**L02: Introduction to Software Engineering**

**1. software :** Computer programs and associated documentation. developed for a particular customer or general market.

**Generic products**

Stand-alone systems, for any customer. The developer owns the specification, makes decisions on change.

ex) PC software, CAD software, software for specific markets

**Customized products**

commissioned by a specific customer to meet their own needs. customers own ~...

ex) embedded control systems, air traffic control software, traffic monitoring systems.

**2. attributes of good software**: functionality and performance, maintainable, dependable, usable.

**Maintainability**:  can evolve to meet the changing needs of customers.

**Dependability and security**:  not cause physical or economic damage of system failure. Malicious users cannot access or damage the system.

**Efficiency**: not wasteful of system resources(memory and processor cycles, **responsiveness**, processing time, memory utilization, etc.)

**Acceptability**: must be understandable, usable and compatible with other systems.

**3. software engineering**: engineering discipline that is concerned with all aspects of software production.

**SE**: concerned with all aspects of software production.

**E**: Using theories and methods to solve problems.

**software production**: technical process of development, project management and the development of tools, methods etc. to support software production.

**4. fundamental software engineering activities**: Software specification, software development, software validation and software evolution.

**Software specification**, establishing requirements and the constraints on the system.

**Software development**, software is designed and programmed.

**Software validation**, software is checked to ensure that it is what the customer requires.

**Software evolution**, modified to reflect changing customer and market requirements.

**5. Difference between SE and CS?**: CS focuses on theory and fundamentals; SE is concerned with the practicalities of developing and delivering useful software

**6. difference between SE and system engineering?**: System engineering is concerned with all aspects of computer-based systems development (hardware, software, process engineering)/ SE is a more general process.

**7. key challenges of SE?**: Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.

**8. costs of SE: 60% of development costs**, 40% are testing costs. For custom software, evolution costs > development costs.

**9. best SE techniques and methods**: different techniques are appropriate for different types of system. no method is better than another.

**Batch processing systems** : process data in large batches.

**Entertainment systems** : personal use, intended to entertain the user.

**Systems for modeling and simulation** : model physical processes or situations, include many, separate, interacting objects.

**Data collection systems** : collect data using a set of sensors and send them to other systems for processing.

**Systems of systems** : composed of a number of other software systems.

**10. differences from web to SE**: availability of software services, possibility of developing highly distributed service-based systems. advances in programming languages and software reuse.

**Cloud computing** : approach to the provision of computer services where applications run remotely on the ‘cloud’. Users do not buy software but pay according to use.

**Software reuse**:  approach for constructing web-based systems. assemble them from pre-existing software components and systems.

**Incremental and agile development**: should be developed and delivered incrementally. it's impractical to specify all the requirements in advance.

**Service-oriented systems**: software components are stand-alone web services.

**Rich interfaces** technologies - Angular, React, AJAX and HTML5

**Ethics**: Software engineers have responsibilities to the engineering profession and society. They should not simply be concerned with technical issues. Professional societies publish codes of conduct standards.

**L03-04: Software Processes**

**software process**: structured set of activities required to develop a software system. (Specification, Design and implementation, Validation, Evolution)

**Plan-driven processes**: all of the process activities are planned in advance and progress is measured against this plan.

**agile processes**: planning is incremental and it is easier to change the process to reflect changing customer requirements.

**software process model**: abstract representation of a process. +Products, Roles, Pre- and post-conditions

**The waterfall model(SPM)**

Maybe a plan-driven or agile model. Incremental development. Specification, development and validation are interleaved. using  reusable/configurable components.

phases: Requirements analysis and definition/System and software design/Implementation and unit testing/Integration and system testing/Operation and maintenance

**Bad**: difficulty of change after the process. has to be complete before moving onto the next phase. Inflexible , difficult to accept customer requirements.

-> appropriate when the requirements are well-understood and less changes / large systems' engineering projects where a system is developed at several sites. (Embedded System, Critical System,Large software systems)

**Incremental development(SMP)**

**Good**: reduce cost of changing. less work has to be redone. EZ to get customer feedback. Customers can check and comment on demonstrations. rapid delivery and deployment.

**Bad**: The process is invisible. not cost-effective to produce documents of every version of the system for managers. System structure degrades as new increments are added. changement corrupts its structure(if no refectoring). difficult and costly to incorporate further software changes.

**Integration and configuration(Reuse-oriented)(SMP)**

reuse software /for systems are integrated from existing components or application systems. (=COTS (Commercial-off-the-shelf) systems). maybe plan driven or agile.

reusable software: Stand-alone application systems (COTS), Collections of objects (.NET or J2EE), Web services (service standards and remote invocation)

**Good**: Reduced costs and risks,Faster delivery and deployment.

**Bad**: may not meet real needs of users, Loss of control over evolution of reused system elements.

**Process activities**: interleaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system. *waterfall model-in sequence, incremental -interleaved.*

**(Software specification) Requirements engineering process:**

Requirements elicitation and analysis- What do the system stakeholders require or expect from the system?

Requirements specification- Defining the requirements in detail

Requirements validation- Checking the validity of the requirements

**Software design and implementation:** The process of converting the system specification into an executable system. closely related and may be interleaved.

Software design: Design a software structure that realizes the specification;

Implementation: Translate this structure into an executable program;

**Design activities**

Architectural design: identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.

Database design: design the system data structures and how these are to be represented in a database.

Interface design: define the interfaces between system components.

Component selection and design: search for reusable components. If unavailable, you design how it will operate.

**System implementation**

Programming is an individual activity with no standard process.

Debugging is the activity of finding program faults and correcting these faults.

**Software validation**: show that a system conforms to its specification and meets the requirements of the system customer. checking and reviewing, testing.

Verification: Are we building the product RIGHT? (Design follows specification??)

Validation: Are we building the RIGHT product? (works as intended?)

System testing : executing the system with test cases that are derived from the specification of the real data to be processed by the system. most commonly used V & V activity.

**Testing stages**

Component testing: Individual components are tested independently;

System testing: Testing of the system as a whole.

Customer testing:Testing with customer data to check that the system meets the customer’s needs.

**Software evolution**: business circumstances change, software also evolves and changes.

**Reducing the costs of rework**

Change anticipation: anticipate possible changes before significant rework is required.

Change avoidance: how to avoid future changes?? (prototype system )

Change tolerance: designed so that changes can be accommodated at relatively low cost.

**Software prototyping**: initial version of a system to demonstrate and try out. developed quickly to check the customer’s requirements and the feasibility of design decisions. can be used in requirements engineering process, design processes, testing process.

Benefits: Improved system usability, A closer match to users’ real needs, Improved design quality, Improved maintainability, Reduced development effort.

Prototype development: based on rapid prototyping languages or tools, leaving out functionality( no error checking and recovery, Focus on functional-no reliability and security)

Throw-away prototypes: discarded after development. cuz of undocumented. structure is degraded through rapid change, not meeting normal organizational quality standards.

**Incremental delivery:** requirements broken down into increments and prioritized, each increment delivering part of the required functionality. Once started, the requirements are frozen ->  App, web, plug-in-based applications

Incremental development: Develop the increments, evaluate it before start next increment;

Incremental delivery: Deploy an increment to end-users; realistic evaluation of practical use.

Good: system functionality is available earlier, Early increments act as a prototype, highest priority system can receive the most testing.

Bad: Problematic when a new system replaces an existing system, cannot provide a set of basic facilities, Uncertainty(requirements are not defined in detail until an increment is to be implemented), organizations want complete specification and deliverables but cannot.

**Process improvement**: understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

The process maturity approach: focuses on improving process and project management and introducing good software engineering practice.

The agile approach: focuses on iterative development and the reduction of overheads in the software process. rapid delivery of functionality and responsiveness to changing customer requirements.

**Process improvement activities**

Process measurement : measure attributes of the software process or product. forms a baseline if process improvements have been effective.

Process analysis : assess current process, identify process weaknesses and bottlenecks.

Process change : address some of the identified process weaknesses. The cycle resumes to collect data about the effectiveness of the changes.

**L05: Project Management**

**Software project management**: ensuring that on time and on schedule, accepting requirements/ cuz of budget and schedule constraints by the organization.

**Success criteria**: Deliver at the agreed time/ Keep overall costs within budget/meets the customer’s expectations/ Maintain a coherent and well-functioning development team.

**Distinct characteristics of SE projects**: The product is intangible / Many software projects are 'one-off' projects / Software processes are variable and organization specific.

**Factors influencing project management**: Company size /Software customers Software size /Software type/Organizational culture /Software development processes

**Universal management activities:** Project planning / Risk management / People management  / Reporting / Proposal writing

**Risk management**: identifying risks and making plans to minimize their effect on a project. important because of the inherent uncertainties in software development.(loosely defined requirements, requirements changes, difficulties in estimating the time and resources required, and differences in individual skills.)

**Risk classification**: The type of risk (technical, organizational, ..)

what is affected by the risk: Project risks ->schedule or resources; Product risks->quality or performance of the software; Business risks->organization(ex. introducing a new project by a competitor);

**The risk management process**

Risk identification: Identify project, product and business risks; team of individual

checklist: Technology risks./ Organizational risks. /People risks. /Requirements risks. /Estimation risks.

Risk analysis: Assess probability and seriousness of each risk.

Risk planning: Draw up plans/develop a strategy to avoid or minimize/manage the effects of the risk;

*Avoidance strategies*-The probability that the risk will arise is reduced;

*Minimization strategies*-The impact of the risk on the project or product will be reduced;

*Contingency plans*-If the risk arises, contingency plans are plans to deal with that risk;

*What-if questions*-What if engineers are ill(Reorganize team)/budget cuts of 20%(briefing )/OSS sucks/software components go out(Replace ) …

Risk monitoring: Monitor the risks throughout the project; Assess each identified risks regularly to check probable, the effects of the risk have changed, at management progress meetings.

**People management factors**

Consistency: comparable way without favorites or discrimination.

Respect: different skills should be respected.

Inclusion: Involve all members and consider people’s views.

Honesty: always be honest about what is going well and what is going badly in a project.

**Motivate people** (Physiologic, Safety, Social, Esteem needs)

**Teamwork**: best size for SE is 4 to 6 members. (not more than 12 members), should consider motivation, Technical knowledge and ability, Personality

small groups-> minimize communication problems

cohesive group->Members think group>individuals, Loyal, establish its own quality standard. (consensus), Knowledge is shared, Refactoring and improvement is encouraged.

**Group communications affected** by Group size/structure/composition/ Physical environment/ Available communication channels

**L06: Project Planning**

**Project Planning:** breaking down the work/ assign these to project team members/ anticipate problems and prepare tentative solutions to those problems.

* Proposal planning: bidding for a contract, estimating cost(staff, hardware costs, software costs, etc)
* Project startup planning: know system requirements /do not have design or implementation information. Create a plan with enough detail and define project monitoring mechanisms.
* Development Planning: project schedule, cost-estimate and risks have to be regularly revised

**Software pricing**: cost of development + profit for the developer. Factors affecting(Contractual terms- allow to retain ownership of the source code / Cost estimate uncertainty-may increase /  Financial health - if hungry, cost down / Market opportunity - cost down for new market / Requirements volatility - cost down first, increase after)

**Plan-driven development:** development process is planned in detail. ‘traditional’ way of large projects.

good: issues and potential problems and dependencies are discovered before the project starts

bad: decisions have to be revised after all

Supplementary project plan: Configuration management plan, Deployment plan,Maintenance plan,Quality plan,Validation plan

serious problems-> risk mitigation, re-plan (renegotiating customer)

**Project scheduling:** deciding separate tasks + when and how, who, resources. Split project into tasks, Organize tasks concurrently, Minimize task dependencies. Bar charts is used.

process: identify activity -> identify dependencies -> estimate ~ -> allow people -> create project charts

\* Productivity != #of people / Adding people late makes communication overheads.

Project activities: basic planning element.  duration, effort estimate, deadline, end-point

Milestones : points in the schedule that can assess progress. need formal report for management

Deliverables: products delivered to the customer

**Estimation techniques:** two way -

Experience-based techniques: based on manager’s experience, people involved in the effort estimation, hard for new techniques.

Algorithmic cost modeling : formulaic approach. Effort = A  × Size\*B  × M

* A: organization-dependent constant (e.g. type of software)
* Size: code size(SLOC: the number of lines of source code )
* B: complexity of the SW or functionality(1~1.5)
* M: a factor that takes into account process, product, and development attributes.

bad: complex and difficult to use. so many attributes ,uncertainty.

**COCOMO cost modeling** : empirical model based on project experience. Well-documented, ‘independent’.

sub-models in COCOMO:

* Application composition model(composed from existing parts)
* Early design model. (requirement O / design X)
* Reuse model. (compute the effort of using reusable)
* Post-architecture model. (when architecture designed is available)

**L07-08: Agile Software Development**

**Agile methods**: Focus on code(>design). incremental, iterative approach. reduce overheads(minimize doc), respond quickly to changing requirements without excessive rework. high-quality code.

Agile development practices: User stories for system specification, Frequent releases of the software, Continuous software improvement ,Test-first development, Customer participation in the development team.

principle: Customer involvement (Customers should be involved in the process), Incremental delivery(developed in increments in each increment), People, not process (skills of the development team. own ways of working), Embrace change(Expect change, design for it)

Maintain simplicity (eliminate complexity)

**Agile development techniques**:

Extreme programming: ‘extreme’ approach to iterative development. New versions per day, Increments for customers every 2 weeks, All tests for every build… practice:

* **Incremental planning**(recorded Requirements on story cards, included in a release by time priority. break these stories into ‘Tasks’)
* Small releases(minimal set is developed first. Releases often, incrementally add functionality)
* Simple design (for current requirements and no more)
* **Test-first development**(using unit test framework to write tests before start implementation. **TDD:** clarifies the requirements. Tests are written as programs. testing framework. All tests are run automatically when new functionality is added.)
* **Refactoring** (keeps the code simple and maintainable. understandability, reduces documentation. howto: remove duplicate, reorganize class hierarchy, Tidying up and renaming, replacement inline code with methods of the library.)
* **Pair programming**(Developers work in pairs. informal review process, encourages refactoring, sharing of knowledge: reduces the risks when team members leave)
* Collective ownership(no islands of expertise develop ,everyone takes responsibility, Anyone can change anything.)
* Continuous integration (integrated into the whole system when task is done, run unit tests)
* Sustainable pace (no overtime)
* On-site customer(representative of the end-user should be available full time)

**Agile project management**: for on time/ budget. plan-drive: draw up a plan for what , when, who.

Scrum: agile method that provides a project management framework. (or daily meeting of it) three phases:

* initial phase: outline planning, establish the objectives and design SW architecture.
* sprint cycles: fixed time period(2-4 week). Each cycle develops an increment of the system.
* The project closure phase: wraps up the project, completes required documentation

Development team: group of SW developers, under 7 people.

Potentially shippable product increment: in a finished state and no further work

Product backlog: TODO list.

Product owner: who identify features or requirements, prioritize ,review the backlog... can be a customer or product manager or other

Scrum master: guides the team in the effective use of Scrum. protect the team from external distractions.

Velocity: how much backlog that a team can cover in a single sprint.

good: The product is a set of manageable,understandable chunks. team communication is improved. on-time delivery and feedback. Trust between customers and developers

**Scaling agile methods**:

‘Scaling up’ : using agile methods for developing large software systems.

‘Scaling out’ : how agile methods can be introduced into existing large organizations.

agile fundamentals: Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.

not good for large systems(no up-front design, conflict with doc and organizational practice.

good for a small team that can communicate informally. short-lifetime and no external regulation. skilled members.

**L09: Requirement Engineering**

Requirement: set out what the system should do and define constraints on its operation and implementation.

**Functional and non-functional requirements** :

Functional requirements: statements of the services that the system must provide or are descriptions of how some computations must be carried out.

Non-functional requirements :constrain the system and process, relate to the emergent properties,, apply to the whole system.

**Requirements engineering processes**: iterative process (requirements elicitation, specification and validation)

**Requirements elicitation**: iterative process, spiral of activities(requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation)

* using interviews and ethnography. User stories and scenarios, facilitate discussions...

**Requirements specification**: process of formally documenting the user /system requirements.

requirement documentation: agreed statement of the system requirements, for both customers and developers.

**Requirements validation**: process of checking the validity, consistency, completeness, realism and verifiability of requirement.

**Requirements change**: Requirements management process managing and controlling the requirement changes.

**L10: System Modeling**

Model: abstract view of a system, ignores system details. Complementary system models show the system’s context, interactions, structure and behavior.

**Context models**: show how a system is positioned in an environment with other systems and processes.

**Interaction models**:

case diagrams/sequence diagrams : describe the interactions between users(external actors) and systems.

**Structural models**: show the organization and architecture of a system.

Class diagrams: define the static structure of classes in a system and their associations.

**Behavioral models**:

Behavioral models : describe the dynamic behavior of an executing system. This behavior can be modeled from the perspective of the data , or events

Activity diagrams : model the processing of data, where each activity represents one process step.

State diagrams: model a system’s behavior in response to internal or external events.

**Model-driven engineering**: system is represented as a set of models that can be transformed to code

**L11: Architectural Design**

A software architecture : description of how a software system is organized.

**Architectural design decisions**:

type of application, distribution of the system, the architectural styles

**Architectural views**:

can be documented from perspectives of... (conceptual view, a logical view, a process view, and a development view)

**Architectural patterns**:

Architectural patterns : reusing knowledge about generic system architectures. describe the architecture, explain when it may be used and advantages/disadvantages.

**Application architectures**:

application architecture model: to understand and compare applications, validate application system designs and assess large-scale components for reuse.

Transaction processing systems : interactive systems, many user can remotely accessed and modified to DB

Language processing systems: translate texts and to carry out the instructions. translator and an abstract machine to execute.

**L12: Design and Implementation**

**Object-oriented design using the UML** :

OOP process: design the system architecture, identify objects, describe design using object models, document the component interfaces. use UML interface stereotypes to define interfaces.

***Understand*** *and define the context and external interactions with the system;*

system context model: demonstrates the other systems in the same environment

interaction model: dynamic model that shows how the system interacts with its environment as it is used.

***Design*** *the system architecture;*

***Identify*** *the principal system objects;*

***Develop*** *design models;*

static models (class models, generalization models, association models)

dynamic models (sequence models, state machine models).

* Sequence models : show the sequence of interactions.
* State machine models : show how objects change their state in response to events

***Specify*** *object interfaces.*

**Design patterns**:way of reusing abstract knowledge about a problem and its solution. use of object-oriented characteristics such as inheritance and polymorphism.

**Implementation issues**:

reuse: consider the possibility of reusing existing software. abstraction level(concept)/object level(lib)/The component level / system level (entire)

* cost:looking for, buying, adapting and configuring, integrating

Configuration management: process of managing changes of SW system. Version management, System integration, Problem tracking, Release management

host-target development: develop on IDE on a host machine, transferred to a target machine for execution.

**L13-14: Open source software development**

**Open source development**: approach where source code is published and volunteers are invited to participate in the development process.( Linux OS, Java, Apache , mySQL …)

good: develop more cheaply, quickly, and create a community of users of SW.

**License**: GNU General Public License (**GPL**) - make it OSS to use OSS / GNU Lesser General Public License (**LGPL**) - no need to publish the source of components. / Berkeley Standard Distribution (**BSD**) License. - no need to re-publish any changes. can use OSS in proprietary systems.

**Contribute**: Mostly changes (Patches), Documents, Reporting issues/bugs, Testing/validation