that information is not accessed by unauthorized persons.

Integrity Ensuring that information is not altered by unauthorized persons.

Availability Ensuring that information is accessible to authorized users when needed.

Privacy Ensuring the confidentiality of personal data.

Authentication Verifying the identity of users or systems.

Authorization Granting permissions to users or systems.

Non-repudiation Ensuring that actions or transactions cannot be denied.

Infrastructure Security Security of the systems and networks providing infrastructure services.

Application Security Security of individual application systems.

Operational Security Secure operation and use of systems within an organization.

Interception Gaining unauthorized access to an asset.

Interruption Making a system or part of a system unavailable.

Modification Tampering with a system asset.

Fabrication Inserting false information into a system.

0.14 Chapter 24: Quality Management

Software Quality Management Ensuring that the required level of quality is achieved in a software product.

Quality Planning Setting out the desired product qualities, how these are assessed, and defining the standards to be used.

Software Quality Attributes Attributes such as safety, security, reliability, and resilience that determine software quality.

Process Standards Standards that define the processes to be followed during software development.

Product Standards Standards that apply to the software product being developed.

Quality Culture A culture where all team members are committed to achieving a high level of product quality.

ISO 9001 An international set of standards for developing quality management systems.

Software Metrics Quantitative measurements used to assess software processes and products.

Empirical Software Engineering Research area using data from software projects to form and validate hypotheses about software engineering methods and techniques.

0.15 ACM Code of Ethics and Professional Conduct

Contribute to Society The obligation to use skills for the benefit of society and its members.

Avoid Harm Taking actions to minimize negative consequences.

Honesty Being transparent and providing full disclosure.

Fairness Ensuring equality, tolerance, and justice.

Respect Privacy Protecting personal data and ensuring informed consent.

Honor Confidentiality Protecting sensitive information unless disclosure is legally or ethically justified.

Professional Competence Maintaining technical knowledge and skills.

Ethical Practice Adhering to ethical standards and principles in professional work.

Public Awareness Educating the public about computing technologies and their impacts.

Robust Security Designing and implementing systems to prevent unauthorized access and misuse.

Public Good Prioritizing the welfare of society in all professional activities.

Social Responsibilities Ensuring organizational policies align with ethical principles.

0.16 The Scrum Guide

Scrum A lightweight framework for generating value through adaptive solutions for complex problems.

Empiricism Making decisions based on what is observed.

Lean Thinking Reducing waste and focusing on essentials.

Commitment Dedicating oneself to achieving goals and supporting the team.

Focus Concentrating on the work of the Sprint to make the best possible progress.

Openness Being transparent about the work and challenges.

Respect Valuing each team member's capabilities and independence.

Courage Facing tough problems and doing the right thing.

Scrum Team A small team consisting of a Scrum Master, a Product Owner, and Developers.

Sprint A fixed-length event of one month or less to create consistency.

Product Backlog An ordered list of what is needed to improve the product.

Sprint Backlog The set of Product Backlog items selected for the Sprint and the plan for delivering them.

Increment A concrete stepping stone toward the Product Goal.

Definition of Done A formal description of the state of the Increment when it meets the quality measures required for the product.

0.17 WIPO Understanding Copyrights

Copyright A legal right that grants creators exclusive rights to their original works.

Economic Rights Rights to reproduce, distribute, and publicly perform a work.

Moral Rights Rights to claim authorship and oppose harmful modifications.

Limitations and Exceptions Permitted uses of copyrighted works without authorization.

Duration of Protection The period during which copyright protection is in effect, usually the life of the author plus a number of years.

0.18 WIPO Understanding Industrial Property

Industrial Property Includes patents, trademarks, industrial designs, and geographical indications.

Patent A legal right granted for an invention, giving the inventor exclusive rights for a limited period.

Trademark A sign that distinguishes the goods or services of one enterprise from those of others.

Industrial Design Protects the ornamental or aesthetic aspects of an article.

Geographical Indication Indicates that a product originates from a specific place and has qualities or a reputation due to that origin.