Why Software Engineering was created?
To combat the Software Crisis:

- projects running overtime, over budget, unmaintainable or cancelled
- Inefficient or low quality software
- Software often not meeting requirements
- Software never being delivered

What causes software failure?

- unrealistic project goals scope
- badly defined requirements
- changing requirements
- poor communication
- poor project managements

Process of creating software:
Reqs gathering -> Design -> implementation -> testing -> evolution -> reqs gathering

Define software engineering:
It is the study and application of engineering to the design, development and maintenance of software. It is concerned with the developing and maintaining software systems that behave reliably and efficiency and are affordable to develop and maintain and satisfy all requirements that customers have defined it for them.

Describe waterfall process:
Requirements engineering -> Software design -> Implementation -> Testing -> Evolution

- Macro development process:

serve as controlling framework for the entire software development lifecycle. Measure in weeks to months in teams effort.
E.G Waterfall, agile, iterative, model driven development

- Micro development process:

Process within processes, serve as a controlling framework for individual development activities, measured in term of days and controlled by macro process.
e.g. Requirement engineering, design, implementation, debugging, testing, evolution

Define Requirement engineering:
It is the concern of finding out what the customers wants and translate that want into a system design.

Requirement engineering process:
Feasibility study -> reqs elicitation -> reqs spec -> reqs validation -> reqs evolution
Feasibility study: decide whether or not the proposed system is worthwhile.
Requirement elicitation: Find out what system stakeholders require from the system.

Requirements Specification: define the requirements in detail
Requirements validation: check if the requirements are correct

Functional requirements: relate directly to WHAT function the system should provide. e.g. “search for product online”
FR statement usually is written as: <The system should **do <requirement>>**

Non-functional requirements: Constraints on the system and define how the system should work as a whole. e.g. usability, security, performance. and usually are also called Software Qualities.
NFRs is usually written as: <The System should **be/have** <requirements>>

Prioritising requirements: Using MoSCoW (Must, should, could, won’t this time (would))

Requirements elicitation: Involves software analysts working with customers to identify stakeholders, business processes, FRs and NFRs and to priorities and record this information.

Business process: a set of structured tasks that gives rise to a service or product.

Requirement elicitation techniques:
Process: Interviews, workshop, focus group and observation
Representation: Scenario, use cases, models, prototype, protocols, and formal specifications

Closed interviews: a pre-defined set of questions are answered
Open interviews: where there is no pre-defined agenda and a range of issues are explored with the stakeholders.

Interviews aren’t good for understanding domain requirements:
requirements engineers cannot understand specific domain terminology; some domain knowledge is so familiar that people find it hard to articulate or think it isn’t worth mentioning.

Requirement elicitation scenarios:
- As-is scenario: story or example of events taken from real world
- To-be scenario: future vision of a designed system
- Use case scenario: a way in which a system is used

Requirement specification: activity in RE or a product of requirements specification

Use cases are a scenario-based techniques, focus on identifying actors and all possible interaction with the system, comes in two forms:
- use case diagrams (UCD): is a UML representation of a use case
- use case specifications (UCS): is a structured description of a use case

Type of UML Models:

- Functions model
    - use case diagram: describe the functional behaviour of the system as seen by the user
    - activity diagram: describe the dynamic of the system, in particular the workflow
- Structural model
    - class diagram: Describe the static structure of the system: objects, attributes and association
- Behaviour model
    - Interaction diagram: are objet diagrams that specify interactions between objects
        - split into two parts:
            - Sequence diagrams:
            - communication diagrams:
    - state diagram: describe the internal behaviour of a single object

Software development process is all about realising use cases.
Use cases are realised by domain classes. domain classes are refined into system classes, which are used for implementation of the system.

Domain classes are conceptual, system class implements them.

Objects vs classes:
Object: have behaviour and they are instances of classes.
Class: have only the structure

Kind of software processes:

- Sequential processes: E.g. waterfall model
- Iterative process: E.g. Unified Process
- Aigle process: E.g. SCRUM

Sequential process: Strong emphasis on well defined complete phases, each phase has its own deliverable, each phase deal with the complete system, the feedback may cause adjusting to previous phase.

Iterative process: Iterate over well-defined cycles, each cycle has short phase and each phase has well defined activities, each iteration produces a product release

Agile process: Iterate over short cycle, each iteration produce part of the software, software presented to customer after each iteration, requirements changes checked with customer after each iteration.

Waterfall model:

- Pros:

Well-defined milestones and deliverables make budgeting and planning easier.
Everything is well documented so problems can easily be traced back
promotes specialisations

- Cons:
  later a problem is found the more it cost to fix. HAS to get it right from the start.
- Common problems:
  - deliverables are often not produced on time or are rushed.
  - analysis paralysis often occurs
  - specialisation can cause poor communication
  - deliverables does not meet the users' needs, often 45% of features not used
  - usually late and grow out of budget
- Why not to use it:
  - You won't get requirements right from first time
  - User won't see anything working until very end
  - requirements might change, and what user really want is different

Agile model:

- Pros:
  Strong emphasis on testing. frequent interactions with customers, done in small teams with little specialisation.
  Cope much better with requirement changes. Code quality is good
- Cons:
  Requires competence and confident from all developers. Long-term planning is difficult.
  lack of documentation may cause problems later. not suitable for large project

Extreme programming (XP) form of agile method:

- requirements are determined from user stories
- custom representative always available
- unit test are written BEFOre the code to be tested
- programming in pairs, one code the other reviews, swapping frequently
- simplicity of design and refactor when needed and when possible

Unified process phases:

- Inception: define scope of projects including initial feasibility and project go-ahead
- Elaboration: plan project, specify features and baseline architecture
- Construction: build the product
- Transition: transition the product into end user community, acceptance testing and maintenance

---

Nine principles of design:

- Simple and natural dialogue: no irrelevant or rarely used information,
- Speak the user's language: use concepts from user's world, don't use system specific-engineering terms

- Minimise user memory load: don’t make user remember thing one action to another.
- Be consistent: action sequence learned in one part of system should be used in other parts
- Provide feedback
- provide clearly marked exits.
- provide short cuts.
- good error messages
- prevent errors