Introduction to Agile Development and Scrum

Module 1: Introduction to Agile and Scrum

- Agile = mindset, not just a process
- Requires thinking differently about team and project management
- Leadership must focus on quick, iterative value delivery
- Success needs a cultural change
- Agile can't be learned only from books
  - Like learning to drive: requires practice with an experienced guide
- Course includes:
  - What to do and what to watch out for
  - Patterns and anti-patterns
  - Importance of small, co-located, cross-functional teams
  - Empower teams to self-manage for faster decisions
  - Agile planning with software development examples
  - Lean manufacturing concepts in Agile and Scrum
  - End of iteration: ask "pivot or persevere?"
- Course tools:
  - Videos
  - Quizzes
  - Discussion forums
  - Hands-on final module to build planning skills
- Practice improves skill
- Challenge yourself to complete final module
- Build real agile planning experience

Agile Principles

- Agile = iterative approach to project management
- Teams deliver value quickly, respond to feedback
- Plan small increments, not whole year
- Adaptive planning = plan a little, deliver, adjust
- Evolutionary development = build in small pieces, evolve over time
- Early delivery = critical, put product in customer's hands
- Feedback helps decide: pivot or persevere
- Continuous improvement = improve team + product with feedback
- Responsive to change = replan quickly as requirements shift
- Agile Manifesto values:
  - Individuals and interactions > processes and tools
  - Working software > comprehensive documentation
  - Customer collaboration > contract negotiation
  - Responding to change > following a plan
- Items on right have value, items on left valued more
- Don't lose focus: working software is the goal
- Documentation and contracts still exist, but collaboration and delivery matter more

- Agile software development = iterative + follows Agile Manifesto
- Emphasizes:
  - Flexibility
  - Peer/customer interaction
  - Transparency = everyone knows what others are doing
  - Responsibility to deliver value
- Teams = small, co-located, cross-functional, self-organizing, self-managing
- Key takeaway: build what's needed, not what was planned
- If customer doesn't like the plan, change it
- Agile = build based on changing needs, not fixed plans
- Agile = iterative, collaborative approach using small teams
- Focus on flexibility, interactivity, transparency
- Core values = individuals, working software, collaboration, response to change

Methodologies Overview

- Waterfall = traditional, structured, step-by-step development
  - Phases:
    - Requirements → gather + document all requirements
    - Design → architects design full system based on requirements
    - Coding → devs code entire system, often in isolation
    - Integration → modules combined, first time checking if they work together
    - Testing → test full system, bugs go back to coding or design
    - Deployment → hand off to operations to run
  - Problems with Waterfall:
    - No provision for change once a phase ends
    - No intermediate delivery → only see results at end
    - Sequential flow = opportunity for lost info/blockers between phases
    - Late-found mistakes = very costly to fix
    - Long lead times → slow delivery
    - Teams work in silos → unaware of impact on others
    - Ops team runs code they didn't build → inefficient
- Extreme Programming (XP) = iterative method, introduced in 1996 by Kent Beck
  - Tightly nested feedback loops:
    - Release plan → months
    - Iteration plan → weeks
    - Acceptance tests → days
    - Stand-ups → daily
    - Pair negotiation → hours
    - Unit tests → minutes
    - Pair programming → seconds
  - Goals = improve software quality + be responsive to change
  - Recognized as one of the first Agile methods
  - XP Values:
    - Simplicity → do only what's needed, no extra
    - Communication → high team interaction, know what others are doing
    - Feedback → continuous feedback to guide progress
    - Respect → all ideas valued equally, peer collaboration
    - Courage → honest estimates, commit realistically
- Kanban = from Japanese manufacturing, means "billboard"
  - Focus = continuous flow of work
  - Core Principles:

        - Visualize workflow → make all work visible
        - Limit work in progress → avoid overload, focus on fewer tasks
        - Manage/enhance flow → optimize and improve work movement
        - Make process policies explicit → everyone understands how work is done + “done” definition
        - Continuously improve → use feedback to refine process

Working Agile

- Work in small batches
    - From lean manufacturing
    - Avoid big batches to reduce waste
    - Example: 1000 brochures in batches of 50 = 16 mins for first product
    - Delay in detecting errors (e.g., typo, no glue)
    - Single piece flow = 24 seconds to first product
    - Faster feedback, quicker pivots, less waste
- Minimum Viable Product (MVP)
    - Not Phase 1 or beta
    - Minimum to test a hypothesis and gain learning
    - Focus on learning, not delivery
    - End of each MVP: pivot or persevere
    - Example:
        - Wrong: wheel → chassis → car
        - Right: skateboard → steering → pedals → motorcycle → convertible
    - Customer gets what they *desire*, not just what they *asked for*
- Behavior Driven Development (BDD)
    - Test from outside-in
    - Integration/UI level
    - Based on customer’s view
    - Common syntax for devs and stakeholders
    - Feature file:
        - As a [role]
        - I need [function]
        - So that [business value]
    - Gherkin syntax:
        - Given [preconditions]
        - When [event happens]
        - Then [observable outcome]
- Test Driven Development (TDD)
    - Test from inside-out
    - Unit testing level
    - Focus on module inputs/outputs
    - Write test *before* code
    - Workflow:
        - Write test
        - Test fails (Red)
        - Write code to pass test (Green)
        - Refactor
    - Red → Green → Refactor
- Pair Programming
    - Two programmers, one task
    - One codes, one reviews/thinks/looks up
    - Switch every ~20 minutes
    - Higher quality code, fewer production bugs

- Cheaper than debugging later
- Good for mentorship
  - Senior + junior
  - New + familiar with code
- Improves team skill and code knowledge

Introduction to Scrum Methodology

Scrum Overview

- Agile is a philosophy, not prescriptive
- Scrum is a methodology, prescriptive
- Scrum is for incremental product development
- Emphasizes small, cross-functional, self-managing teams
- Provides roles, rules, and artifacts
- Uses fixed-length increments called sprints
- Goal: build a potentially shippable increment each sprint
- Get things in customers' hands early
- Easy to understand, hard to master
- Not many rules, but difficult to do well
- Like ballet—requires muscle memory, experience
- If new, get someone experienced to guide you
- Sprint = one iteration through design, code, test, deploy
- Mini software delivery lifecycle
- Each sprint needs a goal
- Everyone should know what the increment should do
- Typical sprint length: 2 weeks
- 4 weeks = too long, too much can change
- 1 week = too short for most
- 2 weeks = good balance
- Product backlog = list of all user stories, to-do list
- Backlog refinement = groom stories to be sprint ready
- Sprint planning = select stories for sprint
- Sprint backlog = only stories for next sprint (subset of product backlog)
- Daily Scrum (stand-up):
  - What did you do yesterday?
  - What are you doing today?
  - Anything blocking you?
- Goal: have a valuable shippable increment at end
- Agile is iterative
- Repeat: design → code → test → deploy
- Must include deploy to get feedback
- Each sprint: make plan → do work → deploy → get feedback → input for next plan
- Scrum = Agile methodology
- Scrum provides structure: roles, meetings, rules, artifacts
- Uses small cross-functional, self-organizing teams
- Uses fixed-length iterations (sprints)
- Produces shippable product increment each iteration

The 3 Roles of Scrum

- Three Scrum roles: Product Owner, Scrum Master, Scrum Team
- Product Owner
  - Has the vision
  - Calls the shots, pivot or persevere
  - Role, not job title (not same as product manager)

    - Represents stakeholder interests
    - Go to product owner, not stakeholder, for questions
    - Stakeholder = person funding work (checkbook)
    - Liaison between team and stakeholders
    - Articulates product vision to team
    - Final arbiter of requirements
    - Reprioritizes product backlog
    - Grooms stories, adds detail, ensures sprint readiness
    - Accepts or rejects product increment
    - Decides to ship or not
    - Decides to continue development or pivot
- Scrum Master
    - Facilitates Scrum process
    - Smartest person in room about Scrum
    - Experienced Scrum Master recommended for new teams
    - May serve multiple experienced teams
    - Coaches the team, Agile mentor
    - Creates self-organizing environment
    - Team commits to sprint increment
    - Shields team from external interferences
    - Directs stakeholders to go through Scrum Master
    - Resolves impediments (top priority)
    - Takes impediments, team moves on
    - Enforces time boxes (15-min stand-up, 2-week sprint)
    - Captures empirical data
    - Adjusts forecasts
    - No management authority
    - Not a manager, team must be able to confide in them
- Scrum Team
    - Cross-functional team
    - Includes developers, testers, analysts, ops, etc.
    - Not just software engineers
    - Self-organizing, equal roles
    - Self-managing, assign work to themselves
    - Use kanban board to pull work
    - Small team (7 ± 2 or 5 ± 2)
    - Co-located if possible
    - More successful when co-located or in small geo clusters
    - Avoid 1-person isolated in time zone
    - Dedicated, long-term, full-time members
    - Not on multiple projects
    - Negotiate commitments with product owner per sprint
    - No long-term commitments
    - Autonomy on how to reach commitments
    - Told what to do, not how to do it
    - Let them figure it out themselves
- Summary
    - Product owner: represents stakeholders, sets priorities, owns requirements, vision, and readiness to ship
    - Scrum master: coaches team, protects team, resolves blocks, enforces Scrum
    - Scrum team: small, co-located, cross-functional, self-managing, dedicated, sprint-by-sprint commitments, full autonomy on how to deliver

Artifacts, Events and Benefits

- product backlog
  - list of all stories you're ever going to want to do
  - all requirements for the product not done yet
  - stories not in the current sprint
  - sometimes includes icebox and release backlog
- sprint backlog
  - stories for the next sprint (next two weeks)
  - on deck to be executed
- done increment
  - product increment completed by end of sprint
- sprint planning meeting
  - plan out the sprint
  - product owner, Scrum master, entire team
  - team commits to stories for next sprint
- daily Scrum / stand-up
  - happens every day
  - same time, same place
  - sync on what's done, what's next, any impediments
- sprint
  - two weeks of working time
- sprint review
  - demo time
  - show stakeholders new features, advancements
- sprint retrospective
  - reflect on what went well
  - what didn't go well
  - what can be changed for the future
- benefits of Scrum
  - higher productivity
    - daily meetings
    - Kanban board
    - self-managing team
  - better quality
    - engaged team
    - behavior/test driven development
    - constant testing
  - reduced time to market
    - working in small increments
    - done increment can be tested/released sooner
  - increased stakeholder satisfaction
    - stakeholders see results faster
  - better team dynamics
    - transparency
    - shared knowledge
    - help when someone is stuck
  - happier employees
    - in control
    - pull tasks off backlog
    - commit one sprint at a time
- Scrum vs Kanban

Category	Scrum	Kanban

cadence	fixed length sprints (2 weeks)	continuous flow
release	at end of sprint	continuous delivery
roles	product owner, Scrum master, team	no defined roles
key metric	velocity	cycle time
change philosophy	avoid change during sprint	change can happen anytime

- Scrum artifacts
  - product backlog
  - sprint backlog
  - done increment
- Scrum events
  - sprint planning
  - daily Scrum
  - sprint
  - sprint review
  - sprint retrospective
- benefits of Scrum
  - increased productivity
  - improved product quality
  - reduced time to market
  - increased stakeholder satisfaction
  - better team dynamics
  - happier employees

Organizing For Success

Organizational Impact of Agile

- proper organization is critical to success
  - existing teams may need to be reorganized
  - can’t just make old teams Agile
- Conway’s Law
  - system design copies organization’s communication structure
  - ask 4 teams → get 4-pass compiler
  - UI/app/database teams → build 3-tier architecture
- team alignment
  - teams should be loosely coupled
  - teams should be tightly aligned
  - each team has a mission aligned with business
  - break into small teams by business area
    - order team, accounts team, Shopkart team, recommendation team
  - team owns that business area
- end-to-end responsibility
  - build it, run it, debug it in production
  - team responsible for full lifecycle
- long term mission
  - avoid pulling people on/off projects
  - long term mission builds ownership
- autonomy is critical
  - autonomy is motivating
  - motivated people build better stuff

    - autonomy is fast
        - decisions happen at team level
        - no waiting for top-down decisions
        - fewer handoffs
- wall of confusion (Andrew Clay Schafer)
    - development wants change
    - operations wants stability
    - opposing goals → conflict
    - if these views exist, Agile won't deliver benefits
- real project example
    - Agile team started in January
    - working in sprints
    - deploy request in mid-February
    - ops said "open a ticket"
    - ticket delayed for months
    - deployed in September
    - left project in December
- moral of story
    - if entire organization isn't Agile, benefits are lost
    - DevOps is about making ops Agile too
    - adopt DevOps to get full Agile benefits
    - otherwise, Agile teams wait for ops
- Agile and DevOps alignment

Agile Goal	DevOps Goal
deliver software faster	accelerate time to market
responsive to change	align IT with business for value
obtain higher quality	increase IT productivity

        • full benefit of Agile → adopt DevOps too
    - make ops as Agile as development
- summary (5:13)
    - organization affects system design
    - autonomy → motivated, faster teams
    - without full adoption, get bottlenecks
    - Agile and DevOps goals are aligned

Mistaking Iterative Development for Agile

- common pitfall
    - companies claim they're Agile
    - project starts with studying and approval
    - heavy design and planning upfront
    - called the "fuzzy front end"
    - looks like waterfall
- development phase
    - iterative development
    - just iterating on the plan
    - not Agile
    - no feedback from customer
    - nothing deployed
    - no pivot or persevere decisions
    - just following the plan

- last mile
  - deployment takes forever
  - first time everything integrated
  - ops team struggles to get it working
- Water-Scrum-Fall
  - planning → “Scrum” → long deployment
  - not real Scrum
  - just iterative development
  - not being Agile
- real Agile requires
  - fast feedback
  - responsiveness to change
  - frequent deployment
- what Agile is not
  - not just iterative lifecycle
  - not mini Waterfall
  - planning everything upfront = not Agile
  - not just devs working in sprints
  - Agile team = devs + testers + BAs + maybe ops
  - cross-functional teams
- Agile Manifesto
  - no Agile project manager
  - project manager doing command/control = not Agile
- summary (2:07)
  - many companies follow Waterfall but call it Agile
  - iterative development ≠ Agile
  - must be responsive to change
  - must deliver value often
  - Agile PM role is different
  - teams are self-managed
  - self-managed teams assign work to themselves

Module 2: Agile Planning

Planning to be Agile

Destination Unknown

- Douglas Adams quote
  - "I love deadlines. I love the whooshing sound they make as they fly by."
- common issue
  - set deadlines
  - miss them
  - question: why?
  - more important: how to avoid?
- navigating the unknown
  - like walking through penguins
  - can plan first few steps
  - middle unclear
  - penguins move—just like software changes
  - systems, packages get patched—keeps changing
  - better vantage point as you move forward
  - don’t decide everything at start

  - beginning = when you know the least
- common mistake
  - plan everything at beginning
  - know very little then
  - act like we can predict the end
- Agile approach
  - iterative planning
  - plan only what you know
  - adjust plan as you learn more
  - improves estimate accuracy
- estimating
  - 3 months out = maybe 50% accurate
  - 2 weeks out = almost 100% accurate
- key message
  - you’re not omnipotent
  - don’t plan everything up front
  - plan as you go
  - update plan with new info
  - get more accurate as you go
- summary
  - upfront planning → missed deadlines
  - iterative planning → course correction
  - leads to more accurate estimates

Agile Roles and the need for Training

- placing people in new roles without training → failure
  - product manager becomes product owner
  - project manager becomes scrum master
  - dev team becomes scrum team (but only has engineers)
- product manager vs. product owner
  - product manager = job title
  - product owner = scrum role
  - product manager → manages budget
  - product owner → visionary, leads experiments toward sprint goal
  - product manager = ops focus
  - product owner = conduit from stakeholders to team
  - different skill sets
  - some can do both, some cannot
  - job title ≠ role
- project manager vs. scrum master
  - project manager = task manager, manages plan
  - scrum master = coach, supports self-managing team
  - PM wants to assign work
  - Agile teams assign work to themselves
  - PM = documents risks in spreadsheet
  - scrum master = removes impediments, buffers team
  - PM: "What will you do about it?"
  - SM: "I’ll handle it, you keep working"
  - different mindset, different job
  - managing to plan ≠ enabling agility
- development team vs. scrum team
  - development team = software engineers only
  - scrum team = cross-functional

    - includes devs, testers, ops, analysts, security
    - goal: build complete increment
    - must reorganize into cross-functional teams
- Bill Cantor quote
    - “Until and unless business leaders accept the idea that they are no longer managing project with fixed functions, timeframes, and costs as they did with Waterfall, they will struggle to use Agile as it was designed to be used.”
- roles have changed
    - can’t put people in new roles with no training
    - PM will try to turn Kanban board into Gantt chart
    - mindset must change
    - must come from upper management
- leadership shift
    - stop asking “What by end of year?”
    - ask “What in next 2 weeks?”
    - “How will you delight customers next sprint?”
- summary
    - new Agile roles need different focus and priorities
    - proper training required
    - Agile mindset must be adopted by management

Kanban and Agile Planning Tools

- tool won’t make you agile
    - Agile mindset required first
    - tools support process, don’t create it
    - Kanban ≠ Gantt chart
    - Gantt = traditional project management
    - Kanban = Agile process
    - many Agile tools, most do same thing
    - some overcomplicated (tasks, subtasks = micromanaging)
    - prefer epics and stories only
    - ZenHub = preferred tool
- ZenHub
    - plugin for GitHub
    - adds Kanban board to GitHub
    - devs stay in GitHub
    - customizable pipelines (columns)
    - can be simple or complex
- why use ZenHub
    - uses GitHub issues
    - no extra tools to update
    - reduces outdated statuses
    - status shown in Kanban board
    - shows: what’s left, what’s in progress, who’s working, what’s done
    - quick project status
    - one version of truth
    - issues = stories
    - devs open/close issues in GitHub
    - no switching tools
    - all in one place
- what is a Kanban board
    - things to do
    - things doing
    - things done

    - simple, visual progress tracking
    - move stories across board
    - physical example: whiteboard + sticky notes
    - real Kanban = sticky notes move through columns
    - simple = best
- default ZenHub pipelines
    - **New Issues**
        - like inbox
        - default place for new items
        - triage first
        - move or reject, don't leave too long
    - **Icebox**
        - ZenHub-specific
        - cold storage for long-term ideas
        - keeps backlog clean
        - used for future, not active work
    - **Product Backlog**
        - everything you want to do
        - not being worked on yet
        - remove long-term stuff to icebox
    - **Sprint Backlog**
        - work for next two weeks
        - pulled from product backlog
        - devs focus only here
    - **In Progress**
        - active work
        - dev assigns story to self
        - avatar shows who's doing what
    - **Review/QA**
        - after work is done
        - pull request created
        - GitHub + ZenHub can automate
        - signals others to help review
    - **Done**
        - dev finished work
        - code merged
        - not accepted by product owner yet
        - sprint review handles acceptance
- workflow
    - left to right
    - new stories in on left
    - done increment out on right
- summary
    - Kanban = track planned, in-progress, and done items
    - pipelines = columns
    - work flows left to right as completed

User Stories

Creating Good User Stories

- Define a user story:
    - Piece of business value delivered in a done increment
    - More than a requirement (includes who, what, why)

- Good user story includes:
  - What it is
  - Who needs it
  - Business value
  - Assumptions (e.g., using relational DB)
  - Details to guide dev
  - Definition of done = acceptance criteria
- Importance of acceptance criteria:
  - Avoid misunderstandings at sprint review
  - Use Gherkin syntax (Given, When, Then)
  - Makes behavior testable and clear
- Story description format:
  - As a [role]
  - I need [functionality]
  - So that [business benefit]
- Helps prioritize backlog by business value
- Assumptions and details:
  - Not too much detail, just enough
  - Help dev understand expectations
  - Include dependencies, constraints (e.g., opt-in required)
- Acceptance Criteria / Definition of Done:
  - Use Gherkin format
  - Given [precondition]
  - When [action]
  - Then [expected outcome]
  - Testable behavior for story completion
- Sample story example:
  - Role: marketing manager
  - Need: list of names/emails
  - Benefit: send marketing promotions
  - Assumptions: emails exist, users opted in
  - Acceptance criteria:
    - Given 100 customers, 90 opted in
    - When list is requested
    - Then see 90 emails
- INVEST acronym:
  - Independent – can be ranked, moved
  - Negotiable – details can change
  - Valuable – delivers value to user
  - Estimable – size can be estimated
  - Small – fits in a sprint
  - Testable – has clear done definition
- Epics:
  - Used for big ideas
  - Story too large for sprint = Epic
  - Epics broken into smaller stories
  - Common when new ideas first arrive
  - Refined later for sprint planning
  - Hierarchy: Epic > Stories
  - Can't estimate = break into stories

Effectively using Story Points

- story points estimate difficulty to deliver and implement a user story
- abstract measure

- avoid anti-patterns
- estimate includes:
  - effort: how hard to do
  - complexity: how complex or simple
  - uncertainty: done it before or not
- humans bad at estimating wall clock time
- don't use hours, days, weeks
- story points like t-shirt sizes
  - small, medium, large, extra-large
  - compare with other stories in sprint
- use Fibonacci sequence for numeric values
  - 3 = small
  - 5 = medium
  - 8 = large
  - 13 = extra-large
  - don't go too granular
  - keep it abstract
- team must agree what medium (5) means
- compare other stories to the "5" story
- example analogy: building sizes
  - if one is a 5, next to it might be 3 or 8
  - just relative size
- story should be small
  - doable in a couple of days
  - not drag on
- large stories → break into smaller ones
  - if 21 → too large → split into smaller stories
  - maybe track with an epic
- anti-patterns:
  - story points = wall clock time
    - e.g., 1 = 1 day, 3 = 3 days
    - ultimate wrong thing to do
  - story points ≠ hours, days, weeks
  - avoid by all means
- summary:
  - story points = estimate difficulty
  - story points = relative, like t-shirt sizes
  - team must agree what average means
  - never equate story points with time

Building the Product backlog

- define product backlog
- describe how to assemble product backlog
- convert requirements into stories
- product backlog = all unimplemented stories
- not in a sprint, waiting to be worked on
- ranked order, top more accurate than bottom
- only need to rank next sprint or two
- top = business importance
- top stories = more detail
- bottom stories = fuzzier, details added later
- example: building a hit counter service
  - give counter a name

    - increment 1, 2, 3, 4
    - ask count → get number 4
- requirement: allow multiple counters
    - count multiple things
- requirement: counters persist across restarts
    - need database, not in-memory
- requirement: reset counter to zero
- kanban board (ZenHub) columns = pipelines
    - New Issues
    - Icebox
    - Product Backlog
    - Sprint Backlog
    - In Progress
    - Q&A
    - Review/Q&A
    - Done
- only focus on New Issues
- creating new stories
    - use story template:
        - "As a ___, I need ___, so that ___"
- first story: service for counting things
    - as a user
    - I need a counter service
    - so that I can track how many times something is done
- second story: allow multiple counters
    - as a user
    - I need multiple counters
    - so that I can track several counts at once
- third story: persist across restarts
    - as a service provider
    - I need to persist last known count
    - so that users don't lose counts after restart
- fourth story: reset counters
    - as a system administrator
    - I need to reset a counter
    - so that I can redo counting from the start
- specifying role = shows who benefits
- "I need" = shows what they need
- "so that" = shows business value
- move stories from New Issues to backlog
- use New Issues as inbox
- prioritizing backlog:
    - counting service → top of product backlog
    - multiple counters → do later
    - persistence → after MVP working
    - reset → after persistence
- customer gave one-liner requirements
- convert using story syntax:
    - as a ___
    - I need ___
    - so that ___
- rank in priority order for execution
- summary:

  - product backlog = ranked list of unimplemented stories
  - top stories = more detail
  - use “as a, I need, so that” template
  - helps define role, need, business value

The planning Process

Backlog Refinement: Getting Started

- backlog refinement = ranking backlog in priority order
- break large stories into smaller ones
- top stories must fit in sprint
- top stories need enough detail to be sprint ready
- backlog refinement meeting
  - product owner must attend (writes stories, has vision)
  - scrum master helps refine
  - dev team optional (bring lead/architect for tech insight)
  - don’t bring full team—waste of time
- goal = ranked backlog
- only one item can be #1, #2, etc.
- focus on what’s most valuable to business
- stories near top must have enough info to start work
- don’t add detail in sprint planning, do it in refinement
- only focus on these kanban columns: new issues, icebox, product backlog
- new issues = always incoming from customers
- examples:
  - “remove counter”
  - “deploy to cloud”
- new issue triage
  - start with new issues column
  - goal = empty new issues column
  - inbox should be cleared every refinement meeting
- what to do with new issues:
  - product backlog = do soon
  - icebox = maybe later
  - reject = won’t do it
- example triage:
  - “remove counter” → icebox (not ready, no multi-counters yet)
  - “deploy to cloud” → product backlog (important soon, before reset feature)
- backlog refinement workflow
  - product owner ranks backlog
  - re-rank old items if needed
  - optional: add rough story point estimates
    - helps see size of backlog
    - refine estimates later in sprint planning
  - large items → split
  - vague items → clarify
- make stories sprint-ready
  - avoid detailing in sprint planning
  - questions from devs go in assumptions
- complete story template
  - as [role]
  - I need [function]
  - so that [value]

    - add acceptance criteria
    - add definition of done (Gherkin syntax)
        - Given [state], When [event], Then [outcome]
- example story refinement:
    - need counter service
    - assumptions:
        - need atomic increment function
        - need to get current value
    - acceptance criteria:
        - Given counter at 2, When get called → Then return 2
- product owner maintains groomed backlog
- always start refinement by triaging new issues

Backlog Refinement: Finishing Up

- add details to make stories sprint-ready
- identify technical debt stories
- rank backlog for sprint planning
- labels help visualize work
- use a few label colors
- GitHub default labels:
    - bugs: red → danger
    - enhancements: cyan
    - help wanted: green
    - questions, won't fix, invalid
- add custom label:
    - technical debt: yellow → caution
- example story: adds customer value → label as enhancement
- kanban board shows label color and story points
- next story: persist service counter
    - assumption: use Redis (MEM cache)
    - acceptance: restart keeps counter
    - label: enhancement
- next story: deploy service to cloud
    - lacks detail, refine story
    - assumption: IBM Cloud, Cloud Foundry or Kubernetes
    - acceptance: service available via URL
    - label: technical debt
    - customer expects availability, doesn't see added value
- technical debt = work without perceived customer value
    - not a new feature
    - examples:
        - code refactoring
        - maintaining environments
        - changing tech (e.g. databases)
        - library vulnerabilities
    - not bad → builds up naturally
    - pay it down consistently
    - include in every sprint plan
- next story: reset counter
    - assumption: POST to /counters/{name}/reset
    - acceptance: counter resets to zero
    - label: enhancement
- kanban board review:

    - 3 enhancement stories
    - 1 technical debt
    - labels give visual indicators
    - stories in icebox can be refined later
- backlog refinement tips:
    - refine every sprint
    - ideal time: Wednesday before sprint ends
    - sprint planning: Monday morning
    - refine weekly if many new requirements
    - always keep 2 sprints worth refined
    - saves sprint planning time
    - add story points if possible
    - avoid writing stories during sprint planning
- summary:
    - backlog refinement = order backlog + make stories sprint-ready
    - technical debt = no stakeholder value
    - goal = prepare for sprint planning

Sprint Planning

- Sprint planning: determine what stories will be in the next sprint
- Sprint backlog is the output of sprint planning
- Product owner, scrum master, and development team attend sprint planning
- No stakeholders
- Development team = cross-functional (devs, testers, ops, BAs)
- Each sprint needs a clear goal
- Product owner must articulate the sprint goal
- Goal helps team stay focused during sprint
- Goal defines purpose of stories selected
- In sprint planning, dev team moves stories from product backlog to sprint backlog
- Dev team assigns or confirms story points
- Dev team owns estimates since they commit to work
- Use planning poker to agree on story point estimates
- Stories must have enough info to start work
- Stop adding stories when reaching team velocity
- Velocity = story points completed in a sprint
- Velocity changes over time as team matures
- Cannot compare velocity between teams
- Different teams use different scale for estimates
- Fair comparison needs same measurement (not common)
- Create a milestone (or sprint in ZenHub)
- Milestone = title + goal + start/end dates
- Keep title short (e.g., "Sprint 1: Deployed to cloud")
- Description = sprint goal
- Recommended sprint duration = 2 weeks
- Start with groomed stories from product backlog
- Assign estimates if not already there
- Track total story points in sprint backlog
- Open each story, estimate, and assign to sprint milestone
- Example:
    - Large story → assign 8 points, add to sprint
    - Medium story → assign points, add to sprint
    - Technical debt → assign points, add to sprint
- Update sprint backlog until velocity limit reached

- Example velocity = 18 story points
- Stop planning when sprint backlog hits velocity
- Estimate mix: smalls, mediums, larges – not story count
- Key takeaways:
  - Product owner defines sprint goal
  - Dev team builds sprint plan
  - Sprint plan = stories from backlog up to velocity

Module 3: Daily Execution

Executing the Plan

Workflow for Daily Plan Execution

- sprint = one iteration through design, code, test, deploy
- two-week increments
- every sprint needs a goal
- during sprint execution:
  - take next highest item from sprint backlog you have skills for
  - don’t skip or pick favorites
  - order = business importance
  - assign story to yourself
  - move story to “In Progress”
  - everyone sees who’s working on what
  - on the Kanban board:
    - only focus on sprint backlog and right (In Progress, Review QA, Done)
    - open story → assign to self in GitHub
    - avatar shows up in In Progress column
- daily rules:
  - work on only one story at a time
    - can’t ship 50% of two things
    - only ship 100% of one thing
    - prevents split focus
    - exception: if blocked, can pick next story while waiting
- when finished with a story:
  - create pull request
  - move story to Review QA
    - ZenHub auto moves issue to Review QA if PR is linked
    - keeps things tidy and visible
  - when PR is merged:
    - drag story to Done column
    - restart: take next story, assign, work
- learned:
  - keep Kanban board updated
  - work on highest priority story you can do
  - don’t work on more than one story at a time or you risk not finishing either

The Daily Standup

- daily stand-up = daily scrum
  - same place and time every day
  - purpose: team sync on work, who's stuck
  - meeting is for the team, not management
  - stand-up = people stand to help keep meeting short
  - 15-minute timebox = critical

- not a project status meeting
  - not for the product owner to drill the team
  - not a manager's meeting
  - for development team: devs, testers, etc.
  - scrum master should always attend
    - to help unblock anyone
  - product owner = optional
    - can answer questions if asked
    - doesn’t get to ask questions or lead
    - not their status meeting
- 3 stand-up questions (each team member answers):
  - what did I get done yesterday?
  - what am I going to work on today?
  - are there any blockers?
- blockers:
  - scrum master handles blockers ASAP
  - if blocked:
    - take next story off backlog
    - start new story unless unblocked soon
- tabled topics = anything not in the 3 questions
  - also called "parking lot"
  - scrum master writes them down
  - discussed after meeting
  - not rude, preserves 15-minute limit
  - after meeting: anyone interested stays, others leave
- learned:
  - daily stand-up = 15 minutes
  - answer 3 specific questions
  - not a status meeting
  - table unrelated topics for after the meeting

Completing the Sprint

Using Burndown Charts

- burndown chart = tool to track sprint goal progress
  - works off milestones or sprints
    - milestone = demo, conference, etc.
    - most common milestone = two-week sprint
- shows story points completed vs story points remaining
  - over time within the sprint
  - helps forecast if sprint goal will be met
  - “burndown” = watching story points burn down over time
- chart structure:
  - vertical = number of story points in sprint
  - horizontal = number of days in sprint
  - vertical bars = weekends
    - assumes no weekend work
    - avoid team burnout
- optimal path plotted
  - ideal linear burndown based on even progress
  - not expected to be perfect, just a guideline
- actual progress line = blue line
  - each dot = a closed/done story
  - line tracks real story completion

    - if below optimal line = behind
    - if above = ahead
- example shown:
    - good progress early on
    - slows midway
    - team rallies, catches up
    - all stories done by sprint end
- not a management tool
    - for dev team to assess sprint status
    - quick, simple, easy to read
- learned:
    - burndown = story points completed vs remaining
    - useful for any milestone, not just sprints
    - forecasts if team can achieve sprint goal

The Sprint Review

- sprint review = demo time
    - demo valuable, done increment from sprint
    - live demos: show added features
    - product owner checks against acceptance criteria
        - approves = move from Done to Closed column
        - might suggest enhancements = write new story
- attendees:
    - product owner
    - scrum master
    - developers
    - stakeholders
    - customers
    - anyone interested in seeing the demo
- purpose of sprint review:
    - show development progress
    - get feedback from product owner and stakeholders
        - feedback becomes new backlog items
        - might inspire new ideas not considered before
    - benefits of iterative planning
        - builds things not possible with fixed planning
        - stakeholders respond better after seeing something
- rejected stories:
    - not moved to next sprint as-is
    - add label (e.g., unfinished, inaccurate), then close it
    - why close it?
        - preserve accurate velocity
        - team spent effort, should reflect in story points
    - write a new story with updated acceptance criteria for next sprint
- learned:
    - sprint review = demo of implemented features
    - everyone involved can attend
    - feedback is key to shaping product direction
    - backlog is updated from feedback

The Sprint Retrospective

- sprint retrospective = reflect on the sprint
    - critical for continuous improvement
    - measures process health and team health

  - team must speak freely and comfortably
- attendees:
  - scrum master
  - development team
  - product owner not invited
    - team may need to express concerns about product owner
    - avoid intimidation or hesitation to speak freely
    - only invite product owner if team is truly comfortable
- purpose:
  - honest discussion about the sprint
  - improve team and process for next sprint
- three retrospective questions:
  - what did we do well?
    - what should we keep doing?
  - what did not go well?
    - what should we stop doing?
  - what should we change?
    - what can we do differently next time?
- scrum master's role:
  - document suggested changes
  - ensure at least some changes are implemented
    - don't have to change everything
    - show that feedback leads to action
    - not just a gripe session
    - team should feel heard
- goal = improvement
  - reflect, discuss, take action
  - always room for improvement
  - retrospective must result in changes for next sprint

Measuring Success

Using Measurements Effectively

- metrics = critically important
  - can't improve what you can't measure
  - can't go by gut feel
  - high performing teams measure and react
  - take baselines, set goals, measure against them
- vanity metrics
  - example: 10,000 website hits
    - doesn't mean anything useful
    - no info on user behavior or what action to take next
  - no direction for improvement
  - just make you feel good
- actionable metrics
  - enable clear decisions
  - example: A/B testing
    - 50/50 traffic split
    - measure customer reactions
    - do more of what works, less of what doesn’t
- baseline and goal
  - measure current state
    - e.g., 6 teams, 10 hours to deploy
  - set improvement goal

    - e.g., reduce to 2 hours
  - measure across sprints
    - are we getting faster, better, cheaper?
    - fewer bugs in production, more in testing = better
- top 4 actionable metrics
  - mean lead time
    - time from idea to delivery
  - release frequency
    - how often can you release
    - only as fast as needed
  - change failure rate
    - how often releases break things
  - mean time to recovery
    - how fast can you recover from failure
    - better than trying to avoid failure completely
- example actionable goals
  - reduce time to market for new features
  - increase product availability
  - reduce time to deploy/release
  - increase % of defects caught in testing
    - saves money vs. fixing in production
  - provide fast feedback to team
    - improve customer-to-team feedback loops
- summary
  - high performing teams use metrics to improve
  - only use actionable metrics
  - take baseline before measuring change
  - top 4 metrics help improve overall team performance

Getting Ready for the next Sprint

- end of sprint activities
  - move all items in "done" column to "closed"
    - can be done during sprint review or now
  - close out current milestone
    - ZenHub sprints may auto-close
    - GitHub milestones must be closed manually
    - ensures credit for velocity in charts
  - create a new sprint milestone
    - can wait until next sprint planning
    - must be created before assigning stories
- handle unfinished work
  - untouched stories (never started)
    - move to top of product backlog
    - don't move directly to next sprint
      - priority may have changed
    - unassign from sprint milestone
  - unfinished stories (partially done)
    - don’t move into next sprint
      - affects velocity accuracy
    - determine if sizing was wrong
      - if yes, adjust story points
      - if no, split story
        - reduce current story to points completed
        - write new story for remaining points

            - put new story in product backlog
                - can move to next sprint if already created
            - assign new story the remaining points
        - adjust original story description
            - label as “unfinished” or “not completed”
            - helps identify as half-finished but part of sprint
        - close current story to get credit for completed work
- prepare for next sprint
    - close current sprint
    - handle all unfinished stories
        - close partial ones, write new ones
    - remove any stories from old sprint milestone
        - return to product backlog
    - create new sprint milestone
- summary
    - give developers credit for unfinished stories
    - split unfinished stories into new ones for next sprint
    - close each sprint milestone to reflect velocity
    - create new milestone for next sprint

Agile Anti-patterns and Health Check

- anti-patterns
    - no real product owner
        - unclear who owns vision
    - multiple product owners
        - conflicting directions
    - team too large
        - ideal is 5 ± 2 or 7 ± 2
        - communication breaks down
    - team not dedicated
        - members pulled into other projects
        - leads to failure
    - team too geographically dispersed
        - works best in one time zone
        - at least two people per geography
    - siloed teams
        - team should be cross-functional
        - avoid opening tickets with other teams
    - not self-managing
        - teams pick stories from backlog
        - assign stories to themselves
        - not told what to do
- scrum health check
    - everyone is accountable
        - product owner, scrum master, team
        - feel ownership
        - help when things go wrong
        - know and act on roles
    - small sprints
        - 1-2 week ideal
        - 2-4 weeks max
        - 1 week or 2 week preferred
    - ordered product backlog
        - disordered = disarray

        - stories need enough info
    - visible sprint backlog
        - use Kanban board
        - see remaining sprint work
    - sprint planning and forecast
        - sprint backlog and sprint are created
        - don't work from product backlog directly
        - organize around sprint goal
    - daily scrum leads to planning or replanning
        - adjust stories, help team members
    - done increment by end of sprint
        - should deliver something
        - goal is new functionality every 2 weeks
    - stakeholders offer feedback
        - critically important
        - deliver early and often
        - feedback changes plan/product
        - silent stakeholders = not healthy
    - product backlog updated after sprint review
        - add new stories or adjust existing
        - incorporate stakeholder feedback
    - team alignment
        - product owner, dev team, scrum master aligned
    - sprint retrospective
        - what went well / didn't go well
        - what to keep / stop doing
        - improve next sprint
- summary
    - anti-patterns:
        - no product owner
        - team too large
        - team not dedicated
        - team dispersed
        - siloed teams
        - not self-managing
    - health check = guidelines for healthy scrum team