Name: **Student ID:**

Chapter 01: Overview

- **Software** + definition: computer programs & associated doc + types: generic products & customized products + essential attributes: maintainability, dependability, efficiency, acceptability
- **SE** + definition: Engineering discipline concerned with all aspects of software production + history: Originated from the "software crisis" in 1968 + importance: Producing reliable systems economically & quickly + Computer Science (theory & fundamentals while SE practicalities of dev & deliver software) & System Engineering (SysEng includes hard/software, process engineer vs. SE part of SysEng)
- **Software Process** + definition: sequence of activities leading to software production + activities: specification, dev, validation, evolution

Issues heterogeneity, business & social change, security & trust, scale **Diversity** because SE methods & tools used rely type of app, requirements of the customer, background of dev team **Types** stand-alone app (word), interactive transaction-based app (online banking) embedded control sys (car's anti-lock braking system), batch processing sys (electricity billing sys), entertainment sys (PS5), sys for modeling & simulation (flight simulators), data collection sys (GG form), systems of systems (airport) **Web Software Eng** + emerge: web services, cloud computing + concepts: reuse, incremental & agile dev, service-oriented sys, rich interfaces

- **SE Ethics** + professional responsibility: confidentiality, competence, intellectual property rights, computer misuse + ACM/IEEE code of ethics: Eight principles for professional conduct

Chapter 02: Software Processes

- **Software Process Models** + waterfall model: plan-driven, separate phases + incremental dev: interleaved specification, dev, & validation + reuse-oriented software engineering: system assembled from existing components + hybrid model
- **Process Activities** + specification: establish services & constraints + design & implementation: Converting specification into an exec sys + validation: verifying sys meets requirements + evolution: modifying software to meet changing needs
- **Coping with Change** + change anticipation: activities to anticipate possible changes + change tolerance: designing processes to accommodate changes at low cost + prototyping: developing quick versions to check requirements & design decisions + incremental delivery: delivering system increments for customer feedback
- **Boehm's spiral Model** + process represented as a spiral with each loop representing a phase + risks explicitly assessed & resolved throughout the process **Rational Unified Process (RUP)** + modern generic process derived from UML work + 4 phases: Inception, Elaboration, Construction, Transition + static workflows: Business modeling, Requirements, Analysis & design, Implementation, Testing, Deployment + good practices: Iterative development, requirements management, component-based architectures, visual modeling, quality verification, change control

Chapter 03: Project Management

- **Success criteria:** deliver software to customer on time + keep cost within budget + meet customer's expectations + maintain happy, well-functioning dev team **Factors** company size, software customers, software size, software type, organizational culture, software dev processes **Universal management activities** project planning, reporting, proposal writing, risk management, people management
- **Project planning** + essential for software project management + involves breaking down work, assigning tasks, & anticipating problems + occurs at proposal stage, project startup, & throughout dev + includes defining schedules, milestones, & deliverables
- **Risk management** + identifies potential risks & plans to minimize their impact + involves risk identification, analysis, planning, & monitoring + categorizes risks as project, product, or business risks + uses strategies like avoidance, minimization, & contingency planning
- **Managing people** + focuse consistency, respect, inclusion, & honesty + involves motivating team members based on Maslow's hierarchy of needs + consider different personality types: task-oriented, self-O & interaction-O
- **Teamwork** + essential for most software engineering projects + aims for group cohesiveness & team spirit + considers group composition, organization, & communication + balances different personality types for effective teamwork
- **Group Organization & Communication** + affects decision-making & information exchange + can be informal for small groups or hierarchical for large projects + emphasizes good communication for effective group working + considers factor like group size, structure, composition, work envi

Chapter 04: Requirement Engineering

- **Requirements engineering** + process of establishing customer's system requirements & constraints + includes user requirements (for customers) & system requirements (for developers)
- **Functional & Non-functional Requirements** + Functional: Services the system should provide + Non-functional: Constraints on services or functions (e.g., performance, security) + Domain requirements: Constraints from the operational domain
- **Requirements Engineering Processes** + elicitation, analysis, validation, management + in practice, RE is an iterative activity
- **Requirements Elicitation & Analysis** + involves working with stakeholders (end-users, manager, engineers…) to gather in4 + techniques include interview, ethnography, & scenarios/user stories + challenges include stakeholder conflicts & changing requirements
- **Requirements Specification** + writing down user & sys requirements + can use natural language, structured specifications, or formal method + should focus on what the system should do, not how it should do it
- **Requirements Validation** + ensures requirements define the system the customer wants + techniques include reviews, prototyping, & test-case generation + checks for validity, consistency, completeness, realism, & verifiability
- **Requirements Management** + process of managing changing requirements + involves tracking requirements, assessing change impacts, & establishing formal change processes + includes requirements identification, change management, traceability policies, & tool support

Chapter 05: System Modeling

- **System Modeling** + process of developing abstract models of system + uses graphical notations, often UML + helps understand system functionality & communicate with customers **System Perspectives** + External, interaction, structural, & behavioral perspectives **UML Diagram Types** Activity, use case, sequence, class, & state diagrams
- **Context models** + illustrate operational context & system boundaries + include process models using UML activity diagrams
- **Interaction models** + use case diagrams & sequence diagrams + model user interactions & system-to-system interactions
- **Structural models** + show sys organization & component relationship + include class diagram, generalization hierarchie, aggregation model
- **Behavioral models** + represent dynamic system behavior during execution + include data-driven models & event-driven models
- **State Machine models** + model system behavior in response to events + use state diagrams to show system states & transitions
- **Model-Driven Development** + write & implement software quickly, effectively, minimum cost + methodology known as model-driven software dev, model-driven engineering & model-driven architecture. + focuses on the construction of a software model

Chapter 08: UI Design

- **Design issues** + UI should designed to match skills, exp & expect of users + usually judge UI > func) + poorly UI cause user make catastrophic errors, never used **Human factor** limited short-term memory, make mistakes (alarm & messages increase stress), difference, different interaction preferences **Principals** user familiarity, consistency, minimal surprise, recoverability, user guidance, user diversity
- **UI design process** iterative involving liaison between user & designer. 3 core activities: user analysis, system prototyping, interface evaluation
- **User analysis** + described in terms that users & other designers can understand + scenarios describe typical episodes of use **Scenarios requirements** + user need help choose appropriate search + able select collection + able carry out search, request copy of relevant material **Analysis techniques** task analysis, interview + question, ethnography
- **UI prototyping** + aim to allow users to gain direct exp with the interface + no direct exp, impossible to judge the usability of an interface + 2 stages: paper prototypes, design then refined & sophisticated automated prototypes then developed. **Prototyping techniques** script-driven prototyping, visual programming, internet-based prototyping

Chapter 09: Software Testing

- **Introduction** + purpose: show program functionality & discover defects + process: execute program with artificial data, check results + goals: demonstrate requirements are met, discover incorrect behavior + *reveal presence of errors not absence* + verification vs validation: "are we building the product right?" vs "are we building the right product?"
- **Inspections** analysis of static system representation to discover problem (static verify), use tool-based doc & code **testing** excercising & observing product behaviour (dynamic verification). **BOTH are complementary, used during V&V process**
- **Dev testing** include all testing activities carried out by the team dev

1. Unit testing + defect testing (inheritance make unit testing more difficult) + automated whenever possible (frame JUnit) + automated test component: setup, call, assertion + test case should show component does what it's supposed to do, if defects, should reveal by test cases + 2 types: first type: normal input check component work as expected, second type use abnormal inputs check properly processed & not crash component + strategy: partition, guildline-based +*testing*

2. Component testing interface testing detect faults (interface errors) + interface types: param, shared mem, procedural, message passing +*interfaces* + interface errors: misuse, misunderstanding, timing error

3. Sys testing check compatible, interact correctly, transfer right data at right time + reusable components that have been separately developed & COTS integrated→complete system then tested + collective than individual process + use-case testing to identify system interactions, force them occur + testing policies to define the required system test coverage developed because exhaustive system testing is impossible

- **Test-driven development** write tests before code + develop code incrementally with tests + benefits: code coverage, regression testing, simplified debugging + process: identify functionality, write test, implement, refactor **Regression testing** + ensure changes don't break existing functionality + test "successfully" before change is committed.
- **Release testing** + purpose: convince supplier sys ready use + black-box testing based on sys specification + separate team conducts release testing + requirements-based testing + performance testing & stress testing
- **User testing** + alpha testing: users test at developer's site + beta testing: users test released ver + acceptance testing: customers test for deployment readiness + agile methods: Continuous user involvement in testing **V-model** linking dev stages with corresponding testing stages
- **Interface evaluation** + carried out to assess its suitability + full scale evaluation is expensive & impractical + evaluated against usability specification **Usability attributes** learnability, speed of operation, robustness, recoverability, adaptability

Chapter 10: Agile Software Development

- **Plan-driven Development** + based separate dev stage with the output produced at each of these stages planned in advance + waterfall, plan-driven, incremental dev + iteration **Agile Development** + specification, design, implementation & testing are inter-leaved + output decided through a process of negotiation during the software development.
- **Agile Methods** + rapid software dev to meet changing business needs +interleaved specification, design, & imple + frequent delivery of new ver for evaluation + minimal documentation, focus on working code + principle: customer involvement, incremental delivery, people over process, embracing change, maintaining simplicity
- **Extreme Programming (XP)** + influential agile method with frequent builds & releases + key practices: user stories for specification, refactoring, test-first development, pair programming + continuous integration & auto testing + customer involvement in dev & testing
- **Agile Project Management** + scrum: focus on managing iterative dev + scrum roles: dev team, product owner, scrummaster + sprint cycles: fixed-length iterations (2-4 weeks) + daily scrum meetings for team communication + product & sprint backlogs for task management
- **Scaling Agile Methods** + challenges in applying agile to large systems & organizations + issues: contractual agreements, maintenance, distributed teams + balancing agile & plan-driven approaches based on project factors + adapting agile for large systems: multi-team Scrum, product architects + organizational considerations: skill levels, quality procedures, cultural resistance

Chapter 06: Architectural design

- Design process identifying sub-systems making up a system + framework for subsystem communication & control → **architectural design**. Output is description of the **software architecture.**
- **Architectural design:** 1.early stage of system design 2.represents critical link between specification & design processes 3. carried out in parallel with some specification activities 4. Involves identifying major system components & their communications.
- **Pros of explicit architecture:** stakeholder communication, system analysis, large-scale reuse. → **when use?** facilitate discussion about the sys design, document an architecture that has been designed.
- **Architectural design decisions** creative process, affect the non-functional characteristics of the system. **Reuse** systems same domain, designed around one of more architectural patterns or 'styles' **Characteristic** performance, security, safety, availability, maintainability.
- **Architectural views** each architectural model only shows one view or perspective of the system. **Architectural model 4 + 1** logical, physical, development, process, use-case +*view*.
- **Architectural patterns** Model-View-Controller (MVC), Repository, Layered architecture, Client–server, Pipe & filter
- **App architectures** app system designed to meet an organizational need → generic application architecture (configured & adapted to create a system that meets specific requirements). **Use of app architectures** 1. starting point for architectural design 2. design checklist 3. way of organizing the dev team's work 4. means of assessing components for reuse 5. vocabulary for talking about application types
- **Examples of application types** data processing app, *transaction processing app*, event processing sys, *language processing sys*

Chapter 07: Object-Oriented Design

- **Design & implementation** is stage in the SE process at which an executable software system is developed + are invariably inter-leaved.
- **Build** (design process becomes concerned how use the configuration features) or **buy** off the-shelf systems (COTS) (cheap + fast than build)
- **Object-oriented design using the UML**

1. OO dev includes: analysis (OOA), design (OOD), programming (OOP)

2. Process stages + Define the context & modes of use of the system + Design the system architecture + Identify the principal system objects + Develop design models + Specify object interfaces

- **Design patterns** + patterns is a description of the problem & the essence of its solution + sufficiently abstract to reused in different settings + make use of OO characteristics (inheritance, polymorphism) **Elements** name, problem + solution description, consequences **Design problem** any design problem you facing may have associated pattern that can be applied
- **Implementation issues** + Reuse: should reuse existing code + Configuration management: keep track of different versions + Host-target development: should use a different computer to host instead of the same for host & execute. **Reuse lvl** abstraction (design patterns), object (libraries), component level (frameworks), system (COTS) + level

Reuse costs may high for large sys (also include adapting, configuring integrating elements with each other costs) **Configuration management** aim to support the system integration process(manage changes of code…) **activities** include version management, sys integration, problem tracking **Host-target** + most software developed on 1 computer (host) but runs on separate machines (target) + dev platform usually has different installed software than exec platform.

- **License** The GNU General Public License (GPL), The GNU Lesser General Public License (LGPL), The Berkley Standard Distribution License (BSD).