- **Systems development life cycle:**
  - **Old version:** System concept → Implementation → Maintenance → Disposition.
  - **New version (DevOps):** No final version of SW; Continuous development.
    - **Dev** part includes: design, code, check-in, build, test and quality assessment.
    - **Ops** part includes: monitoring, operating and deployment.
- **Project management triangle:**
  - The value is the main element in the development decision.
  - The quality of work is constrained by the project's budget (cost), deadlines (time) and features (scope).
- **Software development paradigms and models:**
  - **Waterfall model (Herbert D. Benington, 1956):**
    - The breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one.
    - **The most well-known. Teams structure is functional.**
    - **Functional/Cross-Functional team:**
      - Functional structure divides the organization into departments based on their function.
      - Cross-functional team is a group of people with different functional expertise working toward a common goal.

Advantages	Disadvantages
• Easy to understand and implement. • Widely used and known • Reinforces good habits: define-before- design, design-before-code • Identifies deliverables and milestones • Document-driven • Works well on mature products	• Idealized, doesn't match reality well. • Doesn't reflect iterative nature of exploratory development. • Unrealistic to expect accurate requirements so early in project • The software is delivered late in project, delaying discovery of serious errors. • Difficult to integrate risk management • Significant administrative overhead, costly for small teams and projects. • Difficult and expensive to make changes to documents (swimming upstream).

  - **V-shape model:**
    - An extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage.

Advantages	Disadvantages
• Emphasize planning for verification and validation of the product in early stages of product development • Each deliverable must be testable • Project management can track progress by milestones	• Does not easily handle concurrent events • Does not handle iterations or phases • Does not easily handle dynamic changes in requirements • Does not contain risk analysis activities

- **Spiral model:**
  - An iterative model which includes risk analysis and risk management.
  - **Key Idea:** On each iteration identify and solve the subproblems with the highest risk.

Advantages	Disadvantages
• Realism: the model accurately reflects the iterative nature of software development on projects with unclear requirements • Flexibility: incorporates the advantages of the waterfall and rapid prototyping methods • Comprehensive model decreases risk • Good project visibility.	• Needs technical expertise in risk analysis to really work • Difficult to understand and apply, hence not so widely used • Needs competent professional management. • High administrative overhead.

- **Incremental model:**
  - A method of software development where the product is designed, implemented and tested incrementally (a little more is added each time) until the product is finished.
  - It involves both development and maintenance.
  - It combines the elements of the waterfall model with the iterative philosophy of prototyping.
  - **Disadvantages:**
    - Due to the "contractual basis" and without regular communication with the client, it is not possible to receive feedback on an ongoing basis.
    - Changes in the project are also not desirable.
    - The client receives the finished product only at the end of the development period.
- **Agile Software development:**
  - **Manifesto**: while there is value in the items on the right, we value the items on the left more.
    - Individuals and interactions **over** processes and tools
    - Working software **over** comprehensive documentation
    - Customer collaboration **over** contract negotiation
    - Responding to change **over** following a plan
  - **Teams** overlap, that is they share responsibility and ownership of artifacts.
  - **Test Driven Development (TDD):** is the idea of having the developers to write unit tests along with the process of development and use them regularly.
  - Other testing methods like Integration, Regression, Load, Penetration, Mutation and Soak testing are done by the testing department.
  - Any agile team should have a **Customer Representative (Product Owner in Scrum)**; This person is agreed by stakeholders to act on their behalf and should be available for developers to answer questions throughout the iteration.

- **Software methodologies and frameworks:**
  - **Lean concept:**
    - Divide the work activities into the value-adding ones, and the non-value-adding ones
    - Divide them again into the ones that are needed and the ones that are not needed.
      - The value-adding, needed activities should be given the priority and support.
      - The non-value-adding, needed activities needs reorganization and focus.
      - The non-value-adding, not needed activities can be removed.

- **Lack of communication between different people in development cycle leads to having the product totally different from what the client expected**
    - This process is similar to the **Chinese whispering game**, in which the players form a line, the first player comes up with a message and whispers it to the ear of the second person in the line. The second player repeats the message to the third player, and so on. When the last player is reached, they announce the message they heard to the entire group.
    - The first person then compares the original message with the final version, errors typically accumulate in the retellings, so the statement announced by the last player differs significantly from that of the first player.
- **Agile evolution:**
    - **Spiral (1986) => Scrum (1993) => Team Software Process (1996) => Extreme programming (1996) => Agile Manifesto (2001) => Agile Unified Process (2005) => Disciplined Agile Delivery => Scaled Agile Framework (2011) => Large-Scale Scrum (2013)**
        - **DAD** enables teams to make simplified process decisions around incremental and iterative solution delivery.
        - **SAFe** is the most common approach to scaling agile practices, it promotes alignment, collaboration, and delivery across large numbers of agile teams.
        - **LeSS** is a product development framework that extends Scrum with scaling rules and guidelines without losing the original purposes of Scrum.
    - Projects following Agile tend to be more challenged than waterfall, although they have smaller chance to fail, and higher change to succeed.
- **Dark side of Agile:**
    - A misunderstanding of the Agile manifesto, interpreting it as "the only way of SWD" and thinking of the items on the right as useless.
    - This advocates coding without discipline, planning and documenting.
    - Agile manifesto has nothing to do with this false translation and they describe it as "an attempt to legitimize hacker behavior"
- **Half-Arsed (fake) agile:**
    - Another misunderstanding of the Agile manifesto, they are companies who pretend to follow agile, while not letting go of the items on the right, they are ready to follow agile until it's waterfall.
- **Comparison between different type of programmers**

	Traditional Software Engineer	**Cowboy Coder**	**Agile Software Engineer**
Analysis and design	Completes all the analysis and design before proceeding to code.	Doesn't do any analysis or design and rushes to code without thinking.	Does only the necessary analysis and design then starts working.
Documentation	Writes all the comprehensive documentation in a complete and pervasive way.	Doesn't write any documentation.	Writes a clean, self-documenting code, and only the documentation that is needed by people.
Deadlines	Especially close to the deadline, he works like crazy to get the project done.	Only close to the deadline, he works like crazy to get the project done.	Especially close to the deadline, He works no more than 40h per week, keeping a constant pace and a fresh mind.
Code	Doesn't like others to modify his code.	Doesn't like others to modify his code.	The code is of the team and everyone is allowed to

			modify it also extensively, provided the tests keep running.
Integration	Integrates the system only at the end, he defines precise interaction protocols and documents them with maximal details.	Integrates the system only at the end, he doesn't know exactly what to do.	Integrates the system at least daily, to make sure there will be no problems in production.
Customer collaboration	Hides the product from the customer unless it's in a clean pre-done state, he contacts the customer only by formal balanced appointments.	Tries to hide the product from the customer unless it's completely finished.	The customer should be constantly exposed to the product being built and to the development team, and, whenever possible, have a representative on site.
Attitude to the code	If the code is working, he doesn't modify it.	Even if the code is not working, he doesn't modify it, he tries to hide it.	Even if the code is working, constantly refactor I, use the test cases to ensure you do not introduce an undesired bug.
Planning	Plans everything well in advance thinking that there will be no need to change.	Doesn't plan anything and tries not to change.	Plans everything that he can reasonably foresee and is always ready to change; changes occur naturally in software projects.

- **Common problems:**
  - Product owner knowing nothing about the product, while customer knows.
  - Product manager forces the team to do some activities like meetings, retrospective, etc. without telling them the benefits.
  - Teams trying to reinvent the wheel, by using a hybrid development model to work with.
- **Semantic Gap:**
  - Developers and domain experts use different language and terminology to describe the same thing, leading to communication problems
  - Solution is to use a ubiquitous language (creating a common dictionary by asking other about words that are ambiguous) that connects all parts of the design and evolves during the whole development process.
- **Semantic memory:** an abstract network of concepts that affords rich inferential ability.
- **Episodic memory:** specific events from which inference is difficult.
- **Types of questions to ask during stakeholder interview:**
  - Opening questions
  - Introductory questions
  - Transition questions
  - Key questions
  - Ending questions.