HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

System Analysis and Design

IT3120E

ONE LOVE. ONE FUTURE.

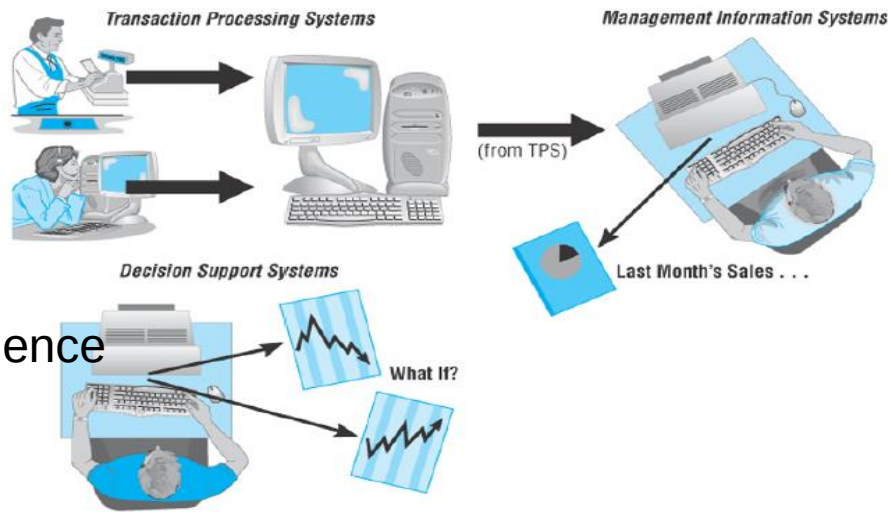
Chapter 1: *General concepts*

Course syllabus

- 1.1 Introduction
- 1.2. Systems development life cycle (SDLC)
- 1.3. Common systems development methodologies

1.1 Introduction

- Information Systems
  - (mostly) mean software in this course: mobile apps, onsite applications, SaaS (Software as a Service) or web development, etc.
  - **essential** part of almost all businesses to support their activities
- Type of Information systems:
  - Transaction processing system
    - Improve transaction processing by increasing speed, efficiency, and simplicity processes
  - Management information systems
    - Provide useful information for management work
  - Decision support system
    - Compare different solutions and recommend an appropriate one
  - Production automation system
  - Office automation system
    - Support users in office activities
  - Expert system and artificial intelligence
    - Store and use the knowledge of experts effectively
  - etc.

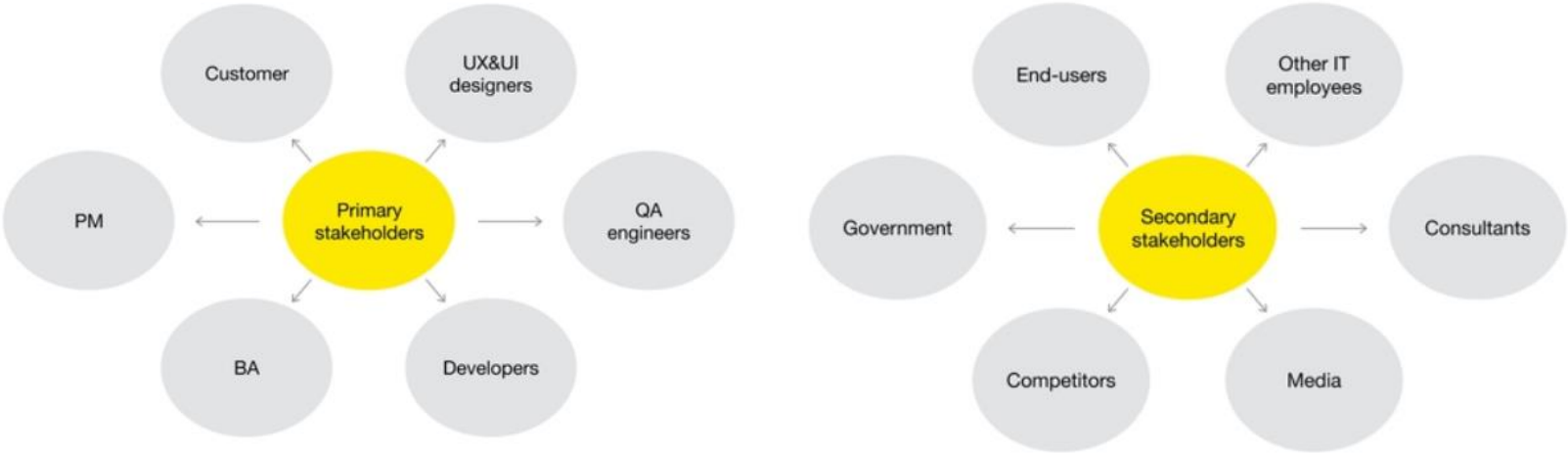


1.1 Introduction

- Software project: **creates and maintains** applications, frameworks, or other software components

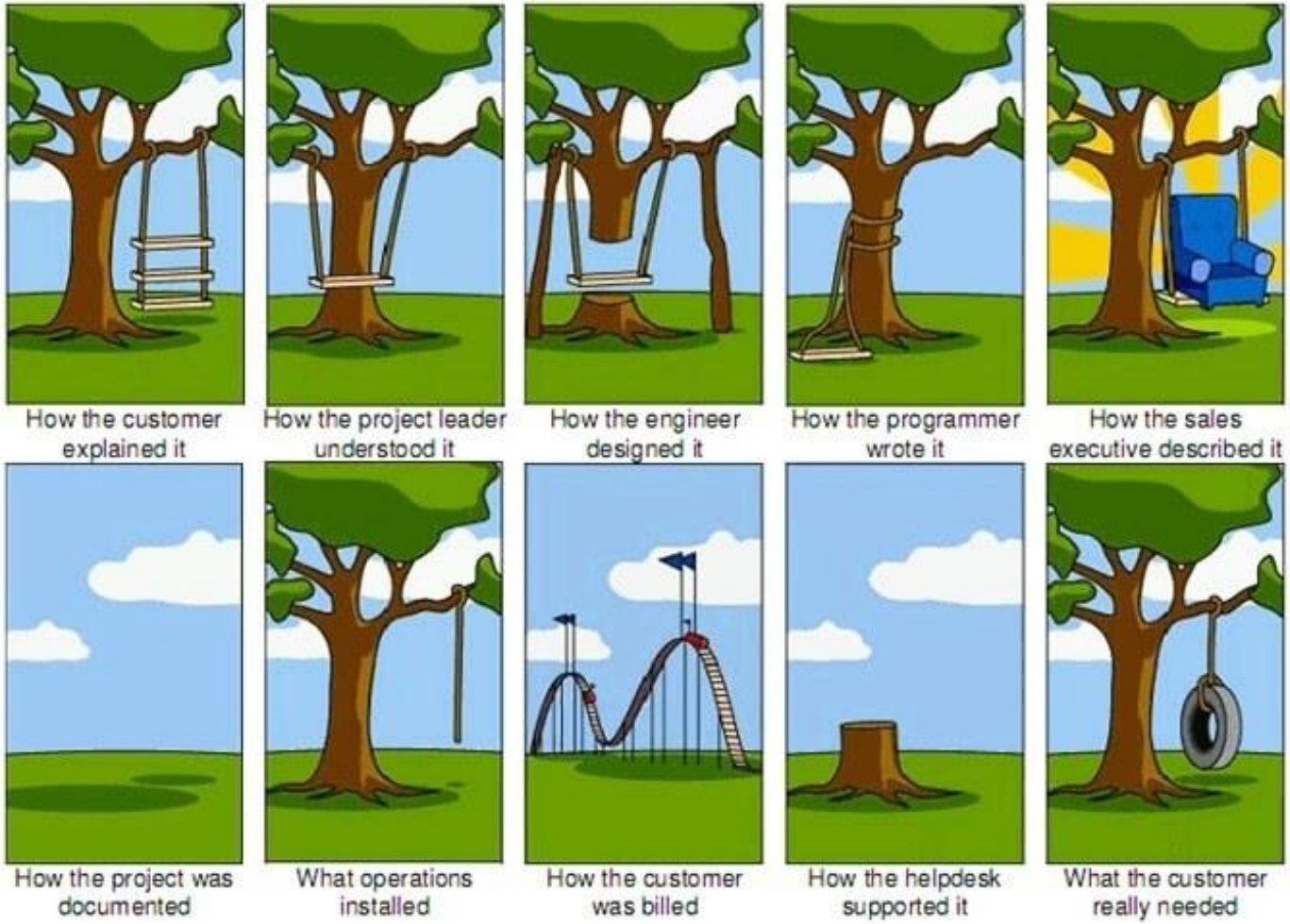
- Software Development Process: process of conceiving, specifying, designing, programming, documenting, testing, and bug fixing involved in a software development project

- Stakeholders in a software project



1.1 Introduction

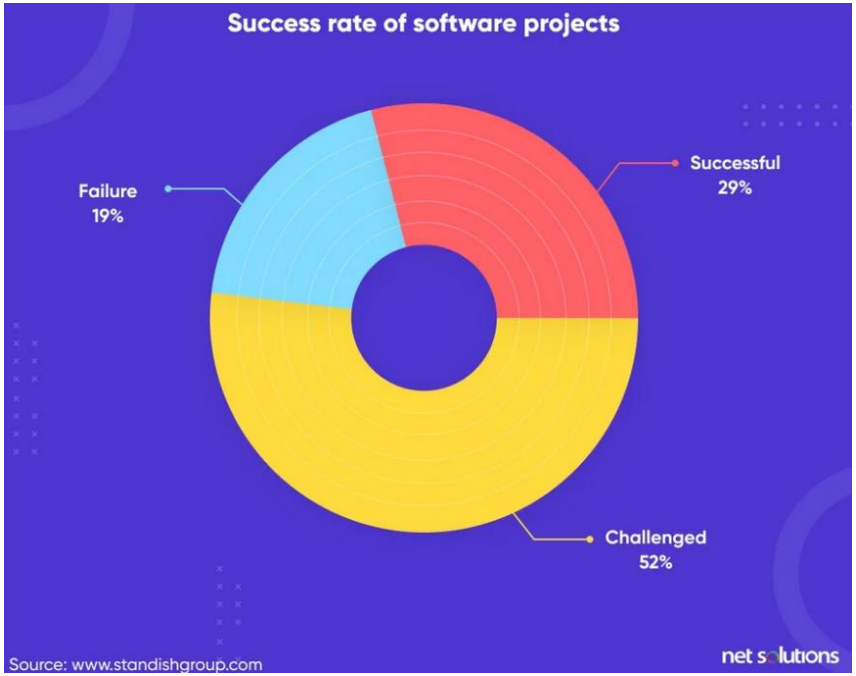
- But, the success rate of software projects is NOT high !



https://lansa.com/blog/rapid-application-development/ambiguity-in-requirements/

1.1 Introduction

- But, the success rate of software projects is NOT high !
  - "Two-thirds of all software projects end in partial or total failure"[1]
  - "70% of organizations experienced at least one software project failure in the previous 12 months"[2]



2021 report studied on 50,000 projects

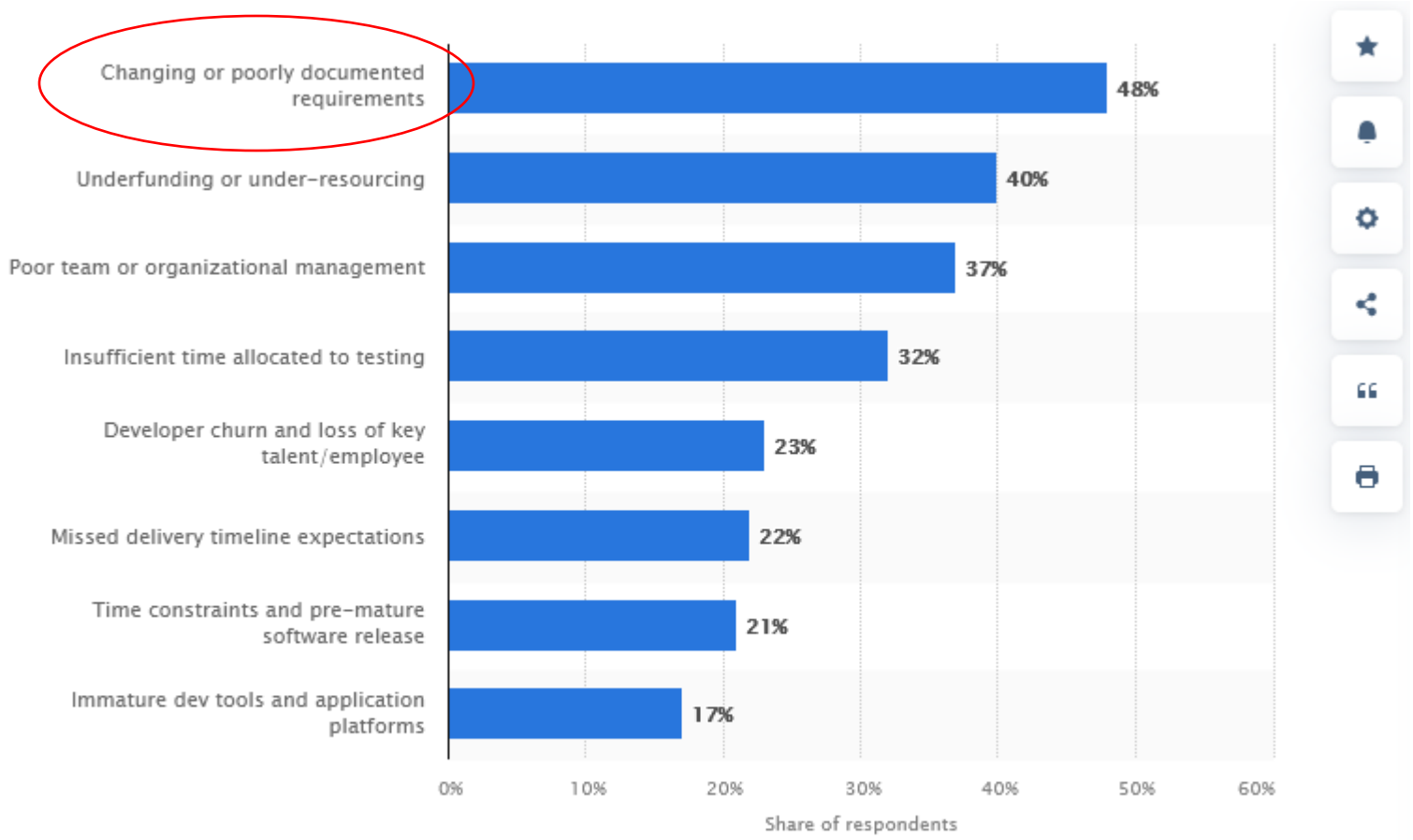
1. https://www.netsolutions.com/insights/23-reasons-why-software-projects-fail-with-solutions/
2. https://www.inapps.net/why-software-projects-fail-6-common-reasons-lead-software-project-failure/

1.1 Introduction

- Reasons of software project failure:
  - Lack of budgeting controls, Unrealistic time and budget expectations, No feasibility analysis etc.
  - Inaccurate requirements, Poor communication between customer and developer, Frequently changing project objectives
  - Risks that no one could foresee
  - Poor use of any new technology
  - Poor project manager
  - Delays caused by the project team members
  - etc.

1.1 Introduction

- Reasons of software project failure:



https://www.statista.com/statistics/627648/worldwide-software-developer-survey-project-failure/ © Statista 2023

Reasons of software project failure

- **Unclear requirements**
  - Customers need to specify and express what they needs clearly.
  - Customers are often so focused on the result at the start of a project that they overlook minor but critical to the overall product.
  - Insufficient requirements management
- **Poor communication**
  - Share knowledge and exchange information and ideas
  - Hard communication between the customer and the developer, developer and developer, among stakeholders.
  - Ambiguous communication
- **No end-user involvement**
  - To ensure that the solution is user-friendly
  - Try to think like a user
- **Poor testing**
  - Tests are performed in haste and thus inaccuracy so bugs that are not always detected right away

Reasons of software project failure

- And the cost..

The iceberg model – CISQ 2018 report

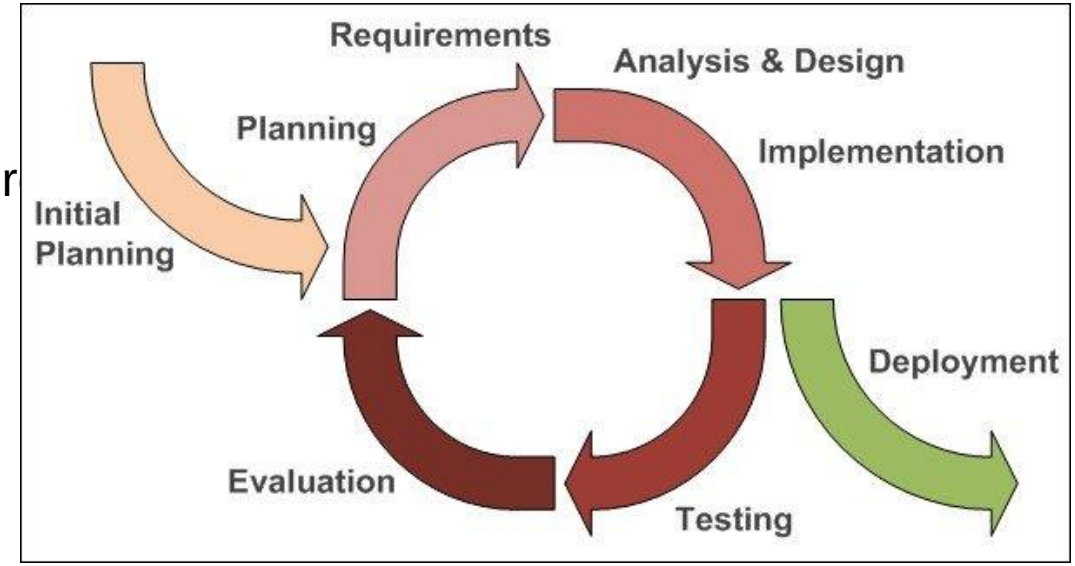
Solution: 6 Best Practices

- => 6 Best Practices:
  - Develop iteratively
  - Manage requirements
  - Use component architecture
  - Model software visually
  - Verify quality continuously
  - Manage changing

Solution: 6 Best Practices

- => 6 Best Practices:
  - **Develop iteratively**
  - Manage requirements
  - Use component architectur
  - Model software visually
  - Verify quality continuously
  - Manage changing

Software Development Process

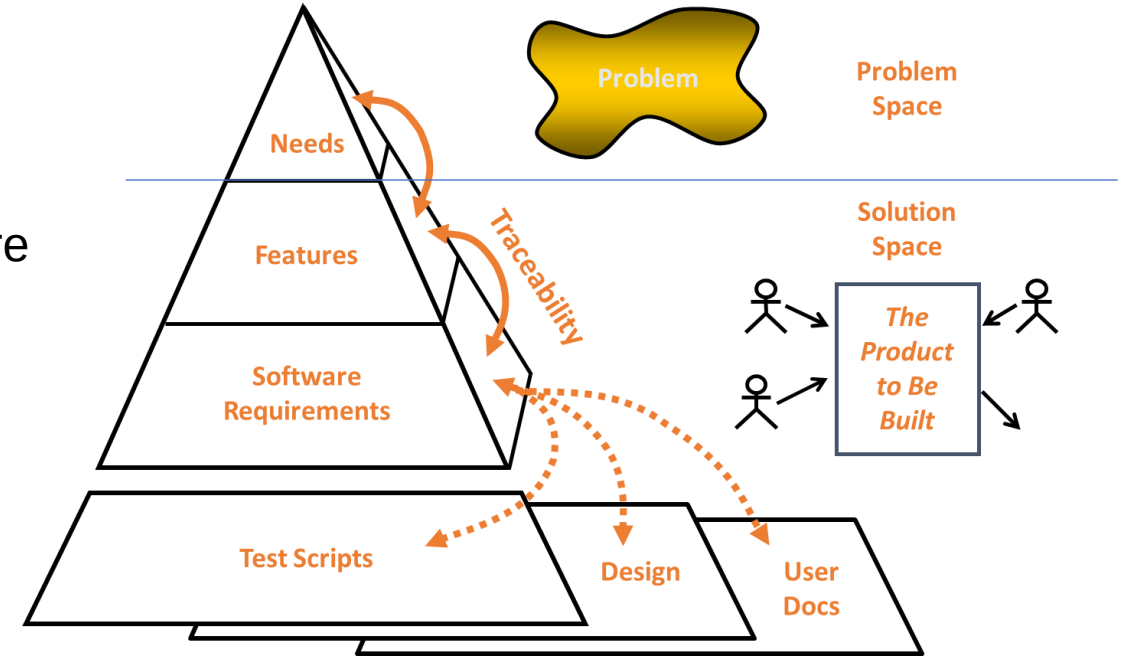


Each iteration results in an executable release

+ Enable early user feedback
+ Testing and integration are continuous
+ Objective milestones provide short term focus

Solution: 6 Best Practices

- => 6 Best Practices:
  - Develop iteratively
  - **Manage requirements**
  - Use component architecture
  - Model software visually
  - Verify quality continuously
  - Manage changing

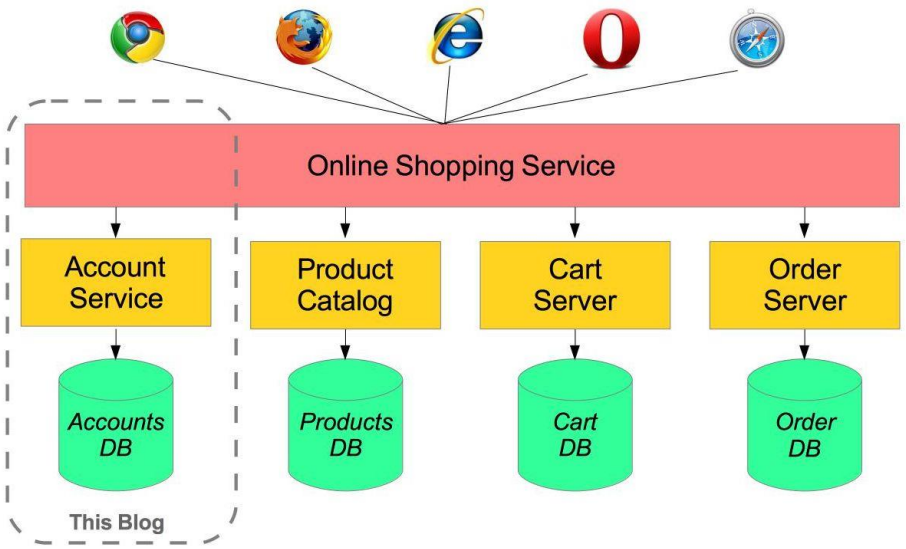


Making sure you
solve the right problem
build the right system !

+ Requirements are dynamic - expect them to change during software development
+ Gain agreement with user on what the system should do and not how
+ Maintain (organizing, documenting, managing) the changing requirements of a software application

Solution: 6 Best Practices

- => 6 Best Practices:
  - Develop iteratively
  - Manage requirements
  - **Use component architecture**
  - Model software visually
  - Verify quality continuously
  - Manage changing

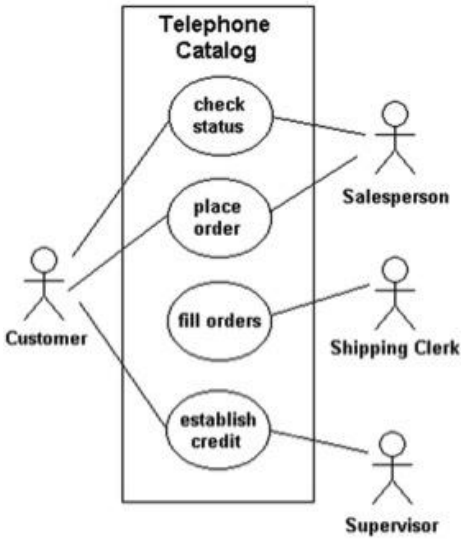


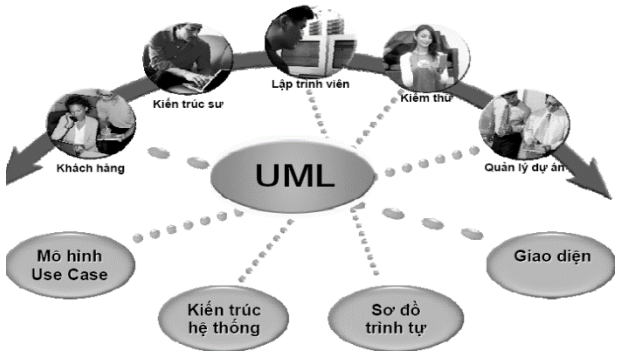
+ Enables reuse or customize components
+ Improves maintainability and extensibility
+ Encapsulates system dependencies
+ Promotes clean division of work among teams of developers

Solution: 6 Best Practices

- => 6 Best Practices:
  - Develop iteratively
  - Manage requirements
  - Use component architecture
  - **Model software visually**
  - Verify quality continuously
  - Manage changing

Visual Modeling With the Unified Modeling Language





+ Capture the structure and behavior of components
+ Keeps design and implementation consistent
+ Hide or expose details for the task
+ Best for clear communication

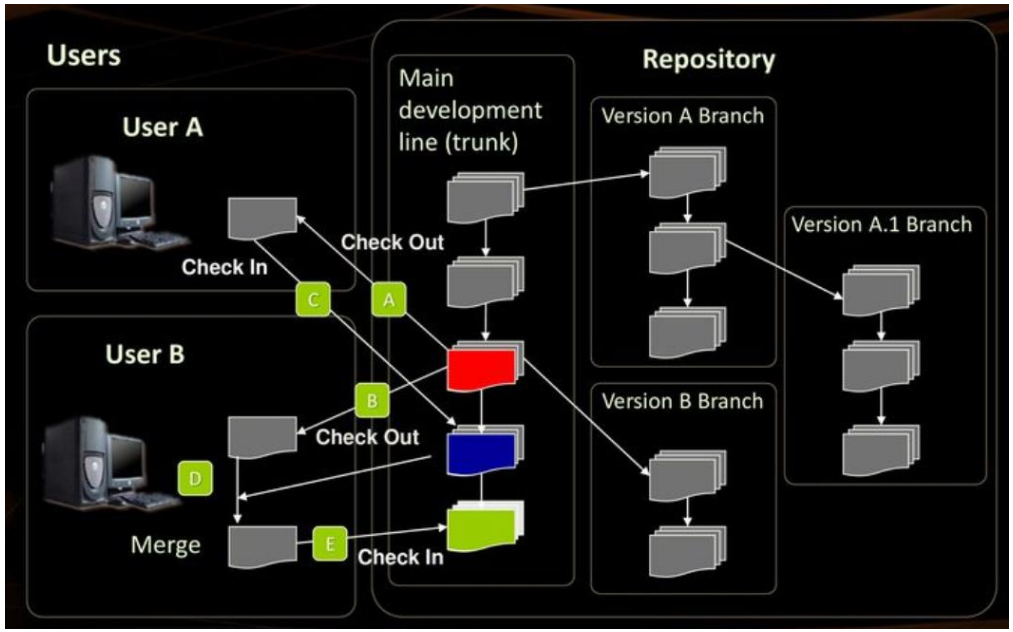
Solution: 6 Best Practices

- => 6 Best Practices:
  - Develop iteratively
  - Manage requirements
  - Use component architecture
  - Model software visually
  - **Verify quality continuously**
  - Manage changing

+ Problems are 100 to 1000 times more costly to find and repair after deployment
+ Need to test functional, usability, reliability, performance etc.
+ Develop test suites for each iteration within the development life cycle

Solution: 6 Best Practices

- => 6 Best Practices:
  - Develop iteratively
  - Manage requirements
  - Use component architecture
  - Model software visually
  - Verify quality continuously
  - **Manage changing**



(Svetlin Nakov: Source Control Systems)

+ Different teams all work together on multiple iterations, releases, products, and platforms
+ Establishing a secure workspace for each developer
+ Plan to introduce change in a particular step/iteration
+ Change Tracking

Solution:

- 6 Best Practices : Following engineering process is important !
- => *Systems development process (Systems development life cycle)*

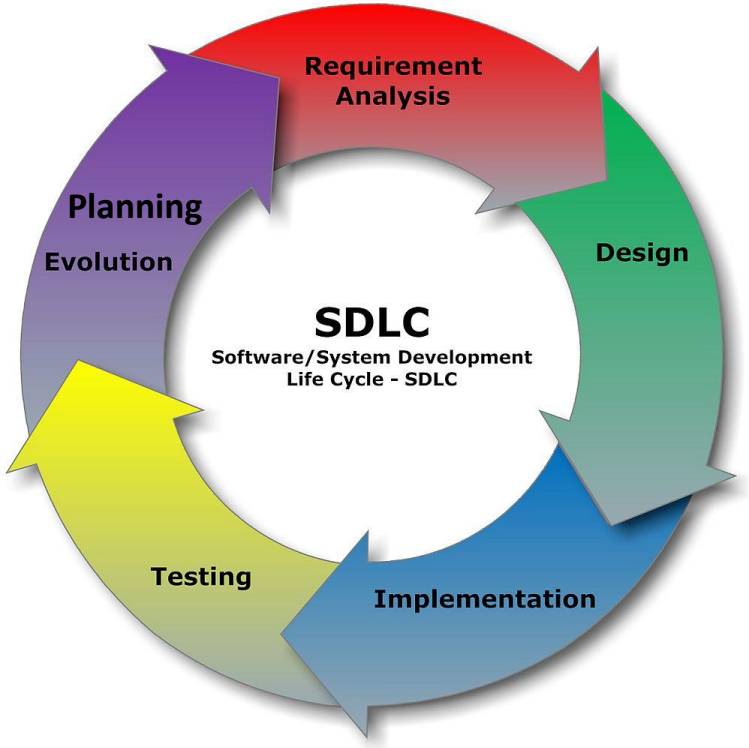
https://www.youtube.com/watch?v=cILODMGbtbk

1.2. Systems development life cycle (SDLC)

- SDLC is a description of phases in the life cycle of a software application.
  - Consists of a detailed plan as how to develop, build and enhance a specific software.
  - Each phase of the SDLC lifecycle has its own process and deliverables that feed into the next phase.
- Importance of Software Development Life Cycle (SDLC)
  - It acts as a guide to the project and meet client's objectives.
  - It helps in evaluating, scheduling and estimating deliverables.
  - It provides a framework for a standard set of activities.
  - It ensures correct and timely delivery to the client.

SDLC phases

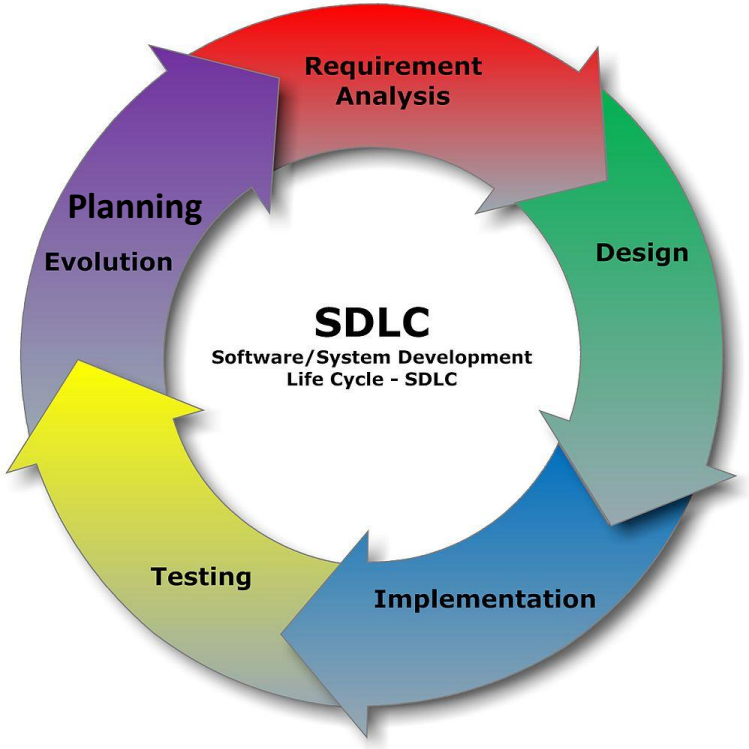
- Planning - why
  - Project Initiation
    - Develop a system request
    - Conduct a feasibility analysis
  - Project Management
    - Develop work plan
    - Staff the project
    - Control and direct the project
- Analysis
  - who will use the system,
  - what the system will do,
  - and where and when it will be used



  - Requirements gathering -> analysis models which describe HOW the business will operate if the new system is developed.
  - Output: System proposal

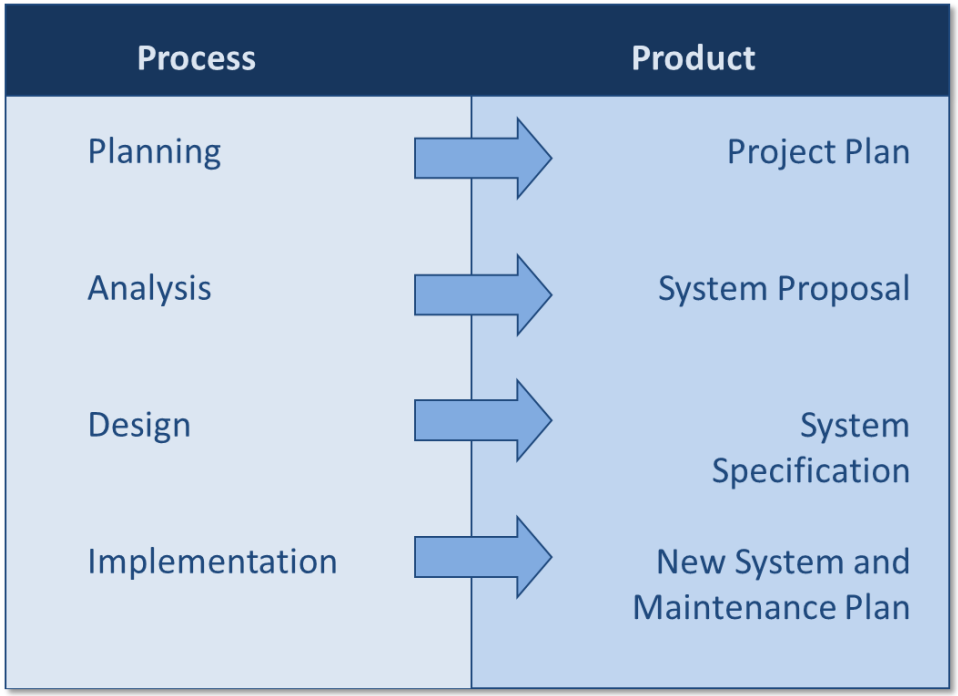
SDLC phases

- Design:
  - how the system will operate (hardware software, network infrastructure; user interface, forms/reports; databases, et
  - Architecture design, interface design
  - Database design
  - Program design
- Implementation
  - System construction
  - Installation
  - Support/maintenance plan



SDLC phases

- Processes and Deliverables

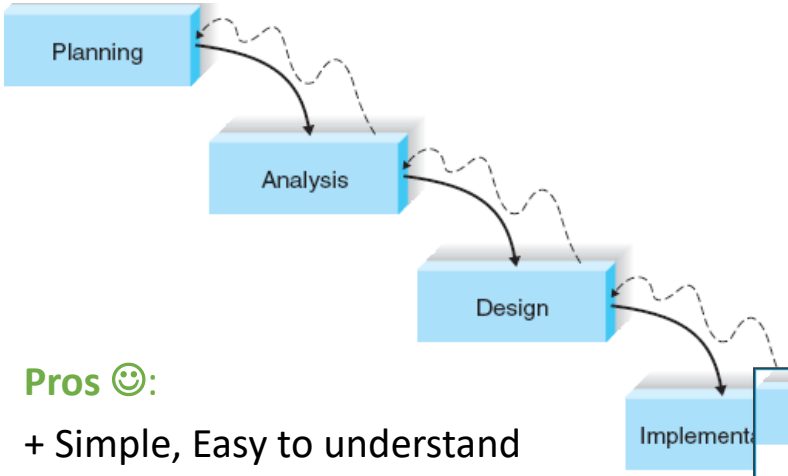


1.3. Common systems development methodologies

- Approach to implementing the SDLC
- Common models:
  - Waterfall Model
  - Parallel Model
  - Phased Model
  - Prototyping Model
  - Spiral Model
  - Agile Model
  - etc.

1.4. Common systems development methodologies

- Waterfall Model



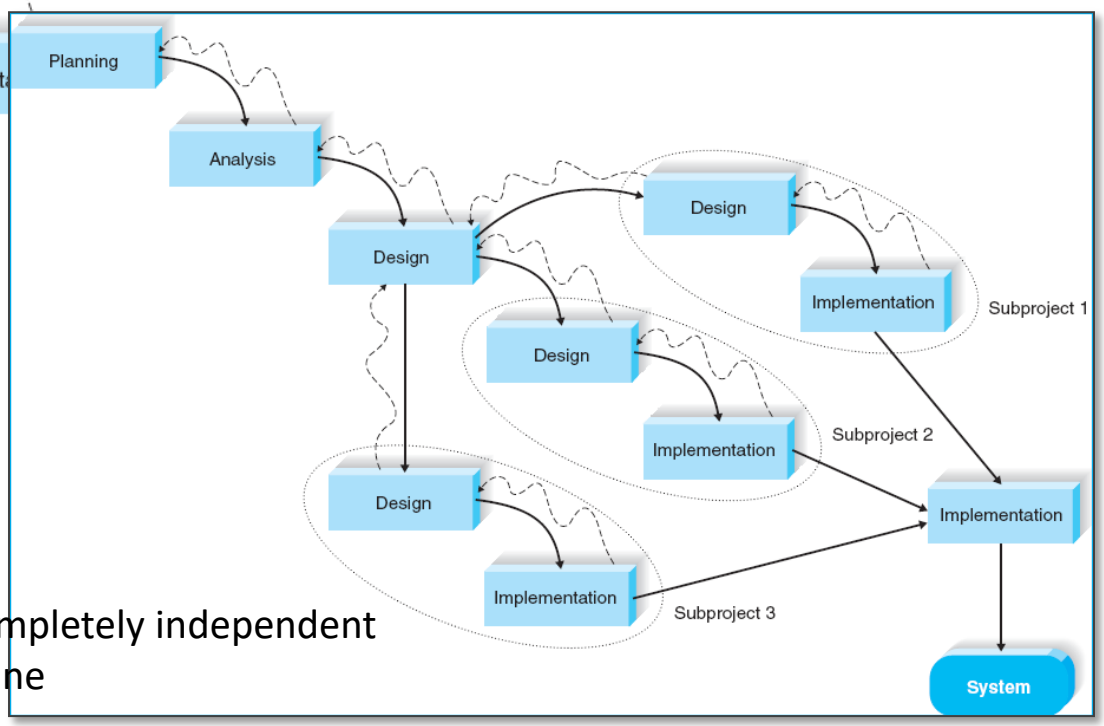
Pros ☺:

+ Simple, Easy to understand

Cons ☹:

- Not flexible for change
- Less Customer Engagement
- First release takes a long time
- High Risk at Release

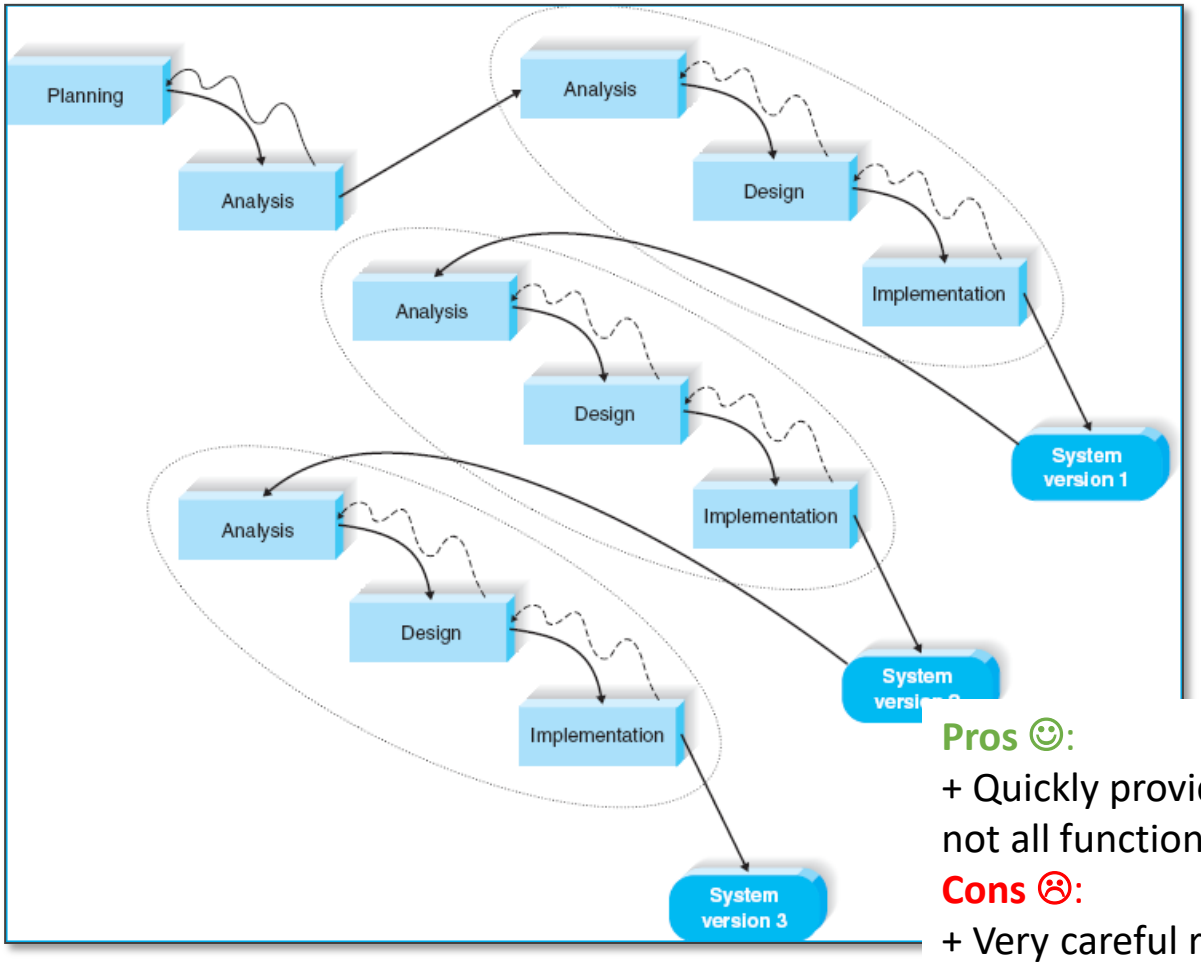
- Parallel Model



Cons ☹:

- sub-projects not completely independent
- takes time to combine

- RAD (Rapid Application Development)
  - Phased Model



Pros ☺:

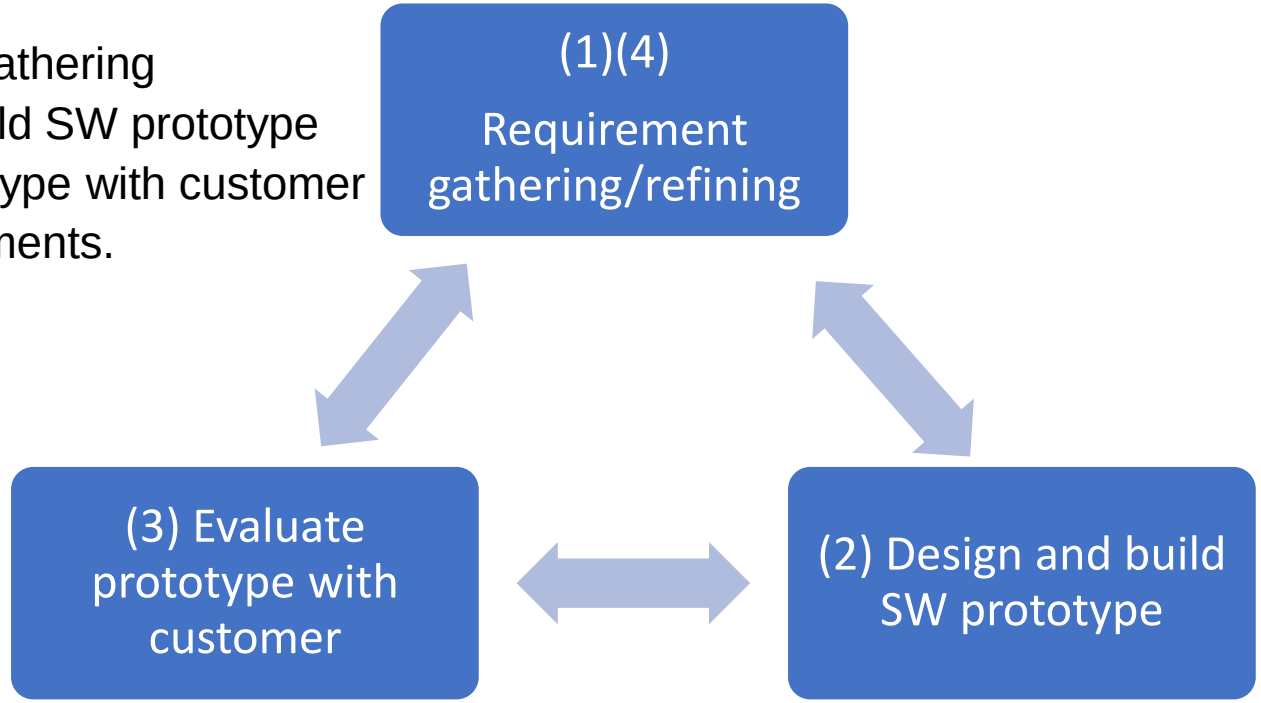
+ Quickly provide the system for users although not all functionality is complete

Cons ☹:

+ Very careful requirement analysis at the very beginning

1.4. Common systems development methodologies

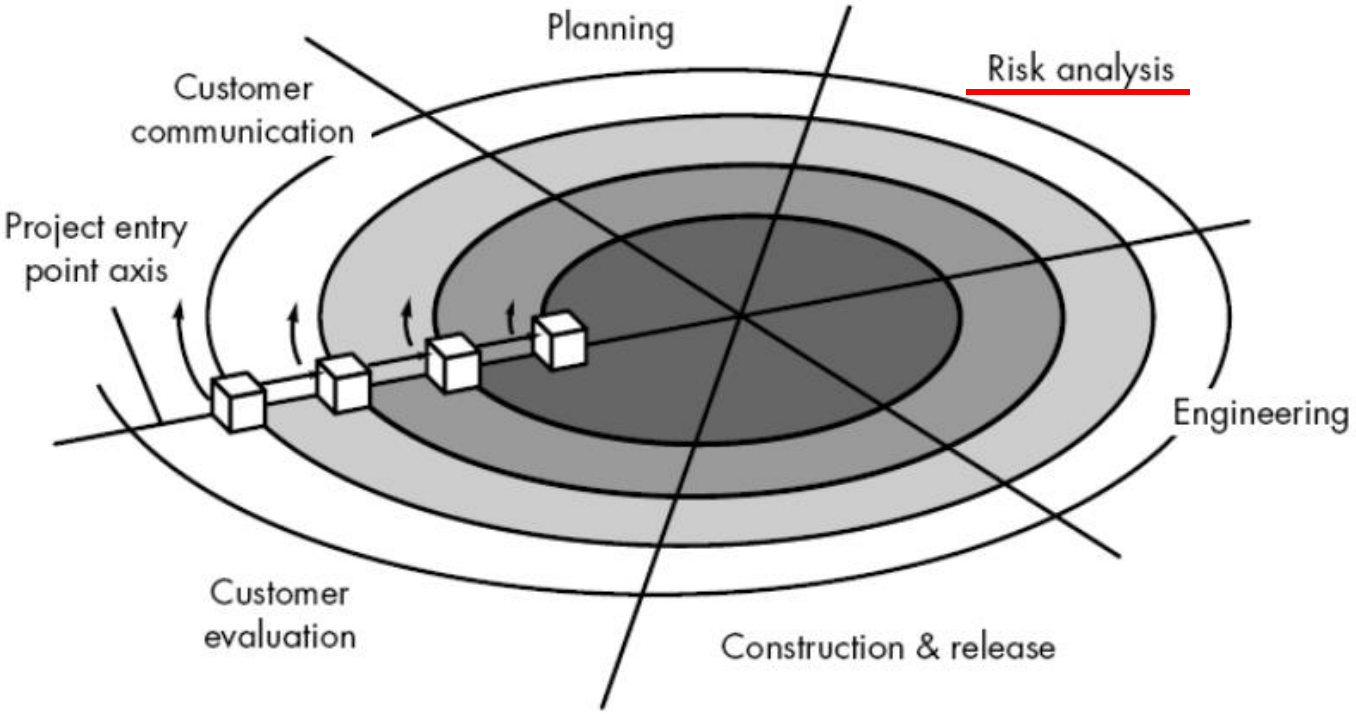
- RAD (Rapid Application Development)
  - Prototype Model
    - (1) Requirement gathering
    - (2) Design and build SW prototype
    - (3) Evaluate prototype with customer
    - (4) Refine requirements.



Pros ☺:
+ More Flexible
+ More Customer Interaction
+ Low Risk at Release

Cons ☹:
- Time Management
- Complex Model
- Costly process

- Spiral Model (Iterative Model)



Pros ☺:
+ More Flexible
+ More Customer Interaction
+ Low Risk at Release

Cons ☹:
- Time Management
- Complex Model
- Costly process

1.4. Common systems development methodologies

- Agile methodology
  - "*Agility is the ability to both create and respond to change in order to profit in a turbulent business environment.*"

-- Jim Highsmith, Agile Software Development Ecosystems, Preface XXIII

- Goal:
  - Develop quickly & Respond to change

1.4. Common systems development methodologies

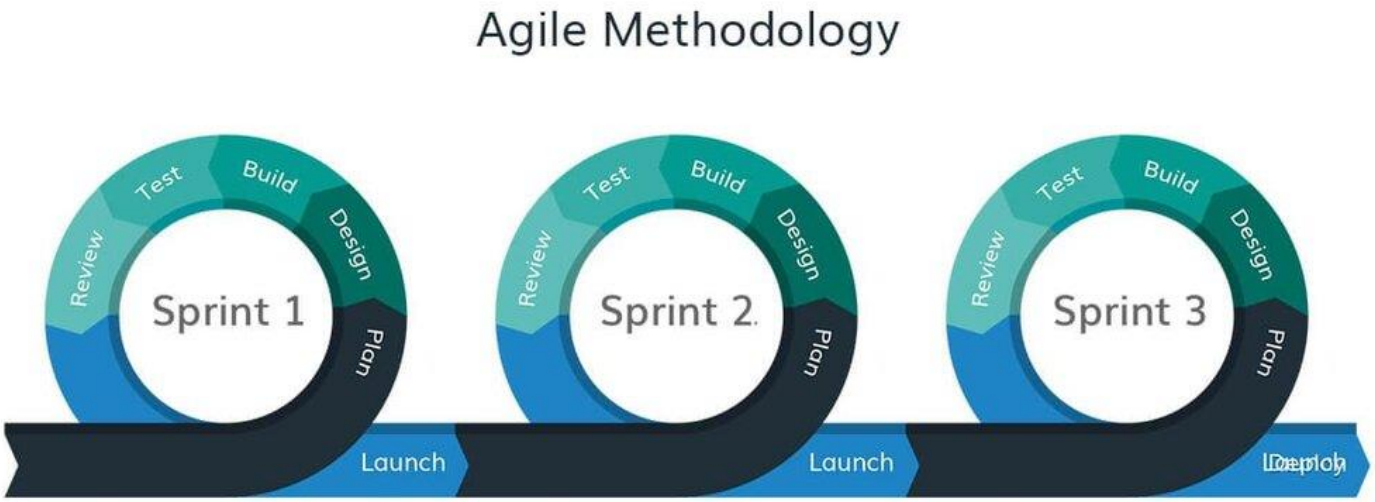
- Twelve principles of *Agile Software Development*

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Deliver working software frequently (weeks rather than months).
4. Close, daily cooperation between business people and developers.
5. Projects are built around motivated individuals, who should be trusted.
6. Face-to-face conversation is the best form of communication (co-location).
7. Working software is the primary measure of progress.
8. Sustainable development, able to maintain a constant pace.
9. Continuous attention to technical excellence and good design.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. Best architectures, requirements, and designs emerge from self-organizing teams.
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly.

https://en.wikipedia.org/wiki/Agile_software_development

1.4. Common systems development methodologies

- Agile methodology



Pros ☺:

+ Change Dynamically to The Customers
+ Distribution of Work, self-organized teams
+ Cost and Delivery prediction

Cons ☹:

- Expert is required
- Not for modest development initiatives
- The cost is high

1.4. Common systems development methodologies

- Selecting the Right Methodology
  - Waterfall, Parallel models: Known problem, known solution
    - implementation of similar systems
  - Phased model: Fairly known problem, fairly known solution except the scale, short time schedule
  - Prototype model: Unclear user requirements, Short time schedule
  - Spiral Model: Fairly unknown needs and outcomes, very risky, very large and complex project
  - Agile Model: Unclear user requirements, Short time schedule, and need to respond to potential changes quickly.