Scrum: Details and Cookbook

Bruce McKenzie ([brucem@fullerton.edu](mailto:brucem@fullerton.edu) / [bjmckenz@gmail.com](mailto:bjmckenz@gmail.com))

[Terms](#_chchbx3014nw)

[Product Definition](#_9175qmsx513h)

[Press Release](#_ecjx3ci9pnxw)

[Elevator pitch](#_29xnzpjxwgud)

[Product Definition Should…](#_hvi1wpabb0fg)

[Product Definition Often…](#_lpi907619lvf)

[Roles / Personae (personas)](#_srvtv28ctrmu)

[Persona](#_3sjhdh3ykjk0)

[Feature](#_nrmco8c595s3)

[(User) stories](#_8eehjwvb06d9)

[Format/template](#_nf7cksmg2i7e)

[Quality Metric](#_dabovx1m0sil)

[Where do they live?](#_ha6cp54g2c4)

[Exercises to find User Stories](#_yqrsdjr494kh)

[Nonfunctional requirement](#_ls3een9urkb5)

[Often quality attributes](#_i8kifpm082x)

[Other examples](#_5dw3k19vh0zg)

[Where do they live?](#_6qgc1i2hplw8)

[Epic](#_n5y3f1n1v95g)

[Theme](#_wqoa4u1b4puf)

[Product Backlog](#_ptsmfhg1564w)

[How is it implemented?](#_ltmts86pcma8)

[What about changes for backlog items that are in the current sprint?](#_y4u9nyq1pv1)

[(Architectural) Spike](#_azqh8un19y6i)

[Task](#_gclfdhvhfil0)

[Sprint backlog (kanban)](#_bvwe06mhgm)

[Suggested Columns (Item state)](#_rez7wb9c2r1t)

[Suggested Tags (item attributes)](#_by0bl3rrjyse)

[What about Bugs?](#_cdr4ym4hg2gn)

[Should Bugs have a Priority?](#_tmwi7zhdcfoj)

[Should Bugs have a Severity?](#_lyioy6cqysbg)

[How is it implemented?](#_eb7kxh50qdzt)

[Technical debt](#_l04j95wl5b4i)

[Refactoring](#_3qq4ncbikl5m)

[Why do we refactor?](#_5y2rc5toe2xa)

[Rituals](#_vqz8pa6ov10q)

[Planning Poker](#_5ft2xbb3sxwd)

[Product backlog refinement meeting](#_kc811lnrv8db)

[Sprint Planning Meeting](#_yjpklnl2cz3p)

[Daily Scrum](#_qb7h7cvd7i1a)

[Sprint Demo](#_3lsyksenj13l)

[Sprint review (retrospective)](#_sryjn1fui6uy)

[Roles](#_rds1rs9yumgh)

[Product owner](#_yl0qba9lgg6g)

[Scrum Master](#_7m3ppyj1mmwd)

[QA](#_qqav1qqelnid)

[Tech chaser](#_jh6sood2f6ng)

[Tech lead](#_o5jqd29yxaw6)

[Process (Steps)](#_y0kydggpju9v)

[Create Product Definition](#_u2z5agr0kep3)

[Choose key development practices (and team norms)](#_bl8i6b2oqksm)

[Perform an Initial Iteration](#_ujujmekq89wy)

[Sprint: Iterate (repeat until project is complete)](#_l2vkbswu4sfo)

[Finalize Project](#_36cuvnybuhnl)

# Terms

## Product Definition

A summary, about a page long that gives a good sense as to the product. Has at least the following components:

* Title
* Press Release or Elevator Pitch

### Press Release

<https://www.linkedin.com/pulse/working-backwards-press-release-template-example-ian-mcallister/>

### Elevator pitch

* how you would sell (get someone excited about it) in 8-20 seconds?
* low tech / no prerequisite knowledge
  + would your aunt/uncle understand it?
* formal template
  + For <user>, Who <statement of need>
  + The <our product> that <key reason/benefit>
  + Unlike <main competitor>, our product <how it's different>.
  + (template from G. Moore, Crossing the Chasm)

### Product Definition Should…

* State why the user wants this
* More about benefits than features
* May indicate top-level architecture
* Mobile app vs. web site vs. …
* Indicate main tradeoffs
* priorities among quality, schedule, features
  + assuming fixed team size
  + if team size can vary, that's another tradeoff

### Product Definition Often…

* includes team and stakeholder lists
* keeps list of satisfied non-functional requirements
  + as reminders after implementation

## Roles / Personae (personas)

* a type of user
* has
  + goals
  + characteristics
  + Background
* for role, a person may have many roles

### Persona

* Fleshed out
* Name, personal problems, etc.
* “Write their story”

## Feature

* provides a benefit to the user
  + worth $
  + can be advertised to user
* Usually "unlike our competition..." or "can now... <something that could not be done before>”
  + Frequent exceptions
    - Usability: often a list of changes, but with a theme/goal
    - Table stakes (“everyone does it”)
* may involve multiple user roles
* may be decomposable into smaller features
* almost synonymous with "functional requirement"
  + FRs are usually lower level
  + more common in non-agile projects

## (User) stories

* provides a benefit to a user
  + aids user in achieving goal
* part of a [feature](#_nrmco8c595s3)
  + could be shipped by itself
* "A placeholder for a conversation"
  + the story is rarely enough detail
  + conversations with [PO](#_yl0qba9lgg6g) flesh it out
  + notes, diagrams, etc are added as it's groomed

### Format/template

* Formal style
  + "As a <user role>, I want to <do something> so that I can <achieve some goal>"
* Informal style
  + <verb> a <noun> (example)
  + assumes we know the/which user role and what their goal is (default: “use our awesomeness”)
  + verb is pretty important because noun phrases are intrinsically static

### Quality Metric

* - Stories are good enough when they are INVEST
  + independent
    - ideally does not depend on other stories
  + negotiable
    - essence, not details
    - room to implement it different ways based on conversations
  + valuable
  + estimateable
  + small
    - well less than a sprint
  + Testable

### Where do they live?

* [PO](#_yl0qba9lgg6g) prioritizes them in the product [backlog](#_ptsmfhg1564w)
  + this is because of Pareto Principle (80% of benefit from 20% of the stories)
* may be broken into multiple tasks when taken into sprint
  + story is only complete when all tasks are complete
* story is done or not done; no partially done
  + if team doesn't complete, it stays on product backlog

### Exercises to find User Stories

* write out or diagram steps of a process flow
  + "search for product" -> look at product -> buy product
* do a low-fidelity user prototype
  + pencil, paper
  + buttons and elements lead to stories
  + stories are not "I want to click this button" or "...fill in this field"
  + STRONGLY recommend throwing this away afterwards because it might anchor your thinking
* scenarios
  + "so Bob want to buy a car but he has no money"

## Nonfunctional requirement

* "things the user doesn't see"
* "things you don't write code for"
  + this is very approximate and rough, not a hard and fast rule
* often known as "constraints"

### Often quality attributes

* performance
* scalability
* security
* usability
* releasability
* (etc)

### Other examples

* may include business things like
  + "use this technology"
  + "interface with this system"
    - semi-non-functional
  + "have a coding standard"

### Where do they live?

* 99.9% of the time, write them as user stories or constraints on existing stories
  + the [PO](#_yl0qba9lgg6g) and Developers may be roles here
* Track them separately when done so we don't forget them
  + not as "to do", but as part of the project overview
* Super important to test since they may be violated without breaking functionality

## Epic

* a large [story](#_8eehjwvb06d9)
  + too large to fit in one iteration
* needs to be broken into smaller user stories
* Epics != [Features](#_nrmco8c595s3)
  + Because stories are not features
* Why do they exist?
  + Because sometimes a simple-sounding user story is just plain big
  + We don’t want to slow down the listing of stories by requiring “design”

## Theme

* a group of [stories](#_8eehjwvb06d9) or [features](#_nrmco8c595s3) that share a characteristic or attribute
  + usually a non-technical grouping
* used by [product owner](#_yl0qba9lgg6g) to think about releases and phases
* Why do they exist?
  + Because we like grouping things
* Themes != Features

## Product Backlog

* a prioritized list of [features](#_nrmco8c595s3) and [stories](#_8eehjwvb06d9)
  + Prioritized
    - high/medium/low or must/should/could at minimum
  + ideally more granular
    - 1-5? 1-10?
  + A single list combining both
  + When a feature is broken into stories, the feature is removed from the list (since it is embodied in the stories)
    - So there is either a Feature in the list, or its Stories
    - Not both!
* Starts (initialized by) from the [Product Definition](#_9175qmsx513h)
* As team works with PO to create stories, the Feature entries are removed
  + When all stories are identified
  + It’s OK for PO to create stories themselves
  + PO should keep initial list of features somewhere to manage product at higher level
* [PO](#_yl0qba9lgg6g) owns prioritization
* PO can (and should) adjust priorities as much as needed to reflect reality
* It is usually not valuable to strictly order stories/features
  + Because we want flexibility to select “close” stories during sprint planning based on tech or team needs
  + It implies a false precision
* “Priority” is NOT Order or Index
  + Priority is a way to create “bins” or groups
  + There should be (almost certainly) more than 3-4
  + Probably less than 10 -- and that’s for the biggest projects
* May also contain entries for big tech debt/refactoring
  + Those \*should\* be written as "how they provide business value"
  + may require PO to trust the team
  + PO can (and should) ask "why can't you do this in pieces over sprints"
  + Be very very suspicious of big refactorings
    - they often promise more than they deliver
    - They can become never-completed
* some people put bugs on it too
* Don’t include tasks
  + They’re too low-level
    - they often require or imply technical knowledge
  + They don’t correlate directly to business value
  + They usually have dependencies
  + These mean that the PO is not going to be effective at prioritizing them
* Items are called “Product Backlog Items,” often abbreviated “PBI”
* top of backlog has more detail
  + Detail added during backlog refinement meeting
  + When items have been reviewed and elaborated by dev team, they are called “groomed”
* anyone can add to the backlog
* probably has "---release here---" notes/lines
  + between stories
  + Up to the PO if they like

### How is it implemented?

* Can be spreadsheet, kanban, github project
* Recommended: spreadsheet
  + easiest to add columns, sort, view all, etc
* github project is easier EXCEPT
  + harder to get overall view
  + harder to make broad changes
* There are plenty of commercial tools
  + JIRA, Wrike. Rally
  + Trello is excellent

### What about changes for backlog items that are in the current sprint?

* entries in current sprint are inviolate
  + can't delete or change them
  + except in concert with dev team
* implicitly at top of list until end of sprint
* Effectively, product backlog is authoritative only for future items

## (Architectural) Spike

* Timeboxed investigation
  + usually of a [product backlog](#_ptsmfhg1564w) item or architectural option
  + no more than 1-2 days, 1-2 people
* Used to reduce risk
  + we don't know how to do something so we can't estimate it
  + we don't know if the approach is going to work
* If connected to a backlog item, spike has to be completed before the risky item can be planned (put in sprint)
* Throwaway code
  + Generally no non-functional requirements (standards, etc)
* Not strictly sprint tasks

## Task

* Tightly focused work item
* about a day of work for one person
  + Usually not “sized”
* may be (not exhaustive)
  + build some code
  + fix a bug
  + refactor some tech debt
  + (less frequently) do a spike
* need not be user-visible or have (direct) user benefit by itself
* must be required as part of a one or more stories
* task is never placed on [product backlog](#_ptsmfhg1564w), the story is
* may (or may not) be individually shippable
  + But it’s very rare to ship a task without the rest of the story
  + Because how do you know it’s sufficient for the rest of the story?
* determined by the team at the time of sprint planning
  + not enumerated by product owner
* They do not transition to new sprints if not completed
  + The undone portion is reflected in an incomplete story, which does go back on the product backlog

## Sprint backlog (kanban)

* A list of sprint tasks (and bugs)
* Usually implemented as a kanban where columns indicate progress towards completion
* Items usually have tags to indicate characteristics

### Suggested Columns (Item state)

* To Do
  + everything starts here
* In Progress
* Ready for Test
  + (only if there are QA besides devs)
* Done (ready for demo)
* On Hold
  + (a place to put things that need resolution within a day or so)
  + discovered stories
  + blocked stories
  + [PO](#_yl0qba9lgg6g) and [SM](#_7m3ppyj1mmwd) watch this carefully
* Delivered
  + (a place to put things after demo)

### Suggested Tags (item attributes)

* duplicate
  + it's a duplicate of another item
* backlogitem
  + is this item from the [product backlog](#_ptsmfhg1564w)?
* Examples of “Not from the backlog”
  + tasks
  + bugs from this sprint
  + techdebt
    - this item notes technical debt
    - ideally taken care of this sprint

### What about Bugs?

* Ideally few, because you test your stuff
* Tags unique to bugs
  + CNR/Invalid
    - could not reproduce
    - Not a valid case/use
  + wontfix
    - valid bug, we're just not gonna fix it EVER
  + usability
    - An example of a bug type
    - Many people add others
    - one of the most useful
* other candidates for bug types
  + database
  + performance
  + crash
  + security

### Should Bugs have a Priority?

* If there are more bugs than can be fixed in a sprint
* In Scrum they would be prioritized into the [product backlog](#_ptsmfhg1564w)
* In non-Scrum they are likely their own list/backlog
  + Large topic, beyond scope here
* Priority is a Product and Operational judgement
  + Bugs may be from operational systems
  + Bug Priority can change!
  + No correlation to Severity
  + Example: “Bugs that make us look stupid”
* Priority Example Taxonomy
  + Immediate
    - Drop everything, work immediately
  + Urgent / High
    - Finish your current task, then work
  + Normal
    - In the mix of current task list
  + Low
    - Only if you’re sitting around

### Should Bugs have a Severity?

* Severity is a measure of the impact of the bug
  + It is objective
  + It only changes if more info is discovered
* Again, most useful for long-running bugs
  + In Scrum, most useful for metrics
  + Since ideally no bugs leave the sprint!
* How is it implemented?
  + An attribute of the bug, so either a tag or field
* Candidate severities
  + Bricks system
    - System cannot even be restarted after this happens
    - Breaks system for multiple users
  + Data loss
    - User/organization data is destroyed or disclosed
    - Not just “the order I was creating”
    - More like “my credit card”
  + Crash
    - System must be restarted
  + Major Function
    - Core purpose of system cannot be achieved because of this
    - “Cannot place order”
  + Minor Function
    - Some secondary function or options cannot be accessed
    - “Shipping method cannot be changed”
    - “Cannot view my past orders”
  + Cosmetic
    - Functionality not impaired
    - “Image doesn’t load”
    - “Text overlaps”
* These are often given numbers from 0-N
  + 0 = most severe

### How is it implemented?

* Many commercial and free tools
* Best balance
  + Github project
  + Automation can be created. Quite powerful
  + (“Convert to Issue” each item so you can assign and add tags/labels)
* Solid general tool
  + Trello
* Commercial
  + JIRA
    - Painful unless customized carefully
* Best fallback
  + Google Sheets

## Technical debt

* Code (or architecture) we would write differently based on
  + new skills in team
  + new features (features added now or since code)
  + cost to change things
  + (failed) experiments
* Commonly (industry standard) "crappy code"
  + My view: crappy code should not be checked in
  + was person lazy? rushed? Forgetful?
  + Ideally "author" fixes it ASAP
* Make a list of known tech debt!
  + although you should fix it when detected
  + you don't always have time
  + Lets you track and learn from it

## Refactoring

* changing code for some purpose without changing functionality
* best done in small bites
* ideally fits into sprint tasks
* tests must pass exactly like they did before

### Why do we refactor?

* reduce cost of change
  + Make it easier to read, understand, test
  + better fit for functionality/architecture
* remove unnecessary/dead code
* improve code quality
* removing technical debt
* Examples
  + simple - renaming variables
  + deep - rewriting whole modules

# Rituals

## Planning Poker

* Goal: Resolve differences in understanding
  + Scope, complexity
* Most commonly/importantly used
  + Grooming User Stories during Product Backlog Refinement meetings
* Who:
  + All team members (except [PO](#_yl0qba9lgg6g)) regardless of role
* How:
  + Team agrees on what a “1” (basic) item is
    - Say, adding a field to a form
    - All aspects of the task!
      * Testing
      * Delivery
  + Team reads a epic, story, or task definition
  + Each member secretly assesses size relative to basic item
    - Candidate numbers are: 1,2,3,5,8,13+
  + Team simultaneously reveals their assessment
  + If the numbers are not identical…
    - Talk about why people made choices
    - What do they know or suspect? What are they worried about
  + Repeat the assessment
    - Until the numbers are identical
* Planning poker works because humans are better at estimating relative things than absolute things
  + It's not voting
  + It doesn’t stop til the numbers are identical
* Norms
  + It's OK to trust others
  + “Oh, Sam thinks it’s an 8 and she knows the most about it”
  + It's OK to ask people to help you understand the basis for their number
  + It's not OK to pressure others to change
* Simultaneous reveal reduces implicit and explicit pressure
* Save the numbers for Sprint Planning
  + They’re not strictly required, but they’re helpful to remind people
  + If your team is tracking velocity, save the numbers after that

## Product backlog refinement meeting

* The purpose of this meeting is to ensure that the top of the [product backlog](#_ptsmfhg1564w) ready for sprint planning
  + for each of the top ranked stories (or top 25% of product backlog)
* break feature into stories
* Play Planning Poker to ensure estimate of story/item is the same
  + If item is “13+”, it’s an epic
* break epics into smaller stories
  + until stories are "yeah, the team could accomplish this in a sprint"
  + I.e. Planning Poker size of 8 or less
* (these may lead to stories getting added to the backlog)
* make sure that every person understands all words of the story
* if no one has good sense of size or risk, add a spike for this
  + complete the spike before sprint planning or the story can't be done
* jot down notes about definition of done and acceptance testing
  + if there is a QA member, create testing/acceptance notes
* After a story is addressed here, it’s called “groomed”
* Meeting is occasionally called “Product Backlog Grooming”

## Sprint Planning Meeting

* input is a [product backlog](#_ptsmfhg1564w) of groomed stories
  + Do not do planning if stories aren’t groomed!
* output is a [sprint backlog](#_bvwe06mhgm) of tasks
  + The backlog should be full enough that there is no similarly prioritized story that would fit in the sprint
* working high (pri/rank) to low from the product backlog
  + Always take the highest pri/rank items
  + Sometimes OK to fill sprint with a few lower-pri items
    - This is up to the [PO](#_yl0qba9lgg6g)
* team takes a story
  + "can we safely add it to sprint?"
    - if you're tracking velocity, this is based on numbers
  + if so, do
    - If not, you’re either full or potentially taking a lower-priority task
  + product owner marks as "in current sprint"
  + break into tasks
    - each less than a day
    - add them to sprint backlog
      * If the story has no tasks, add the STORY to the backlog
      * If the story has tasks, add the TASKS to the backlog
    - (small stories might be just one task)
    - tasks might have some dependencies among them
      * But no dependencies outside of the sprint
    - if QA, add the QA task for story
      * (probably not a QA task for most \*tasks\*)
    - It’s not necessary to estimate tasks’ size (e.g. story points) because the only use of size is “is the sprint full?”
  + repeat until sprint is full or there are no more broken-down tasks
* Look over sprint backlog and see if there is a recommended order of doing the work
  + There may not be if the stories are similar in priority and have no (task) dependencies
  + Move higher-priority tasks/stories and dependencies to top
  + This lets people take work from the top during the sprint

## Daily Scrum

* The purpose of this meeting is to keep communications open among team members
  + To ensure the [sprint backlog](#_bvwe06mhgm) is up to date
  + To ensure that blocking issues are surfaced as soon as possible
* How frequently in this remote/virtual world?
  + synchronous audio/video at least 2/week
  + OK to do text-based other days
    - still synchronous (due by HH:mm)
* for each person, make all of the following statements
  + "I got X done since our last scrum"
  + "I plan to do Y before our next scrum"
    - Alternate: "I \*commit\* to Y..."
  + "I can't do Z because of BLAH"
    - Scrummaster - Call to Action!
  + "I found out that we might need to account for J"
    - maybe a task
      * will it blow the sprint?
    - maybe an item for [product backlog](#_ptsmfhg1564w) that relates to THIS sprint
      * [PO](#_yl0qba9lgg6g): Call to Action!
      * (generic new backlog items just get added there; no need to call out this way)
    - only bring up here if it risks the sprint or involves multiple people
  + Encourage others; this is closer to a celebration than a witch-hunt

## Sprint Demo

* The purpose of this meeting is to celebrate the releasable features and mark their completion.
  + The completion is marked by the [PO](#_yl0qba9lgg6g) declaring them acceptable
* It often includes acceptance testing by the customer (PO), though detailed tests may have been done during the sprint
* For each completed story, show built code from github
  + Totally releasable code/version
  + not from incomplete tasks
  + if from a local computer, NOT built from unsaved changes
  + Product owner says
    - "done"
      * removes item from [product backlog](#_ptsmfhg1564w)
    - "not quite right"
      * update product backlog item to show needed work
      * This is often worth reflecting on in the Retrospective: why did this mismatch occur?
      * This feature is not released
        + Ideally the code is not merged in
* all other items go back to the product backlog
  + No demos of undone/uncommitted features!
  + Undone
  + Buggy
  + Not what was expected (fails to meet PO's needs)

## Sprint review (retrospective)

* The purpose of this meeting is to finalize the existing sprint and learn any lessons
* **This should be a live/synchronous meeting (Zoom) unless absolutely impossible**
  + Use a shared doc to collect notes
* Clear out [sprint backlog](#_bvwe06mhgm)
  + remove unneeded tasks
  + promote any (ideally none) bugs
  + return undone stories to [product backlog](#_ptsmfhg1564w)
* Retrospective Prime Directive
  + We assume that everyone did the best they could with what they knew and had on their plate
* Have a brainstorming session (retrospective)
  + Read the Prime Directive
    - **We assume that everyone did the best they could with what they knew and had on their plate**
  + make 4L list, either on board or shared doc (when virtual)
    - Liked
    - Learned
    - Lacked
    - Longed For
  + Probably ask...
    - why did all tasks not get done?
    - how did we do on our rituals and items?
    - how do we feel?
    - Did we hit our goals (often quality)?
    - Did we groom an appropriate amount of the product backlog?
    - Were our groomings effective?
  + [Scrum Master](#_7m3ppyj1mmwd): Call to Action!
    - ensure that each person is engaged and contributes
  + Brainstorming portion should last from 15-45 minutes
    - Very rough -- it should go til it gets worked out
  + After that
    - Choose 1-3 things to change/improve next sprint
    - **These must be concrete and specific**
      * “We’ll try harder” is neither
      * “When someone notices the build is broken, we’ll have a Zoom that day” is better
    - **More important to call out process things than technical things**
      * “We should change to MySQL” is OK
      * “When we think we need an arch spike, we’ll brainstorm questions to research in a Zoom call before it is started” is more likely to help you beyond this project or sprint.

# Roles

## Product owner

* prioritizes [product backlog](#_ptsmfhg1564w)
  + though anyone can add to backlog
  + only PO is authoritative
* Adds features to product backlog as product evolves
* makes decisions about details when asked
* accepts/rejects work at sprint demo
  + updates product backlog with results
* "keeper of the vision"
* Key participant in Product Backlog Grooming meeting
* (in this class, probably a part-time role)

## Scrum Master

* in distracted/remote world, lets people know their absence is noticed
* makes sure rituals occur
* encourages full participation in rituals
* #1 problem solver / chaser when something blocked
* NOT a “lead/manager”
* Facilitator in all meetings, but most of all in Daily Scrum and Retrospective
  + “Facilitator” does not mean “Leader” or “Passive”
* (in this class, probably a part-time role)

## QA

* sometimes!
  + if there is a QA/tester on the team
  + if there are many interactions among modules that would be hard to find with just unit tests
  + if the dev team is not that disciplined
  + OK to have a person be the "doomsayer" -- someone who thinks of odd cases for devs to explore
  + really useful to have this person active when stories are fleshed out
* This is not a place to stash “not very technical” people!

## Tech chaser

* Very informal role
* researches tech questions
* Likes digging into things
* finds training
  + shares with team
* prototypes things
* may or may not be tech lead
* (in this class, probably a part-time role)

## Tech lead

* person best suited to make large technical decisions
* usually most experienced in tech
* often most experienced in domain
* rarely formal hierarchical role
  + usually "first among equals"
* usually has side goal of bringing rest of team up in skill (mentoring)
* (in this class, probably a part-time role)

# Process (Steps)

## Create Product Definition

* all contribute to this
* (in this class, follow the formats carefully)

## Choose key development practices (and team norms)

* Engagement
  + working hours/days
    - when are people online
  + amount of time that will be available
    - how much are people committing?
  + (and other norms)
* Iteration length
* daily scrum
  + frequency
  + method
* Tools for...
  + [product backlog](#_ptsmfhg1564w)
  + version control
  + sprint backlog
  + user stories
    - where is extra detail stored?
      * diagrams, tests, notes
  + General documentation, if any

## Perform an Initial Iteration

* *This is all the stuff that takes us from “Product Definition” to “Ready to Code”*
* aka "Iteration 0"
  + don't worry about estimation, planning poker, velocity
  + only iteration where demos might be just docs
  + timebox it to one (standard) iteration length
* Goal
  + Required: ready to start working at next iteration
  + Stretch: "hello, world" of product or tech stack
* (have Daily Scrums in this)
  + super important because tasks are indeterminate and risky
* Key early tasks (ordered)
  + set up sprint backlog structure/tool
    - use it for this iteration to track the tasks here
    - Only time backlog tasks may not be connected to [product backlog](#_ptsmfhg1564w)
    - only time tasks are not always pre-divided to one-day durations
  + choose roles
  + create [product backlog](#_ptsmfhg1564w)
    - brainstorming from product definition
    - initially, probably no stories, just features
    - [PO](#_yl0qba9lgg6g) takes lead here
      * prioritize it
  + Have a Product Backlog Refinement Meeting
* General tasks (loosely ordered)
  + think about risks
    - decide if each is worth worrying about.
  + think about architecture
    - tech stack
    - major subsystems/modules
  + get training lined up
    - based on architecture
  + set up tech pipeline
    - version control
      * permissions
      * branches
      * README, notes
    - build system
    - CI system
  + (do any known spikes)

## Sprint: Iterate (repeat until project is complete)

* begin sprint
  + Have a [Sprint Planning Meeting](#_yjpklnl2cz3p)
* Repeat ~each day
  + have a [Daily Scrum](#_qb7h7cvd7i1a)
  + each person takes a task
    - ideally work on only one task til it's done
    - Take most impactful task that you have capabilities for
      * Task that is blocking another task or story
      * Task for a story in progress
      * (otherwise) task for a story that isn’t being worked yet
    - move task to "in progress"
    - keep sprint backlog up to date
    - When completed
      * done = ready to demo, checked in
      * If QA present, move to "Ready for Test"
      * If no QA present, move to "Done"
      * if QA, notify QA when all tasks for story are done
      * even tasks have to be testable!
  + if sprint backlog is empty, notice this
    - no intrinsic requirement to do more work
    - something to reflect on at Retrospective
* (several days before end of sprint, have a [Product Backlog Refinement](#_kc811lnrv8db) meeting)
  + best practice - do NOT combine it with end of sprint activities
  + do it in the middle of sprint
  + gives people time to reflect, and makes it less hurried
* End of Sprint
  + have a [Sprint Demo](#_3lsyksenj13l)
  + Release the Code (if it's appropriate)
  + Have a [Sprint Retrospective](#_sryjn1fui6uy)
  + Empty the Sprint Backlog (if anything left after Sprint Demo)

## Finalize Project

* Have a project-level Retrospective
* What happened over the course of the project?
* What did we learn that would affect future projects?
* Celebrate!