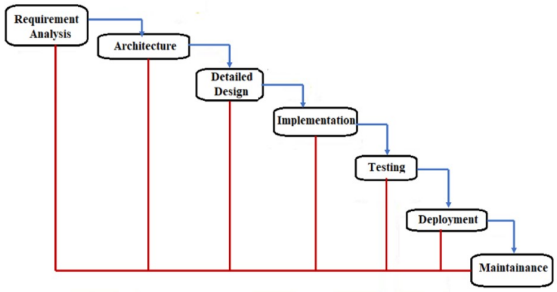
Legacy SDLCs and Agile

10 August 2023 09:49

Waterfall model



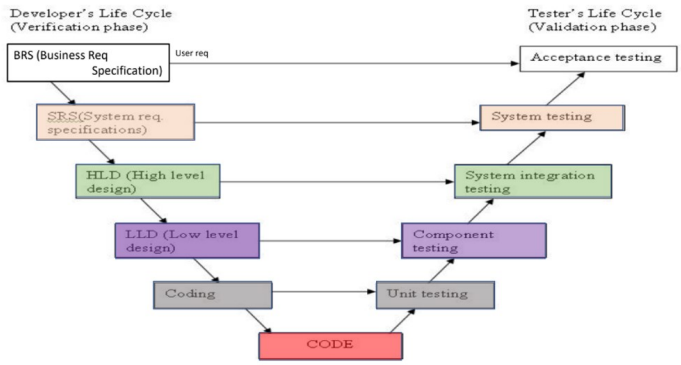
RAD ITD M

Waterfall model

ADVANTAGES	DISADVANTAGES
Simple	Assumes requirements are frozen
Clear identified phases	Difficult to change & sequential
Easy to manage due to rigidity	Poor model for long projects
Each phase – specific deliverables + reviews	Big Bang approach
Easy to departmentalize and control	High risk + Uncertainty

When can you use this model?
Pure form: Short projects where requirements are well known or Variant form: High level in long projects
Product definition is stable & technology is understood

V model



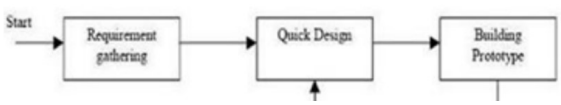
BSHLC -> Code -> ASSCU

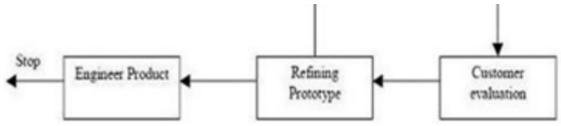
The V model

ADVANTAGES	DISADVANTAGES
Similar to Waterfall model	Similar to Waterfall model
Test development activities can happen before formal testing cycle	No early prototypes of software
Higher probability of success + Increased effectiveness of usage of resources	Change in process => change in test documentation

When can you use this model? Same as waterfall model
Pure form: Short projects where requirements are well known or Variant form: High level in long projects
Product definition is stable & technology is understood

Prototype model





- Cheap
- Entire system prototype is built to understand the requirements
- Types: Throw-away and Evolutionary

RQ BCRE (first 3 up, last 3 down)

The Prototype model – advantages and disadvantages

ADVANTAGES	DISADVANTAGES
Active involvement of users	May increase complexity of system as scope of system may expand beyond original plans
Better risk mitigation, Reduced time and cost, Resulting system is full featured, More stable system	Performance of resulting system may not be optimal

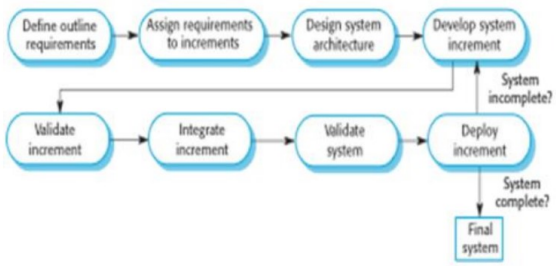
Usage:

- When requirements are not clear
- Users are actively involved

Incremental Model

- Requirements are partitioned
- Working software in first increment(module).
- Each subsequent release adds functionality to previous module
- Continuous integration is done until entire system is achieved

Incremental model



- Partitioned requirements can have a development lifecycle
- Models like waterfall can be used for each partition

ORAI VIVD

Incremental model

ADVANTAGES	DISADVANTAGES
Customer value and more flexible	Needs good planning and design
Easier to test and debug	Needs clear and complete definition of whole system
Easier to manage risk	Total cost is higher than waterfall
Continuous increments rather than monolithic	Hard to identify common functionalities across increments
Reduces over functionality	Management visibility is reduced

Incremental model When to use?

- Major requirements are defined
- Product needs to get to market early
- New technology is used
- High risk features and goals
- Resources with required skill set unavailable

Iterative model

- Iterative model
- Initial implementation starts from a skeleton of product
- This is followed by refinement through user feedback & evolution
- Built with dummy modules
- Rapid prototyping
- Successive refinement

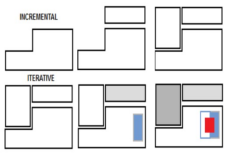
Iterative model

ADVANTAGES	DISADVANTAGES
Help identify requirement & solution visualization	Each phase is rigid with overlaps
Support risk mitigation, rework is reduced, incremental investment, feature creep, increased customer engagement	Costly system architecture may arise

USAGE: Large projects which may get extended

Iterative model vs Incremental model

ITERATIVE MODEL	INCREMENTAL MODEL
Revisit and refine every thing	No need to go back and change delivered things
Focus on details of things	Focus on things not implemented yet
Leverage on learnings	Does not leverage on experience or knowledge



31/08/2023

Limitations of most legacy lifecycle models Software Engineering

- Predictive software development methods
- Upfront planning
- Do not facilitate periodic customer interaction
- Suited for large complex projects
- Regulatory perspectives
- Suited for global and distributed organizations
- Product lifecycle and its ecosystem
- People and skill perspective
- Suitable for projects with clear definition
- Suitable when things are not changing too fast

Agile

Agile philosophy

Agile is an umbrella term used to describe a variety of methods

FDD – Feature Driven Development
DSDM – Dynamic System Development Method

Agile methods encourage:

1. Continual realignment of development goals with needs and expectations of the customer
2. Reducing massive planning overhead to allow fast reactions to change

Agile is not a process, It is a set of values or philosophy
AGILE key words

- Rapid
- Iterative
- Cooperative
- Quality driven
- Adaptable

RICQA

Agile manifesto

Individual and interactions	over	Process and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

While there is value in the items on the right, we value the items on the left more.

Agile principles

- Customer involvement : Their role is provide and prioritize new system requirements and to evaluate the iterations of the system
- Incremental delivery: customer specifying the requirements to be included in each increment
- People not process: Team members should be left to develop their own ways of working without

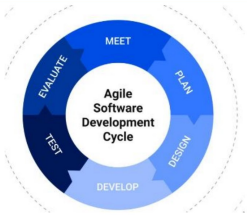
prescriptive processes.
- Embrace change: Design system to accommodate changes
- Maintain simplicity: Actively work to eliminate complexity

CIP CS

Agile pros and cons

Pros	Cons
• Promotes teamwork and cross training • Rapid development • Min resource requirement • Suitable for changing requirements • Early partial working solutions are delivered • Minimal rules • Documentation easily employed • Less planning • Easy to manage • Flexibility for developers	• Not suitable for complex dependencies • Must meet deadlines • Heavily dependent on customer interaction • Minimum documentation generated, might be difficult to transfer to new team members.

SDLC in Agile



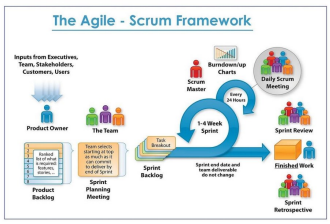
31/08/2023

MPD DTE

Agile methodologies

Scrum

A framework of rules, roles, events, and artifacts used to implement Agile projects.
- Iterative approach
- consists of sprints that typically only last one to four weeks
- ensures that your team delivers a version of the product regularly
- Can be used for complex projects
- Better for software product over software service



Burndown chart : shows the amount of work that has been completed in an epic or sprint, and the total work remaining. Used to predict your team's likelihood of completing their work in the time available.
An agile epic/sprint is a body of work that can be broken down into specific tasks (called user stories) based on the needs/requests of customers or end-users.

How is Scrum aligned to the Agile principles?

Individuals and interactions over Processes and tools - Scrum addresses this with Cross functional teams, Scrum Meetings, Sprint reviews

Working software over Comprehensive documentation - Periodic customer experienceable deliverables at the end of every sprint which can be reviewed and experienced

Customer collaboration over Contract negotiation - Having customers to experience the sprint outcomes and participate in sprint reviews to ensure they can visualize and ensure that product meets their needs

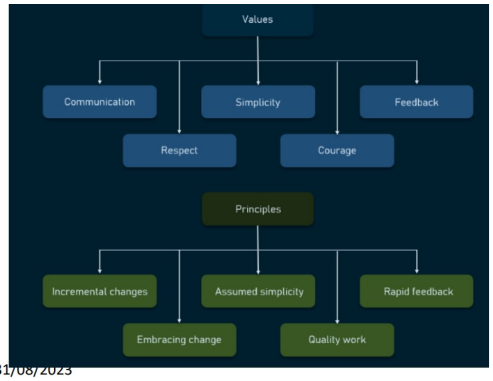
Responding to change over Following a plan - User stories which is picked at the beginning of every sprint which can ensure requirement changes can be factored in and prioritized unlike a plan which needs to be followed

Focus on Simplicity in both the product and the process by keeping the process simple, planning is short term focused and hence simple with more interactions and minimal documentation

XP (Extreme programming)

XP is built upon values, principles, and practices, and its goal is to allow small to mid-sized teams to produce high-quality software and adapt to evolving and changing requirements

XP values and principles



31/08/2023

XP Core practises

1. Planning game
    a. 2 levels
        i. Release planning: The team and stakeholders/customers the requirements that can be delivered into production and when. This is done based on priorities, capacity, estimations and risks factor of the team to deliver.
        ii. Iteration planning : team will pick up the most valuable items from the list and break them down into tasks, then estimates and a commitment to delivering at the end of the iteration.
2. Simple design
    a. Team will not do complex or big architecture and designs upfront
    b. The code is frequently refactored so that it will be maintainable and free of technical debt
    c. SPIKE: XP teams conduct a small test or proof-of-concept workout : helps team understand validity of hypothesis and gauge complexity of solution

3. TDD (Test driven development)
    a. write the unit test cases before starting the coding.
    b. Helps during build and integration stages
    c. Only code that passes test cases has to be generated
    d. Steps of TDD:
        i. Write the unit test case with minimal amount of code to pass the test
        ii. Refactor it by adding the needed feature and functionality, while continuously making sure the tests pass

4. Code standard
    a. consistency in code, style, naming conversion, exception handling and use of parameters
    b. Must be defined and agreed upon before coding
    c. Makes code simple, helps detect problems quickly and increases efficiency of software

5. Refactoring
    a. Improving code quality without altering behaviour
    b. Consider
        i. Ensure code or functions are not duplicated.
        ii. No long functions or methods
        iii. Removing unnecessary variables
        iv. Proper use of access modifiers etc.

6. Pair programming
    a. Working in pairs
        i. Pilot: Focuses on writing clean code, and compiling and running
        ii. Navigator: Focuses on the big picture and reviews code for improvement or refactoring
    b. The pair intermittently switches roles
    c. This way, everyone gets an idea of working of the entire system

7. Collective code ownership
    a. Success and failure is collective

8. Continuous integration
    a. Integrate changes intermittently
    b. Unit test cases executes automatically on integration for entire project
    c. Defect propagation avoided as errors can be fixed immediately

9. Small release
    a. Releases Minimum Viable Product (MVP) frequently
    b. Helps break down complex modules
    c. Demonstrates product to stakeholders and customers and helps work on highest priority tasks

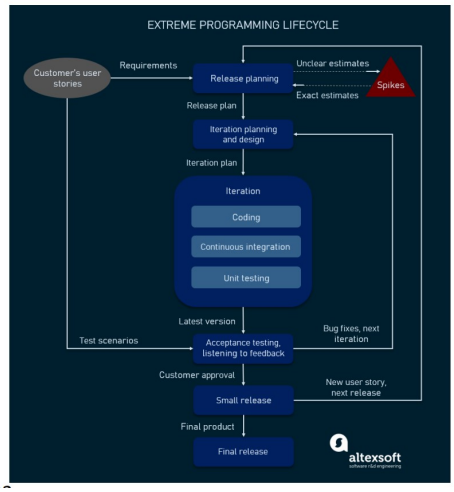
10. System Metaphor
    a. Stories must be simple enough to be easily understood by user and developers and to relate it with code
    b. Ex) Order_Food() is easily explained -- this will be used to order food. It easy for the team to relate to the functionality of the particular component by just looking at its name

11. Onsite customer
    a. They are the experts who know the domain or product and know how to generate a return of investment (ROI) by delivering the minimum viable product (MVP)
    b. Similar role to product owner in scrum

12. Sustainable pace
    a. This is a people-centric practice
    b. XP maintains a sustainable pace by introducing down time during the iteration
    c. The team is not doing actual development at this time but acts as a buffer to deal with uncertainties and issues.

d. Can do research/refactor code to keep up pace

PST CRP C(o)C(i)S(r)S(m) OS(p)



Lean Agile

- Minimize waste, maximize value.
- Quality is priority

Lean Agile - principles

EAD DE BO

CBSE (Component Based Software Engineering)

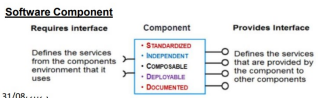
Implement or select components and integrate into systems

- Reuse rather than reimplement.

Advantages	Disadvantages
• Increases productivity • Reduces cost • Black box usage of components • Improves quality	• Requirement trade off • Component certification • Trusting components • Emergent property prediction

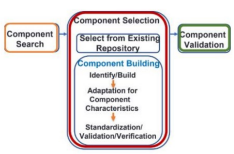
Essentials of CBSE

- Independent components that are completely specified by the public interfaces
- Component standards that facilitate the integration of components
- The component interface is published and all interactions are through the published interface.
- Middleware for component integration provides software support
- Development process that is geared up to CBSE



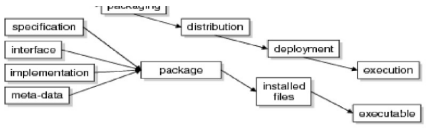
Identifying a component

Search for component -> Select Component -> Compose component from existing ones -> Adapters or "glue" to reconcile different component interfaces -> Validate the component



- Search for component
- Select from repository or Compose component from existing components
- Adapt/ reconcile for different interfaces
- Validate

Component Development Stages



Different forms of component representation:
During development using UML
When packaging - .ZIP
In the execution stage – blocks of code and data

Component characteristics

- Standardized
  - Define component interfaces, metadata, documentation, composition, deployment
  - Must conform to standard model
- Independent
  - Should be able to compose and deploy without using other components
  - If some service is required should be specified in requirements section
- Composable
  - All external interactions should take place through publicly defined interface

Software Product Line

- A group of connected products marketed under a single brand name by the same company
- Set of software systems that share elements. In a software product line, reuse is planned, not accidental.

Software Line Engineering Framework