What is agile or scrum - answer depends on who is answering

It makes it important to understand from all perspective

All IT Software work is categorized in Project, Product, Operations/Service

A project is a temporary endeavor undertaken to create a unique product, service, or result. Projects have a defined start and end, specific objectives, and involve a series of activities that must be executed to achieve the set objectives.

Sounds complex isn't it

Simplified version

A project is a temporary work to create a unique product, service, or result. Projects have a defined start and end, specific objectives, and involve a series of tasks

Destined start and end example, for lets say, a website development project

- **Defined Start**: The project starts on a specific date, when we have the kickoff or initial meeting meeting with the client to discuss requirements and set goals.
- **Defined End**: The project ends on a specific date, when the website is fully built, tested, approved by the client, and goes live.

- temporary, unique , defined start and end
- not a regular, ongoing

Now whereas for product,

products have a life cycle (*different stages* and may exist indefinitely (until they are retired or replaced).

products have a life cycle," we mean that a product goes through different stages from the time it is first introduced to the market until it is removed or replaced.

1. **Introduction**: **product is first launched** into the market. Sales are usually slow at this point because people are just getting to know the product.
2. **Growth**: If the product is liked by customers, **sales will start to grow.** This is a good time for a company to invest more in marketing.
3. **Maturity**: At this stage, the product is well-known and **sales may level off**. Many products spend the most time in this stage.
4. **Decline**: Eventually, **sales start to go down**. This could be because there are newer products, or people's preferences have changed.

Products can be physical goods, digital software, or services and are often the output of one or multiple projects.

What is the relationship between project and product

1. A project can be initiated to create, improve, or retire a product.
2. A product may require multiple projects throughout its life cycle.

Sponsor : pays for the project , COST
Business Uses : give requirements
Project manager : Handles overall tasks execution
Implementation team (UX, architect, developer, QA) :
Requirement Analysis, design, develop/implement, test, deploy
Business users : UAT
Maintenance

At first overall project execution was very ad hoc

then SDLC was used -
Software Development LIFE CYCLE - which has Stages,

earlier
- Ad hoc development
- Code and fix model
- Lack of documentation
- Absence of formal testing
- Manual processes

SDLC has - many models - 4 most popular
Lets see each in detail
Waterfall Model:
A linear and sequential approach (specific order, one after another.)
where each phase must be completed before the next phase begins.

It's simple and easy to understand but lacks flexibility.

Iterative Model:
The software is developed in iterations or versions.
Each iteration is a mini-waterfall cycle with all the SDLC phases.
After each iteration, the product is refined based on feedback.

Incremental Model:
The software is developed and delivered in increments or parts.
Each increment adds some functional capability to the existing product until the complete product is achieved.

V-Model (Validation and Verification):
An extension of the waterfall model.
Each development stage corresponds to a testing phase.
Emphasizes validation and verification.

Waterfall

Let's start with the waterfall model as I personally feel it is a base model - a model from which other models have borrowed some things. Getting familiar with waterfall will help us understand other models better.

It has
A linear and sequential approach (specific order, one after another.) where each phase must be completed before the next phase begins.

Phases:
Requirement , Analysis, Design, Implementation/Development/Coding, Testing, Deployment, Maintenance

1. **Requirements**: Gather all functional and non-functional requirements to define the scope of the software.

   Functional Requirements:

   These are the basic things that a software must do. They describe specific behaviors or functions of a system.

   - **Example**: A login screen must authenticate or check user id and password of users and login user within 2 seconds.

Non-Functional Requirements:

These are the qualities that a software must have, like how fast it works or how easy it is to use.

- **Example**: The system must be able to support 500 users at the same time, or the user interface must be intuitive.

Functional requirements are about what the software should do, while non-functional requirements are about how well it should do it.

2. **Design**: Create system architecture and design documents, detailing how components will interact.
3. **Implementation**: Write the source code according to the design specifications.
4. **Testing**: Conduct tests to identify defects and ensure the software meets all requirements.
5. **Deployment**: Release the finalized code into a production environment for end-users.
6. **Maintenance**: Monitor the system for issues, implement fixes, and release updates as necessary.

Each phase is completed before proceeding to the next, ensuring a linear and sequential approach.

There are Phase gates at each stage
What are they

Phase gates are checkpoints in a project where certain criteria must be met before the project work execution can proceed to the next phase.

For it to go from the current phase to the next phase, it must meet certain conditions.

phase gates are common and clearly defined between each phases
These gates serve as quality checkpoints, where project stakeholders, often including managers, team leaders, or client representatives, so all stakeholders review the current status and deliverables/OUTPUT of the project at that STAGE/PHASE and make sure they meet predefined criteria.

Lets see some examples of phase gates

1. Requirements
Analysis ,
Gathering
sometimes client provides it

Can be captured in a series of meetings with business users, managers and documented all details. The document is reviewed and agreed by all the stakeholders

called specification document

E.g. for requirement phase , criteria is to have requirements documented and reviewed by all stakeholders

Actions: Approval to begin the Design phase or send back for revisions.

supplementary tasks - Test strategy, Scenarios etc can be created here, kind of planning for next phases

2. System and Software Design
Phase Gate: Design Review

Criteria: High-level architecture and detailed design documented and reviewed.
Actions: Approval to begin coding, or requirement to revisit design phase.

3. Implementation (Coding)
Phase Gate: Code Complete Review

Criteria: All features coded, unit-tested, and code-reviewed.
Actions: Approval to proceed to testing, or go back for code revisions.

4. Testing
Phase Gate: Testing Review

Criteria: All planned tests completed, and defects resolved or accounted for.
Actions: Approval to go on to deployment, or requirement to revisit testing or possibly earlier phases for bug fixes.

5. Deployment
Phase Gate: Deployment Review

Criteria: Successful deployment to a staging environment, user acceptance testing completed.
Actions: Approval to move to production or go back for deployment fixes.

6. Maintenance
Phase Gate: Post-Deployment Review (optional)

Criteria: project working as expected (requirements), any reported issues resolved or within acceptable limits.

Actions: Formal project closure and transition to ongoing support, or go back to address issues that require immediate resolution.

If the criteria are met, the project is allowed to proceed to the next stage; if not, it may be delayed for revisions or rework, or in some cases, terminated.

Note : **ALL THE DETAILS - EVERYTHING - should be DOCUMENTED and ALL such DOCUMENTS at each phases, should be REVIEWED and APPROVED by ALL Stakeholders**

Please note DETAILS and REVIEW and APPROVAL of ALL Stakeholders - Who are RELATED and IMPACTED by it.

This is a challenge for waterfall

Good

- Simple to understand
- Very well structured and defined - very clear what to do at each phase, process , procedures, best practices, past projects learning are available, Cant handle changes in Scope (requirement)
- Straightforward to implement and work with
- Detailed documentation is created which is good for
    - Clarity, Traceability , identifying Risk, compliance
- High documentation requirements - regulated industry -
    - MDR project, PISA project - talk about these

- Approval before moving to next stage gives confidence
- Easy to Measure and Control
- Lower Uncertainty and Risk (for whatever scenario at the beginning at the planning)
- Client Involvement at Beginning (requirement) and End (UAT)
- Good for Well-Defined Project with no changes in requirements/Scope
- Ideal for Regulatory Documentation

Assumption

- Everything is predictable
- Clear and Well-Defined Requirements

- Stable Product Definition and requirements- will not change
- Sequential Phases - can be executed in sequence
- No Overlapping Phases
- Predictable Technology and Tools
- Availability of Skilled Resources
- Accurate Estimates - cost , schedule
- Client is available for Involvement Mainly at Beginning and End
- Limited Complexity or Uncertainty - something similar did in past

challenges

- Inflexible to Changes - difficult to run overlapping phases
- Late Discovery of Issues
- Delayed Testing till last stage after all development is done
- High Risk and Uncertainty - because in long duration projects it happens
- Resource Idle Time if previous phase delays
- Client Involvement Limited, not always available - past experience
- Not Suitable for Complex Projects with multiple moving parts or very large
- Long Time to Market as product is available at the very end
- Difficult to Estimate Time and Cost in general for complex project

Just show estimation techniques

- **Analogous Estimating**: Comparing the current project to similar past projects to guess its size or duration.
- **Parametric Estimating**: Using statistical models and historical data to make projections, like calculating the cost per square foot to build a house.
- **Three-Point Estimating (PERT)**: Taking the best, worst, and most likely estimates to find an average, helping to account for uncertainty.
- **Bottom-Up Estimating**: Breaking down the project into smaller parts and estimating each one, then adding them up for the total.
- **Top-Down Estimating**: Starting with the big picture and using it to estimate the time and cost of individual tasks.
- **Expert Judgment**: Asking someone with experience or expertise to give an estimate.
- **Monte Carlo Simulation**: Using computer software to model different scenarios and predict project outcomes.
- **Reserve Analysis**: Adding extra time or money into the estimate as a safety net for unknowns or risks.
- **Cost of Quality**: Estimating the money needed to make sure the project meets quality standards.
- **Vendor Bid Analysis**: Looking at bids or quotes from suppliers to figure out project costs.
- **Decomposition**: Breaking the project down into smaller pieces to make estimating easier.
- **Delphi Technique**: Collecting estimates from experts, summarizing them, and re-estimating until everyone agrees.

- **Historical Information**: Using data from past projects to help make estimates for the new project.
- **Earned Value Management**: Using past performance and planned budgets to figure out future project status.
- **What-If Scenario Analysis**: Exploring different situations that could happen and how they would affect the project.

- No Working Software Until Late after development is completed and QA is done / UAT
- Requires Complete Knowledge Upfront
- Risk of Project Abandonment - if milestone based contract
- Poor Adaptability to Emerging Technologies - for large projects in beginning design is done based on technologies are available at that time, any new version of the tools/libraries or different tools (which was estimated as alternate) is release
    - We provide 3 different approaches / technologies to client and provide benefit analysis comparison and chose best one
- May Overemphasize Documentation - even when not really required and we tend to give more importance to it when not required
- High Involvement of client during Requirement management - sometimes not possible to get involvement if requirement phase is longer
- Client not sure how the end product should look like - but they will give requirements which can change later on after execution begins
- Need for having the best features prioritized for better marketability but cannot prioritize for delivery as all requirements regardless of priority will get delivered at same time/stage
- Little or no Business value is delivered until the system is complete

Best suited for

- Well-Defined Requirements
- No changes in requirements / stable requirements- mostly impossible for large duration projects
- Simple or Routine Projects
- Fixed Timeline and Budget
- Regulated industry (Construction)
- Stable Technology
- Regulatory Documentation Needed
- Low Uncertainty or Risk
- Limited Client Involvement
- Skilled and Specialized Team
- Single Delivery Point

- Has high predictability
- All risks are addressed (managed, mitigated, contingency)
    - **Risk management**

For Negative Risks or Threats:

1. Avoid: Change your plans so that bad things won't happen.
2. Mitigate: Take steps to make it less likely or less harmful.
3. Transfer: Make it someone else's problem, maybe by paying for insurance.
4. Accept: Know it might happen, but don't do anything for now. Have a backup plan ready.

For Positive Risks or Opportunities:

1. Exploit: Do what you can to make sure the good thing happens.
2. Enhance: Try to make the good thing even better or more likely.
3. Share: Team up with someone to make a good thing happen.
4. Accept: If it happens, great. If not, that's okay.

For Any Risk, Good or Bad:

- Escalate: If you can't handle it, tell someone higher up who can help.

Risks in project management can be categorized in various ways to better understand and manage them. Here are some common types of risks:

Based on Source:

1. **Technical Risks**: Issues with technology, software, or hardware.
2. **Financial Risks**: Running out of money or budget issues.
3. **Operational Risks**: Problems with day-to-day operations.
4. **Market Risks**: Changes in market demand or competition.
5. **Legal Risks**: Legal issues like lawsuits or regulatory changes.
6. **Environmental Risks**: Natural disasters or environmental concerns affecting the project.

Based on Impact:

1. **High-Impact Risks**: Could have a big negative effect on the project.
2. **Low-Impact Risks**: Less damaging but still important to manage.
3. **Positive Risks**: Opportunities that could benefit the project.

Based on Probability:

1. **High-Probability Risks**: Very likely to happen.
2. **Low-Probability Risks**: Less likely but could still happen.

Based on Predictability:

1. **Known Risks**: Risks that you know about and can plan for.
2. **Unknown Risks**: Risks that you can't predict or haven't thought of yet.

1. **Plan Risk Management**: Decide how to approach, plan, and execute risk management activities for the project.
2. **Identify Risks**: Create a list of potential risks that could affect the project.
3. **Perform Qualitative Risk Analysis**: Prioritize risks based on their impact and likelihood.
4. **Perform Quantitative Risk Analysis**: Numerically analyze the probability and impact of identified risks.
5. **Plan Risk Responses**: Develop specific plans for the highest priority risks, categorizing them as threats or opportunities and planning responses accordingly.
6. **Implement Risk Responses**: Execute the risk response plans as needed during the project.
7. **Monitor Risks**: Continuously monitor and reassess risks; identify new risks, and evaluate the effectiveness of risk responses.

It's worth noting that while the Waterfall model is straightforward and easy to understand,
it's not always the best fit for some kinds of projects, especially those where requirement may change,
projects that are complex and require flexibility.
In such cases, other SDLC methodologies like Agile or Iterative might be more appropriate.

MVP

The idea behind MVP is to quickly build, release, and test a basic version of a product to learn from it and make improvements based on feedback.

MVP - PB prioritization, refinement, DoD can be minimum, during sprint review show minimum required features necessary, increment delivered to focus on MVP

SDLC takes care of

Scope is fixed

Time and cost are variable

Agile takes care of

Cost - No of resources in sprint/release is fixed

Time - Sprint duration is fixed

Scope is adjusted

Why any company moves to Scrum

Cust satisfaction

Faster delivery

Motivated team

Iterative

- The software is developed in iterations or versions.
- Each iteration is a mini-waterfall cycle with all the SDLC phases.
- After each iteration, the product is refined based on feedback.

- In the Iterative model, you make versions of the whole project again and again, improving it each time until it's finished.
- **Example**: You build a basic car, test it, then rebuild it better, test it again, and so on until you have the perfect car.

- **Example**: You release a simple word processor that just allows text input. In the next iteration, you add spell-check, and in the one after that, you add formatting options, always improving the entire software.
- **Example**: You build a basic car, test it, then rebuild it better, test it again, and so on until you have the perfect car.

Incremental Model:

In the Incremental model, you add new pieces or features to the project step-by-step until it's complete.

- **Example**: First, you build the car's engine, then you add the wheels, then the body, and so on until the car is complete.
- **Example**: First, you develop the user login system. Once that's done, you add a file storage feature. After that, you add the ability to share files with other users. Each feature is a new increment, and they all add up to make the complete software.

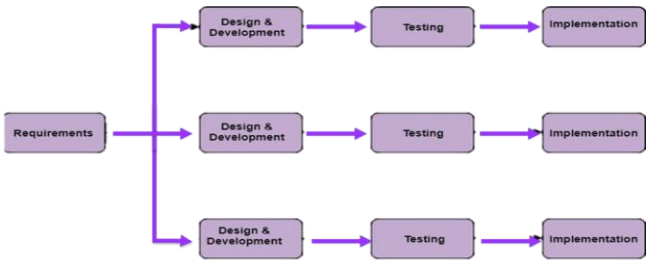
- The software is developed and delivered in increments or parts.
- Each increment adds some functional capability to the existing product until the complete product is achieved.

- **Planning**: Define the scope, resources, and initial requirements for the project.
- **Design**: Create architectural and detailed design specifications based on current requirements.
- **Implementation**: Code and develop the features for the current iteration.
- **Testing**: Evaluate the functionality and performance of the developed increment.
- **Evaluation**: Gather feedback and review the iteration's outcome to plan for the next cycle.

The idea behind MVP is to quickly build, release, and test a basic version of a product to learn from it and make improvements based on feedback.

The Incremental and Iterative models are more flexible and allow for changes to be made more easily as the project progresses.

Client get to see product relatively early (SDLC)



Still mini SDLC

Disadvantages of Waterfall still present

similar roles of SDLC, no specialized roles for this model

No client involvement after requirements

Inadequate Stakeholder Engagement

Lack of Formalized Planning and Review:

High Risk due to relatively still late testing
Complex requirement management
No guidelines on duration, feedback, requirements mgmt.

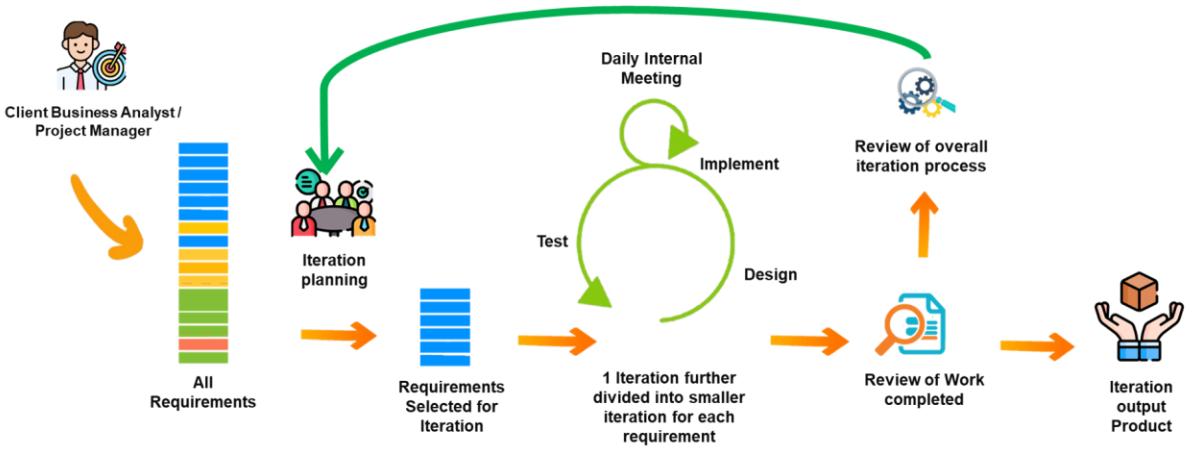
High Risk due to late testing
Lengthy Time to Market

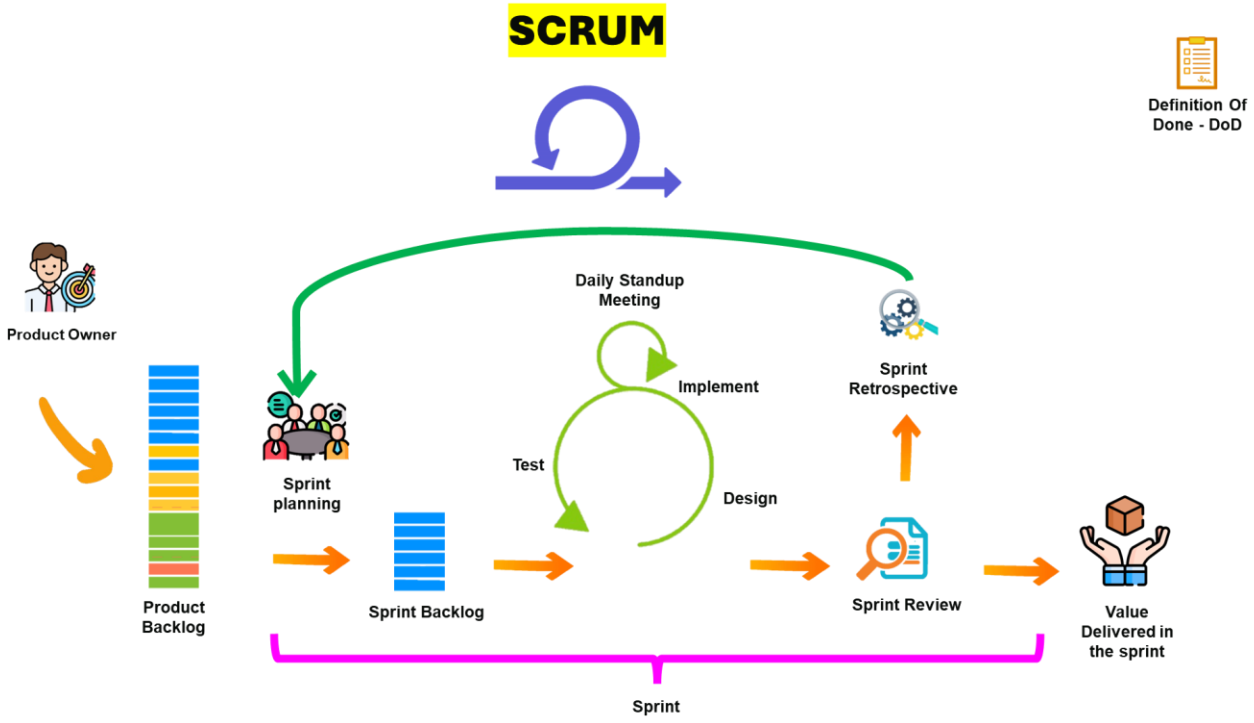
Need for better framework for

- Formalized requirement management
- Clear prioritized requirements
- each iteration
  - Consistent duration, shooter duration to manage
  - requirement management
  - early testing
- Goal needs to be clear
- Formalized and defined meetings
- Stakeholder engagement
- Early and regular feedback
- Better than SDLC phases, something new

- Waterfall has been the best model there but
- Iterative or Incremental model are better than Waterfall
- Looking at Iterative, incremental models , can we transform them into
  - something like this

Someone from client / Customer side maintains all requirements
Work goes on in iteration - short and manageable
few requirements at a time
Requirements are estimated to check if we have capacity to deliver at the end of iteration
Part of requirements are taken into requirements that will be worked on in iterations
We follow design, implementation and test/QA in iteration and review work at the end of increment
And review overall process iteration process followed
If all ok then deliver product output





- Customer representative who managers requirements = PO
- Requirements called as Product backlog
- Product backlog (requirements) are detailed and prioritized by PO
- Iteration = Sprint
- Iteration planning where we select requirements for iteration and estimates = Sprint planning
- Requirements selected for iterations sprint = Sprint Backlog

- Iteration review = Iteration review to check output of iteration, work done = Sprint review
- PO/Customer checks output of iteration /sprint based on criteria = Acceptance Criteria, Definition of Done
- Process review = Sprint Retrospective

Each requirement in sprint backlog is estimated
User story is created

A User Story is a simple description of a feature or function in a software project, usually written from the perspective of the person who will use that feature. It's a way to break down complex requirements into manageable pieces. A User Story typically follows a simple format like: "As a [type of user], I want [an action] so that [benefit/value]."

User story

As a [role], I want [goal] so that [benefit].
As a user, I want to be able to sign in to the application using my email and password so that I can access my personalized account settings and data.

Acceptance criteria is created by PO, business owners

Acceptance criteria are the conditions or requirements that a feature or task must meet to be considered "done" or "accepted" by the customer, user, or project team. These criteria make it clear what is expected for the feature to work as intended.

Acceptance Criteria Template:

Given [context or preconditions],
When [action or interaction],
Then [observable outcome or result].

For example, if the user story is: "As a book lover, I want to be able to search for books by title so that I can quickly find what I'm looking for," the acceptance criteria might include:

- The search should return results in less than 2 seconds.
- The search should handle and correct minor spelling errors.
- The search results should show the book title, author, and a brief summary.

Acceptance criteria are important because they give everyone a shared understanding of what "done" means for a particular feature, making it easier to test and verify the work.

- During sprint review PO accepts or rejects work done based on acceptance criteria
  - Review of process and interactions in iteration / sprint becomes Sprint retrospective , discussing what went well and identifying areas for improvement.
- Daily meetings to discuss progress and challenges become daily standup meeting
  - Daily Scrum (or Daily Stand-up): A daily meeting where the team reviews progress and discusses challenges.
  -
- Challenges faced by team / obstacles are called as Impediments
- There is no Project Manager, but the lead who sesrves and facilitates day to day work is Scrum master
- Team working in Iteration is called as Scrum Team, consists of Developers and Scrum Master
  - Scrum Master to foster an environment where:
  - 1. A Product Owner orders the work for a complex problem into a Product Backlog.
  - 2. The Scrum Team turns a selection of the work into an Increment of value during a Sprint.
- Increment is work completed at the end of sprint
- Release is 1 or more Iterations i.e. Sprints , can be one sprint in 1 release
- Product will have 1 or more releases
- Sprint backlog is list of high priority items from product backlog selected during sprint planning
- Each sprint backlog item has a user story corresponding to it
- Each user story has Acceptance criterias 3-5
- PO writes AC, with help from SME, Business users
-

PO discusses with stakeholders and decides product vision, which is broken down into product goals. Vision is long term, Product goals are relatively short term, Iteration (Sprint) level or Release (1+ Sprint) level goals

Product Goal becomes or is aligned as sprint Goal during sprint planning meeting

Sprint Planning: The team determines the work to be done during the sprint and creates sprint plan

Sprint Plan

- What will be done - Sprint PBIs selected
    - This gives sprint Goal (Summary of all user stories of SPBI)
- How it will be done
    - Estimation
    - Capacity planning
    - Creation of Sprint Backlog
-

Estimation in Agile / Scrum

- Not like SDLC where it cannot change, very rigid, done by PM using org baseline/experts using skills - expert judgment etc
- Can change
- Should be comparable to other PBI
- Mainly for comparing items rather than providing exact measurements

- Relative Estimation: Scrum uses relative estimation, often using a scale like story points, to compare the effort or complexity of different backlog items relative to each other.
- Planning Poker: The Scrum Team, including the Product Owner, Development Team, and Scrum Master, typically engages in a collaborative activity called Planning Poker. Each team member privately assigns a story point value to a user story, and then they discuss and converge on a consensus estimate.
- Fibonacci Sequence: Story points are often assigned using the Fibonacci sequence (1, 2, 3, 5, 8, 13, ...), emphasizing the exponential increase in uncertainty as complexity grows.
- Big Uncertain Small

Impediment

Impediment List: Records obstacles hindering progress.
An impediment in the context of Scrum refers to any issue, obstacle, or problem that hinders or slows down the progress of the Scrum Team.

"Definition of Done" is a list that tells you when a task is truly finished. It's like a checklist that everyone agrees on.

Sprint Review

Demo, What has been achieved, what's next, demo of Increment/working sw, 100% completed work shown, PO accepts, Acceptance criteria,
Any potential PB priority changes, etc are noted, DoD
Progress toward the Product Goal is discussed.

Sprint Retrospective

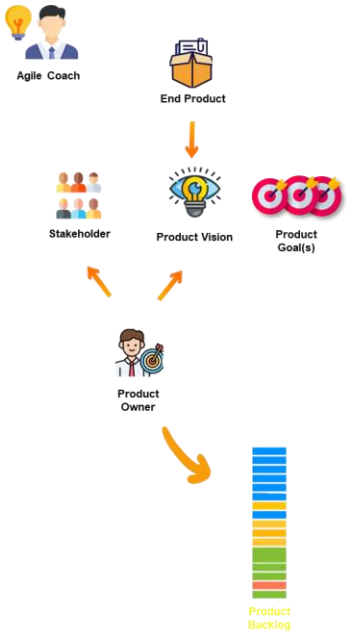
to plan ways to increase quality and effectiveness.
The Scrum Team inspects how the last Sprint went with regards to individuals, interactions, processes,
tools, and their Definition of Done.

Not S/W or PBI, but interaction,
Gather data, insights, ideas, create action items, assign responsibility
Sometimes new PBI is created for improvements for PB or next sprint - sometime basic actions that are not deliverables but improvements such as better laptop config/env etc

Scrum Events - duration for 1 month / 4 week sprint

Sprint Planning - 8 hrs
Daily Scrum - 15 mins
Sprint Review - 4 hrs
Sprint Retrospective - 4 hours

Backlog Grooming - 4 hrs = this is generally done before a sprint



Scrum BUT

The term "ScrumBut" refers to a situation where a team claims to be implementing Scrum, but makes exceptions to the standard Scrum practices for various reasons. The term is a combination of "Scrum" and "But," as in "We use Scrum, but we don't do daily stand-ups" or "We use Scrum, but our sprints are two months long."

In a ScrumBut scenario, the team might believe they are using Scrum, but by not fully adhering to its principles and practices, they often fail to realize the full benefits of the framework. ScrumButs are often seen as red flags that indicate a partial or flawed implementation of Scrum.

The concept is often used to encourage teams to closely examine their own processes and practices, to ensure they are not just using the Scrum label while failing to implement its core elements effectively.

Scrum Challenges

- New to scrum - Agile coach, Training

- Skills for new members - KT, Training
- Resistance to change - open discussion, address concerns
- Over commitments - address specific issues, educate on estimation practices, historical data
- Unclear requirements / Product Goal / Vision - Requirements management

Scrum is a lightweight framework that helps people, teams and organizations generate value through
adaptive solutions for complex problems.

General environments for Software development

Local,, DEV, Integration, QA, Pre Prod, PROD

Product Goal

- 1 Primary goal
- SMART
- Long term (Product vision is even longer term objective)
- PG is part of Product Vision (bigger picture)
- *1+ PG to be achieved to achieve PV*
- Value oriented
- Can have different goals for interconnected sub products, but 1 at prod level
- For scrum team to focus and understand value delivered
- Measurable and achievable
- As scrum executes, iteratively, with each sprint, it moves towards PG
- PG is refined in Sprint Review
- PO is responsible for PG
- PG can change over time, BUT not generally during sprint (when team is developing/working)

PO - Product Owner

Skills - Domain, Stakeholder management, Product management, communication, negotiation, Analysis

Traits - Leadership, Vision (Product / Business) , Decision Making (prioritization)

PO is individual and not group of people or committee

1 PO can be PO for 2+ teams

PO and SM can be same, but not recommended - *generally not possible*

PO from client/business - SM from Development company

Product owner responsibilities

- Set Product vision and strategy, goals
- Communication
- Stakeholder Management
- Define PB - clear requirements
- Prioritize PB - business value, market demand, end user feedback
    - Decide what comes first
- Ensure clear Acceptance criteria AC *with help of SUper Users, SME, Stakeholders*
- Attend Scrum Events*, Sprint planning, grooming - refinement, review, retrospective*
- Accept or Reject work - *based on AC*
    - Carry over to next sprint
    - Start a small unplanned sprint - hardening sprint (not sprint 0)
    - Back to backlog for reprioritization

-

Scrum Artifacts

- Product backlog => Product Goal => PO
- Sprint backlog => Sprint Backlog => ST
- Increment => DoD = Scrum Team

Product Backlog

- Backlog/Product backlog *sounds -ve term*
- Ordered list by priority (HIgh at top)
- Top PBI are clear, detailed, SMART, small (never an epic), clear scope
- Priority is decided by PO (other stakeholders) based on Value delivered
- PB Product backlog - Requirements (New features, enhancements to existing features, bug fixes)
- Maintained by POBusiness language/Domain/Easy to understand

- Mostly not in very technical language
- Any technical requirements are provided as needed
- Scrum team does not work on anything outside PB
- May never complete, there can be always requirements/EPICs or lower priority items

Sprint Backlog

- Is to achieve Sprint Goal
- Contains sprint PBI , selected by ST during SR
- Created by Developers
- Dev and PO can negotiate and can change during sprint
    - But nothing major to impact Sprint Goal

        Sprint Plan
        - What will be done - Sprint PBIs selected
            - This gives sprint Goal (Summary of all user stories of SPBI)
        - How it will be done
            - Task decomposition of 1 day or less
            - Estimation
            - Capacity planning
            - Creation of Sprint Backlog

        - Backlog/Product backlog *sounds -ve term*

Increment (completed PBI, Sprint Backlog etc)

- Value (working/usable/expected software) delivered at the end of the sprint, release, project
- Created by ST
- Verified, usable product
- Increment complies to Acceptance Criteria
- Increment complies to DoD quality standard
- SCrum team confirms when can be released
    - Actually released as per customer schedule and readiness of env

User story

As a [role], I want [goal] so that [benefit].
As a user, I want to be able to sign in to the application using my email and password so that I can access my personalized account settings and data.

- [role] represents the role of the user or stakeholder who will benefit from the feature.
- [goal] describes what the user wants to achieve or the action they want to perform.
- [benefit] explains the value or benefit the user will gain from achieving the goal.

Generally 1 user story per PBI, but for complex and interconnected PBI there can be 1+ user stories

User story is ready when it is detailed and AC is complete

Acceptance criteria

Typically one US can have 3-5 acceptance criterias
Should be specific, measurable

Acceptance Criteria Template:

Given [context or preconditions],
When [action or interaction],
Then [observable outcome or result].

Acceptance Criteria for User Story: "As a user, I want to be able to sign in to the application using my email and password."

Given that I am on the login page,
When I enter a valid email and password,
Then I should be successfully logged in and directed to my dashboard.

Given that I am on the login page,
When I enter an incorrect password,
Then an error message should be displayed indicating invalid credentials.

Given that I am on the login page,
When I enter an unregistered email,

Then an error message should inform me that the email is not associated with any account.

Behavior Driven Development

DoD != AC

"Definition of Done" is a list that tells you when a task is truly finished.
Task means any Backlog Item selected during Sprint as part of Sprint Backlog in Sprint planning meeting
It's like a checklist that everyone agrees on. For example, a task might only be "done" when it's been coded, tested, and approved by the customer. This list helps the team know exactly what they need to do to complete a task.

Definition of Done may be: "Done means coded to standards, reviewed, implemented with unit Test-Driven Development, tested with 100 percent test automation, integrated and documented."

DoD is discussed in sprint review and agreed by all

For multiple scrum teams there can be a baseline DoD (common minimum)
Can discuss and create DoD together

Eg.
- Code is complete, reviewed
- Unit test cases are created and passed
- Functional test are completed
- Documentation is completed as needed - User Guide, tech documentation
- Integration, performance testing, UAT done
- NO major bugs open

EPIC
- Very big PBI, difficult to estimate

Sprint
Sprints are the heartbeat of Scrum, where ideas are turned into value.
1-4 Weeks, ideally 2-3 weeks

Sprint 0 to start with as initial sprint - for new teams, dev/server env setup, access for team,

Hardening sprint - Code cleanup, improve quality, tech debt, final touches, release preparation

Sprint duration can be changed (between 1-4 weeks), can be decided in sprint review (or retrospective feedback), in case if any dependency on other team or hardening sprint

But cannot change duration in between for active sprint

For any reason, the sprint can be canceled and new sprint can start

Sprint can be ended early

Sprint Goal achieved early

Significant changes to requirements, requirements-SG is no longer valid for business

Any other environment factors

Scrum Team - Developers

1 SM, 1 PO (if not separate), Developers

SM, PO, Development Team-DT (Cross function, no QA in DT generally)

no hierarchy

No subteam

ST - generalist, no specialist, cross functional

self-managing, meaning they internally decide who does what, when, and how.

Size 6-8 OR 5-9 (including SM, PO)

too large, they should consider reorganizing into
multiple cohesive Scrum Teams, each focused on the same product.
Therefore, they should share the
same Product Goal, Product Backlog, and Product Owner.

E.g. if 20 developers are required then split into 3 teams, have separate or shared SM,PO

Sprint start and end dates of these to be adjusted accordingly depending on dependencies

Scrum of scrum

Shared product vision and backlog

Dependency management

Increment, environments, interfaces/APIs,

Deliverables will be dependent

E.g. Data Team to bring, clean/transform/process , Dev Team (UI/API) for web, Reporting Team
Shared Team - Component team or platform team
Specialized - DBA, Data Modellers, UX - design, DevOps, Infrastructure

Earlier called as Development Team
Creates increment
Create plan for sprint
Produces quality work
Holds everyone accountable
Adapt the plan each day towards sprint goal
Creates DoD

Scrum team don't commit to Product backlog or features, but for Sprint Goal

Impediments

An impediment in the context of Scrum refers to any issue, obstacle, or problem that hinders or slows down the progress of the Scrum Team.

Categories

1. Technical Impediments:

Issues related to tools, technology, infrastructure, or development environments that hinder the team's work.
Examples: Technical debt, unstable development environment, lack of necessary software or hardware.

2. Process Impediments:

Challenges arising from not following Scrum practices, inefficient workflows, or lack of clarity in processes.
Examples: Incomplete or unclear user stories, inconsistent sprint planning, inefficient code review processes.

3. Organizational Impediments:

Obstacles originating from the larger organization, such as policies, culture, or structural issues.

Examples: Bureaucratic approval processes, conflicting priorities from other departments, lack of support from management.

4. Communication Impediments:

Hindrances caused by misunderstandings, poor communication, or lack of information sharing among team members or stakeholders.
Examples: Misaligned expectations between team and stakeholders, insufficient communication between team members.

5. Dependencies Impediments:

Delays or challenges resulting from dependencies on external teams, departments, or third parties.
Examples: Waiting for input from other teams, external vendors not delivering on time.

6. Skill or Knowledge Impediments:

Limitations caused by team members lacking necessary skills or knowledge to complete certain tasks.
Examples: Skill gaps in the team, lack of expertise in a specific technology.

7. Team Dynamics Impediments:

Issues related to collaboration, team morale, conflicts, or ineffective teamwork.
Examples: Lack of trust among team members, conflicts impacting productivity.

8. External Impediments:

Factors beyond the team's control, such as external market changes, unexpected events, or disruptions.
Examples: Changes in regulations, global economic shifts, natural disasters affecting work environment.

Lack of Clarity in Requirements:

Impediment: The team receives unclear or ambiguous requirements, making it challenging to understand what needs to be built.

Scrum Master's Role: The Scrum Master facilitates communication between the Product Owner and the team. They help the team seek clarification from the Product Owner and encourage them to ask questions to ensure a clear understanding of the requirements.

Team Communication Issues:

Impediment: Team members have difficulty communicating effectively with each other, leading to misunderstandings and delays.

Scrum Master's Role: The Scrum Master facilitates team communication by organizing regular Scrum events (like Daily Standup) where team members share updates, identify obstacles, and collaborate on solutions. They may also facilitate team-building activities to improve relationships and communication.

External Dependencies:

Impediment: The team relies on external teams or departments for certain tasks or information, and delays from these dependencies affect the team's progress.
Scrum Master's Role: The Scrum Master helps identify and track external dependencies. They may work with other teams or stakeholders to ensure timely delivery of needed inputs. They also help the team manage expectations and adjust plans based on external factors.

Continuous Integration Challenges:

Impediment: Frequent integration and testing of code become difficult due to complex technical setups or lack of automated testing.
Scrum Master's Role: The Scrum Master advocates for practices like continuous integration and automated testing. They might help the team identify and implement tools and processes to streamline integration and testing, ensuring the team's ability to deliver a potentially shippable product increment at the end of each sprint.

Lack of Focus or Overcommitment:

Impediment: The team takes on too much work in a sprint, leading to incomplete or rushed deliverables.
Scrum Master's Role: The Scrum Master helps the team establish a sustainable pace by encouraging them to commit to a realistic amount of work in each sprint. They facilitate the team's self-

assessment and reflection during the Sprint Retrospective to address overcommitment and identify ways to improve planning.

Scrum Master

Is not a project manager, but manages and facilitates overall scrum
Interface of ST to external world
Removes impediments / obstacles - challenges faced by ST
Coach self management and cross functionality
Removing barriers between stakeholders and Scrum Teams.

Servant Leader / Leader who serves
Empowers ST
Protects ST
Stakeholder collaboration
Facilitates Collaboration within ST, between ST and PO and ST and external teams
Proactive

Scrum Events - duration for 1 month / 4 week sprint

Sprint Planning - 8 hrs
Daily Scrum - 15 mins
Sprint Review - 4 hrs
Sprint Retrospective - 4 hours

Backlog Grooming - 4 hrs = this is generally done before a sprint

- Responsible (R): The person doing the work.
- Accountable (A): The person making sure the work gets done.
- Consulted (C): People giving advice.
- Informed (I): People who need to know what's happening.

Scrum Event	Sprint Planning
Entry Criteria	Product Backlog available, priorities set
Responsible	Scrum Master
Accountable	Product Owner
Consulted	Development Team

proceeding	SG wrto PG refinement/grooming, create sprint backlog, capacity planning, task ownership, DOD, Why sprint is valuable What can be done in this sprint How will the chosen work get done
Exit Criteria	Sprint Goal defined, selected items, plan created

Scrum Event	Daily Scrum Standup
Entry Criteria	New day
Responsible	Development Team
Accountable	Scrum Master
Consulted	
proceeding	3 questions, what yesterday, what today, impediments Same time every day , morning or evening Not a status update meeting for management
Exit Criteria	Transparency, updated plan, impediments

Scrum Event	Sprint Review
Entry Criteria	End of Sprint
Responsible	Scrum Master
Accountable	Product Owner
Consulted	Development Team
proceeding	Demo, What has been achieved, what's next, demo of Increment/working sw, 100% completed work shown, PO accepts, Acceptance criteria, Any potential PB priority changes, etc are noted, DoD Progress toward the Product Goal is discussed.
Exit Criteria	Stakeholder feedback, potential backlog adjustments

Scrum Event	Sprint Retrospective
Entry Criteria	End of Sprint
Responsible	Scrum Master
Accountable	Development Team
Consulted	
proceeding	to plan ways to increase quality and effectiveness. The Scrum Team inspects how the last Sprint went with regards to individuals, interactions, processes, tools, and their Definition of Done. Not S/W or PBI, but interaction, Gather data, insights, ideas, create action items, assign responsibility Sometimes new PBI is created for improvements for PB or next sprint - sometime basic actions that are not deliverables but improvements such as better laptop config/env etc
Exit Criteria	Improvement actions identified, planned for next Sprint

Scrum Event	Sprint
Entry Criteria	After Sprint Planning and before the next Sprint starts
Responsible	Development Team
Accountable	Scrum Master
Consulted	
proceeding	Sprints are the heartbeat of Scrum, where ideas are turned into value. Only PO can cancel
Exit Criteria	Completed Increment potentially shippable

Scrum Event	Backlog refinement
Entry Criteria	Weekly
Responsible	Development Team
Accountable	Scrum Master
Consulted	
proceeding	Sprints are the heartbeat of Scrum, where ideas are turned into value. Only PO can cancel
Exit Criteria	Completed Increment potentially shippable

Velocity

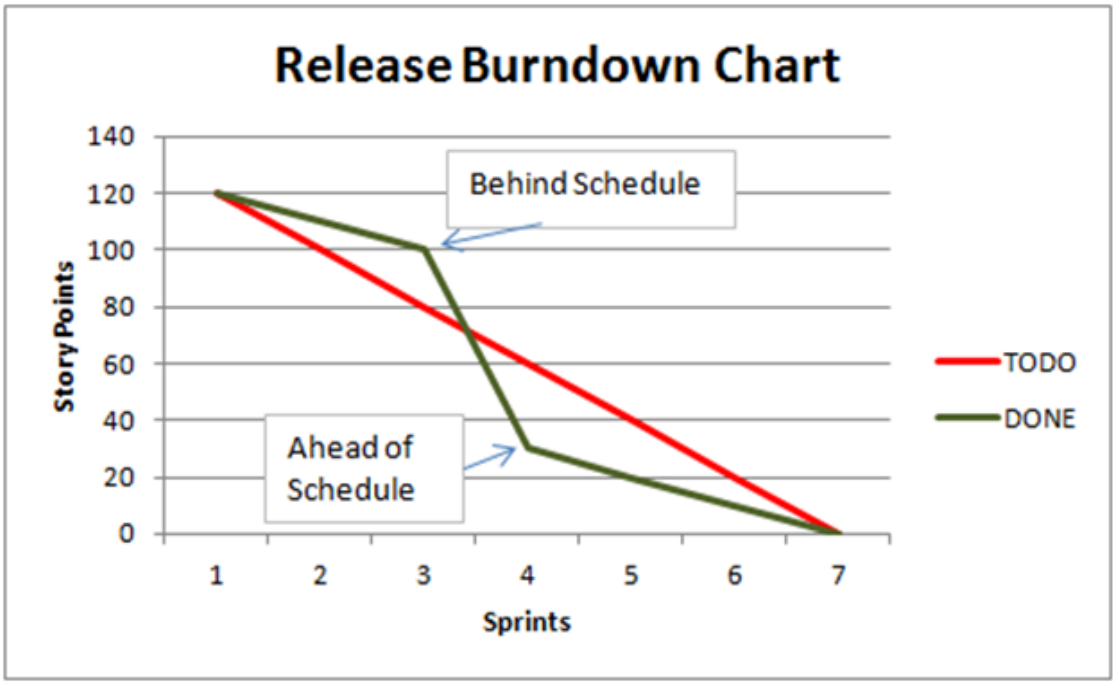
measure of a Development Team's capacity to deliver work within a sprint.

Is team specific and cannot be compared with other teams

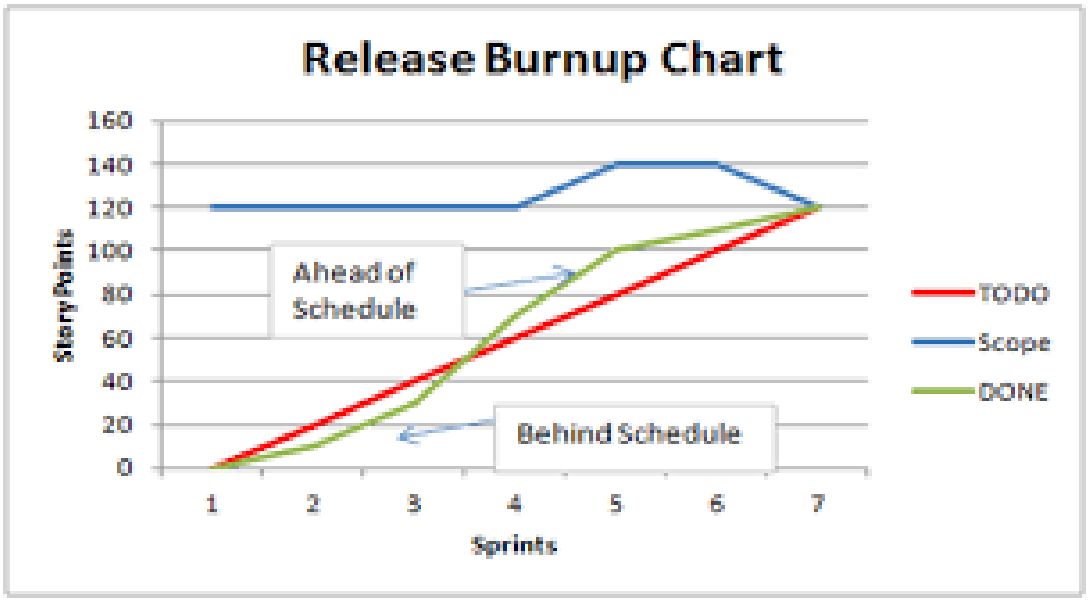
Can vary depending on team size, new member added etc

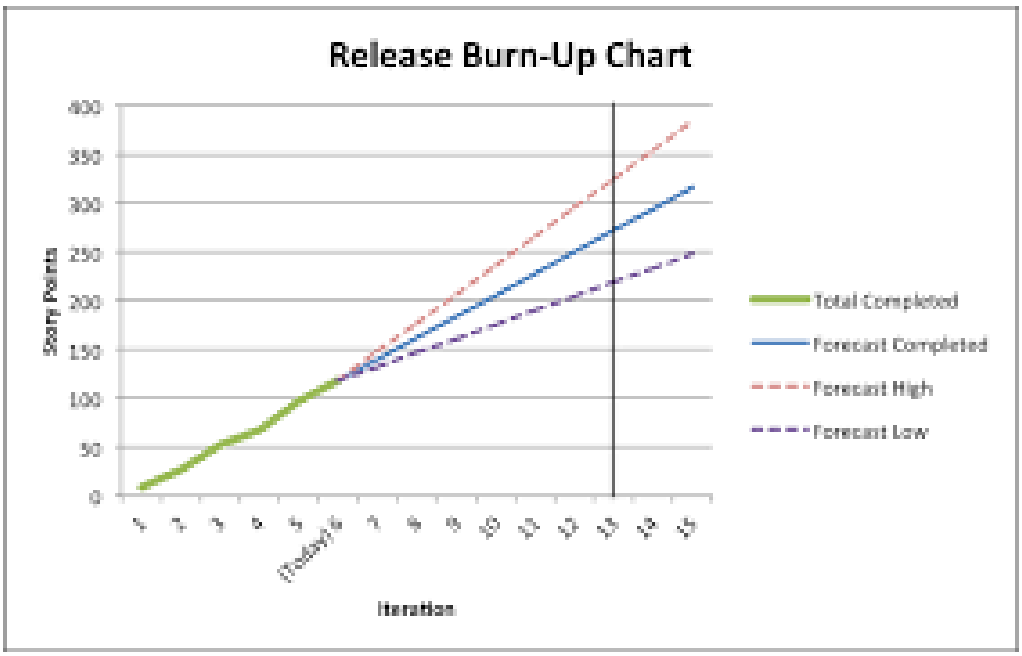
Over period of time , for stable team, velocity can increase

Burn down chart



BUrn up chart





Release

to where - Staging, QA or PROD

Sometimes it is released to QA after sprint and release moves it to PROD, but not necessarily

1+ sprint

Increment is released to QA, UAT, Pre PRod etc env

SCOPE/Goal

Product- Backlog, SPrint - SB, Release - RB, increment - DoD

QA

- Is qa part of ST or a separate QA ST shared across different development ST
- QA in same sprint or behind (current deliverables are taken up in next sprint for QA
- Dev team takes up any defects of past sprint in current along with new SB
- Where QA done, staging or QA env, when QA build is done
- Sprint demo in QA for Staging

-

UX design at the beginning at the sprint / Project / Product

Agile Manifesto

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

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

Individuals and interactions over processes and tools

In Scrum, people and how they work together are more important than just following rules or using fancy tools.
It means talking and cooperating is better than getting stuck in rigid plans or systems.
The focus is on a team's ability to solve problems together, rather than just relying on set procedures or software.

In Scrum, the value is placed on the competencies and collaboration of individuals over strict adherence to processes and tools. The focus is on human capital and team dynamics as the primary drivers of project success.

Working software over comprehensive documentation

making a working app or program is more important than writing lots of detailed plans or documents. It means it's better to have something that works, even if it's not perfect, than to have lots of papers about how it should work. It's about focusing on creating a useful product rather than spending too much time on paperwork

In Scrum, the emphasis is on delivering functional, value-driven software over generating exhaustive documentation. The priority is to produce tangible, working deliverables that fulfill user needs rather than adhering to detailed documentation protocols.

Customer collaboration over contract negotiation

Talking and working with customers is more important than just sticking to what a contract says. It's better to understand what the customer really needs and change plans to meet those needs.

In Scrum, fostering customer collaboration takes precedence over contract negotiation to ensure alignment with end-user needs. The focus is on building a partnership that is responsive to customer requirements rather than being constrained by contract terms.

Responding to change over following a plan

In Scrum, being able to change and adapt is more important than strictly sticking to a plan. It's better to adjust to new situations than to just do what was planned at the star

In Scrum, agility and adaptability are prioritized over rigid adherence to initial plans. The methodology values the capacity to pivot in response to evolving requirements or unforeseen challenges.

Agile Principles

1. Customer Satisfaction: Deliver valuable software early and continuously.
2. Welcome Change: Embrace changes in requirements, even late in the project.
3. Frequent Delivery: Deliver working software frequently, with a preference for a shorter timescale.
4. Collaboration: Business and developers must work together daily.

5. Motivated Teams: Build projects around motivated individuals and give them the support and trust they need.
6. Face-to-Face Communication: The most efficient way to convey information is through face-to-face conversation.
7. Working Software: is the primary measure of progress.
8. Sustainable Pace: Maintain a sustainable pace of work for the team.
9. Technical Excellence: Pursue technical excellence and good design.
10. Simplicity: Focus on what's necessary and do the simplest work that could possibly be done.
11. Self-Organizing Teams: Allow teams to self-organize around their work.
12. Reflect and Adjust: Regularly reflect on performance and adjust as necessary.

1. Highest Priority is to Satisfy the Customer
    - Customer Satisfaction: Deliver valuable software early and continuously.
    - Make the customer happy by giving them useful stuff quickly.
    - Keep giving them updates that they like.
2. Welcome Changing Requirements
    - Welcome Change: Embrace changes in requirements, even late in the project.
    - Don't worry if plans change, even late in the project.
    - Adapt and make those changes.
3. Deliver Working Software Frequently
    - Frequent Delivery: Deliver working software frequently, with a preference for a shorter timescale.
    - Finish useful parts of the project quickly.
    - Show these to the customer often.
4. Work Together Daily
    - Collaboration: Business and developers must work together daily.
    - Customers, managers, and developers should talk every day.
    - This helps everyone stay on the same page.
5. Motivated Individuals
    - Motivated Teams: Build projects around motivated individuals and give them the support and trust they need.
    - Trust your team and give them what they need.
    - They will do their best work if they're happy.
6. Face-to-Face Conversation
    - Face-to-Face Communication: The most efficient way to convey information is through face-to-face conversation.
    - Talking to people directly, it's the best way to share ideas.
    - Use video or meet in person if you can.
7. Working Product is the Main Measure

    - Working Software: is the primary measure of progress.
    - The best way to see if it's going well is to have something that works.
    - Focus on making a working thing.
8. Sustainable Pace
    - Sustainable Pace: Maintain a sustainable pace of work for the team.
    - Work hard, but don't get too tired.
    - You should be able to keep the pace long-term.
9. Excellence and Good Design
    - Technical Excellence: Pursue technical excellence and good design.
    - Make things as simple as possible, but make sure they work well.
    - Good design helps in the long run.
10. Simplicity is Essential
    - Simplicity: Focus on what's necessary and do the simplest work that could possibly be done.
    - Do only what is needed and no more.
    - Keep it simple to move fast.
11. Self-Organizing Teams
    - Self-Organizing Teams: Allow teams to self-organize around their work.
    - The team knows best how to get the job done.
    - Let them make decisions.
12. Regularly Reflect and Adjust
    - Reflect and Adjust: Regularly reflect on performance and adjust as necessary.
    - Take time to see what is and isn't working.
    - Change things to work better.

Agile: A {flexible | Lightweight (*compared to SLDC)* approach to software development that emphasizes customer feedback, collaboration, and delivering small, valuable increments frequently.

Scrum: A specific framework for implementing Agile principles, organizing work into short cycles called sprints, with roles like Scrum Master and Product Owner, and events like daily stand-ups and sprint reviews.

Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems.

Based on empirical lean practices

The three pillars of Scrum are:

- Transparency: All team members should have a clear and shared understanding of the work, progress, and goals. This is achieved through practices like the daily stand-up meeting and the use of visual aids like Scrum boards.
- Inspection: Regularly inspecting the work and progress helps identify any discrepancies between the expected and actual outcomes. This is done through events like sprint reviews and retrospectives.
- Adaptation: If the team identifies any issues or areas for improvement during the inspection, they should adapt their practices and processes accordingly. This is a key part of the continuous improvement mindset in Scrum.

  Transparency through three artifacts, > enables Inspection
  Inspection through five events > enables adaptation
  Adaptation adjusts process (outside limits) or product (unacceptable)
  Adaptation as soon as possible to minimize further deviation

- Transparency: Making the work and progress visible to everyone involved.
- Inspection: Regularly checking the work and progress to identify any issues.
- Adaptation: Making changes based on the inspection to improve the work and progress.

visual tools commonly used in Scrum:

1. Product Vision Board is used to capture and communicate the high-level vision, goals, and key elements of a product. It helps align the Scrum Team and stakeholders on the overall direction and purpose of the product.
    a. Vision - purpose of creating product ? What +ve change it brings ?
    b. Target - Market Segment, target customer, user
    c. Needs - what problem does the product solve, which benefit it provides ?
    d. Product - What product is it, what makes it stand out, is it feasible to develop the product ?
    e. Business Goals - HOw is the product going to benefit the company, what are business goals
2. Scrum Board: Visualizes tasks and progress in a project.
    a. Like kanban board
    b. Backlog, Sprint, todo, in progress, verify, done
3. Product Backlog: List of desired features and improvements.
    a. Backlog items, priority, release #, Sprint #, Tentative Date - top items

4. Sprint Backlog: Tasks chosen for a specific work period.
    a. - is part of Scrum board
    b. May have other parameters for SPBI
5. Burn-down Chart: Tracks work remaining in a sprint.
6. Burn-up Chart: Illustrates work completed over time.
7. Velocity Chart: Measures team's productivity in sprints.
8. Cumulative Flow Diagram (CFD): Displays work stages and bottlenecks.
9. Task Board: Organizes tasks and workflow visually.
10. Release Burndown Chart: Tracks progress towards completing a release.
11. Sprint Goal: Clear objective for a sprint's outcome.
12. Definition of Done (DoD): Agreed-upon quality criteria for a task.
13. Impediment List: Records obstacles hindering progress.
14. Gantt Chart: Displays project schedule and tasks over time.

Scrum values are a set of guiding principles that underpin the Scrum framework, promoting collaboration, focus, and adaptability. They serve as a foundation for effective teamwork and product development.

Official Definition:

The Scrum values, as described in the Scrum Guide, are **Courage**, **Focus**, **Commitment**, **Respect**, and **Openness**. These values support the Scrum principles and help individuals and teams work together to achieve their goals.

The five values of Scrum are:

1. Commitment: Each team member commits to achieving their team's goals and delivering high-quality work.
2. Courage: The team has the courage to question the status quo, address tough issues, and make difficult decisions.
3. Focus: The team focuses on completing the tasks in the current sprint and achieving the sprint goal.
4. Openness: The team is open about their work, progress, and challenges, fostering transparency and collaboration.
5. Respect: Team members respect each other's skills, opinions, and contributions, creating a positive and supportive work environment.

here are examples of the Scrum values in the context of an IT software project:

1. Commitment: Each team member commits to achieving their team's goals and delivering high-quality work.

   Example: The team commits to completing a certain number of user stories within the sprint, and they work together to fulfill that commitment.

2. Courage: The team has the courage to question the status quo, address tough issues, and make difficult decisions.

   Example: A team member speaks up when they foresee a potential risk to the project, even if it's uncomfortable.

3. Focus: The team focuses on completing the tasks in the current sprint and achieving the sprint goal.

   Focus: The team decides to postpone a discussion about a future feature until the next sprint planning meeting, so they can stay focused on completing the current sprint's tasks.

   Example: The team avoids multitasking during a sprint to ensure they complete high-priority user stories.

4. Openness: The team is open about their work, progress, and challenges, fostering transparency and collaboration.

   Openness: During the sprint retrospective, the team openly shares what went well and what didn't, including any obstacles they faced, so they can learn and improve.

   Example: The team openly discusses challenges they faced during the sprint retrospective, enabling them to collectively find solutions.

5. Respect: Team members respect each other's skills, opinions, and contributions, creating a positive and supportive work environment.

   Respect: When a team member struggles with a task, instead of criticizing them, the team offers support and works together to find a solution.

   Example: Team members actively listen to each other's opinions during discussions and appreciate diverse viewpoints.

stories that you can use during your Scrum training to illustrate the benefits and challenges of Agile Scrum implementation in the IT industry:

1. The Turnaround Project: A software company was struggling with a project that was behind schedule and over budget. They decided to adopt Scrum, breaking the project into sprints and prioritizing features. Within a few sprints, the team delivered a minimum viable product (MVP) to the client, who was thrilled. The team continued to add features and improvements in subsequent sprints, ultimately delivering a successful project.
2. The Unhappy Customer: A team was using Scrum but wasn't involving the customer in their sprint reviews. The customer was unhappy with the final product, as it didn't meet their needs. The team learned the importance of involving the customer throughout the process and adjusted their approach for future projects.
3. The Overwhelmed Team: A Scrum team was consistently failing to complete their sprint commitments. The Scrum Master realized the team was overcommitting and helped them focus on a realistic number of tasks. The team started completing their sprints successfully and felt a renewed sense of accomplishment.
4. The Resistant Team Member: A team member was resistant to adopting Scrum, feeling it was just another management fad. However, as the team began to see the benefits of Scrum, such as improved communication and faster delivery, the resistant team member became one of Scrum's biggest advocates.
5. The Pivot: A software startup was developing a new app using Scrum. After a few sprints, they realized the market for their app was smaller than anticipated. They quickly pivoted and adjusted the app's features to target a larger market. The flexibility of Scrum allowed them to make this change without losing much time or money.
6. The Missed Opportunity: A team was using Scrum but wasn't conducting retrospectives. Over time, the team's performance plateaued, and they missed opportunities for improvement. They realized the importance of retrospectives and started conducting them regularly, leading to continuous improvement and increased performance.