

---

# Scheduling

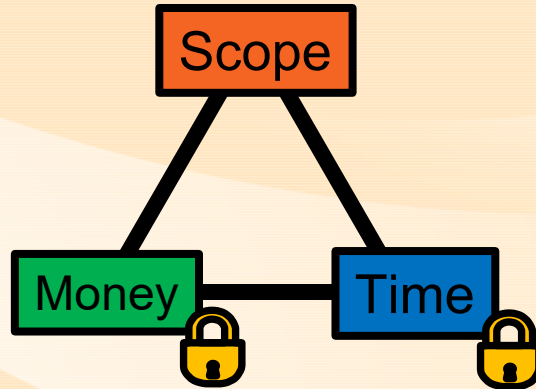
---

# Triple constraint

---

The three things that limit production

- All three can not be fixed (Pick 2)



# What are the steps to scheduling?

---



First: Gauge the resources available

- Staffing
- Licensing
- Physical resources

Money



Second: Recognize how much time is available

- Budgeting
- Projected launch date

Time

Third: Identify what takes priority

- Break down the tasks to be completed
- Select tasks until that time is filled
- Organize tasks in a timeline

Scope

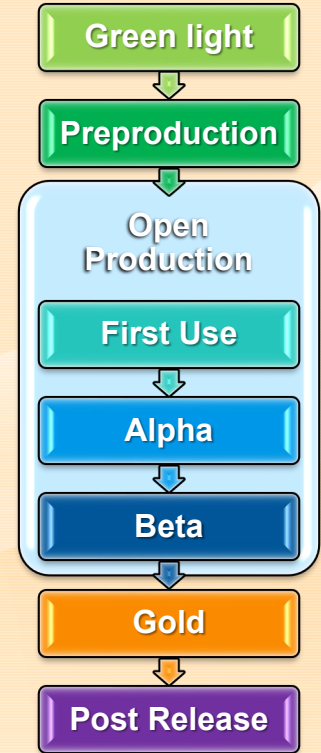
---

# Phases of Production

---

# Phases of Production

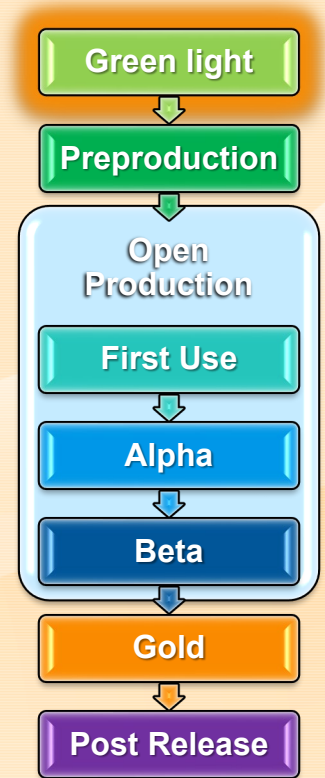
- Greenlight
- Pre-Production
- Open Production
  - First Use/Playable
  - Alpha
  - Beta
  - Gold
- Post Release



# Green light

## Concept and funding

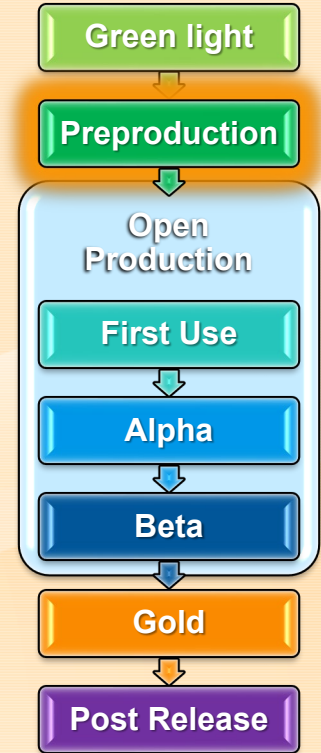
- Core documentation is written
  - Project description
  - Business case
  - History of like projects
- Conceptual artwork is created
- Risk/complexity assessment
  - Tech, Design, Assets, Organizational...
  - Paper and electronic prototypes are created, tested, and prove the idea works
- Funding has been structured



# Phases: Pre-Production

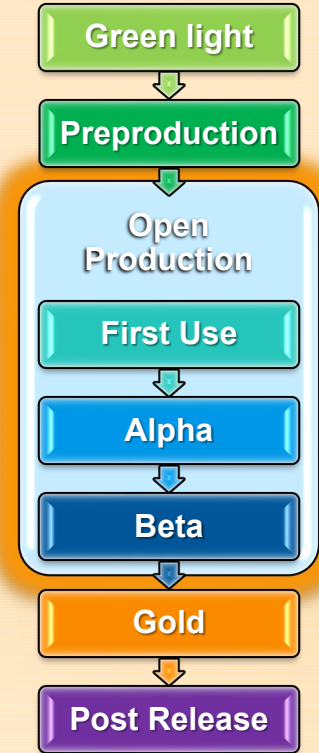
- Project management plan
  - Formalize process
- Project scope baseline
  - Design understood and documented
  - Task breakdown/Product backlog written
  - Engine/Tech Research completed
- Budgeting baseline
  - Licenses
  - Physical needs
  - Evaluate Human Resources
- Schedule baseline
  - Milestone dates/Gantt charts

Important note: We should be at this point at the end of AHI



# Phases: Open Production

- Planning is done let's get building
- Obvious aspects
  - Complete tasks
  - Verify completeness
- Change requests
  - How to you handle changes to the game designed in preproduction

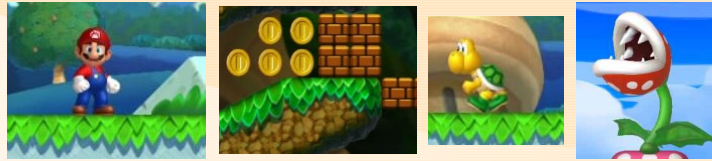




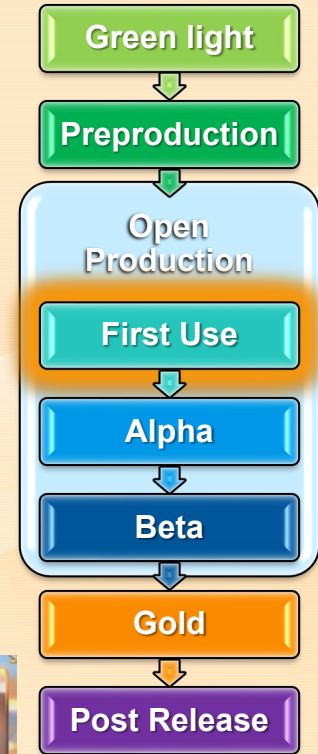
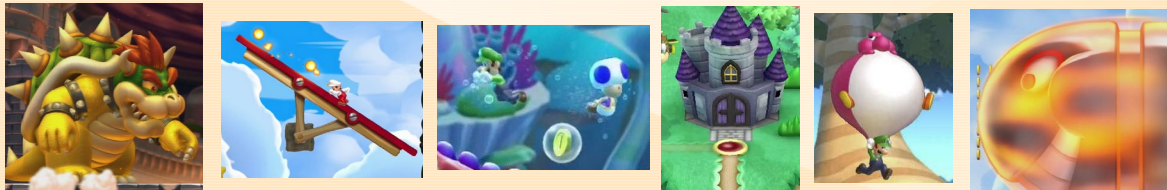
# Milestone: First Use/Playable

- A completed single level that displays most Global aspects of product in a Local environment.

- Global: Things necessary for every portion of the product
  - Main player actions
  - Environment
  - Main obstacles



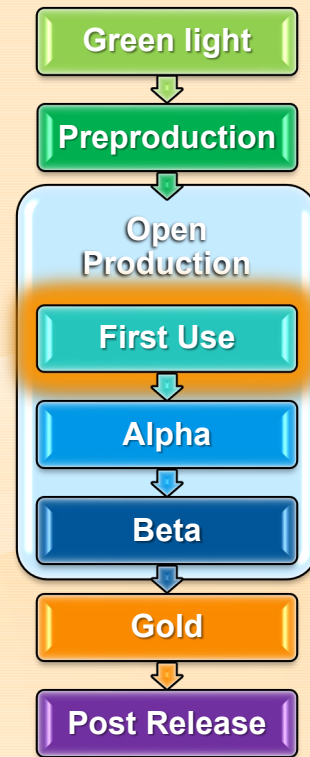
- Local: Things only necessary for specific portions
  - Everything else



# Milestone: First Use/Playable

- Vertical Slice
  - Useable product
  - Intended gameplay achieved
  - Fun factor realized
- Play/Use testing can start
  - Product must be able to sell itself
- Often when projects get canceled

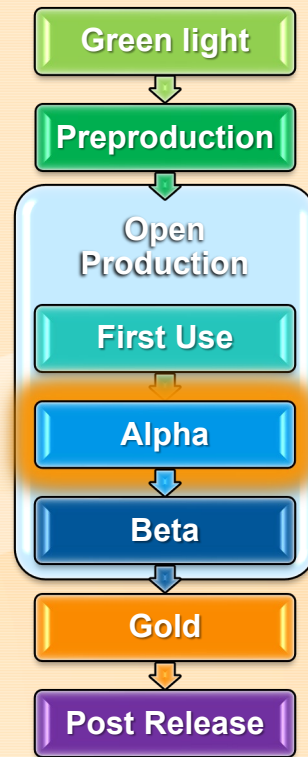
Important note: We should be at this point at the end of PP2



# Milestone: Alpha

Completed the construction of all features

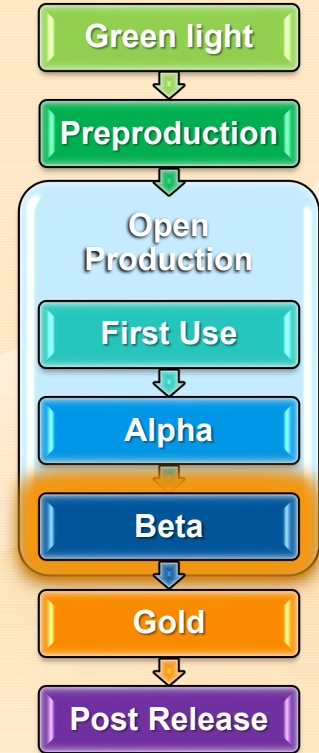
- Example of every features exists in the game
- Unnecessary features dropped
- Active development on new functionality stops



# Milestone: Beta

Finalizing content for the product

- All placeholders and temporary content replaced with final quality versions
  - Art
  - Sound and Music
  - Design/Layouts
- Balancing and play testing completed
- Removed all debugging tools

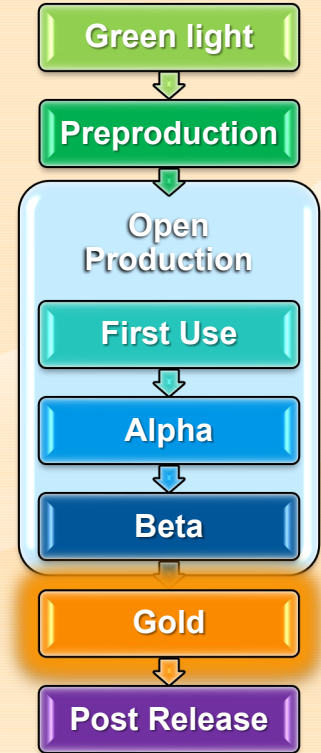


# Milestone: Gold!

---

Released

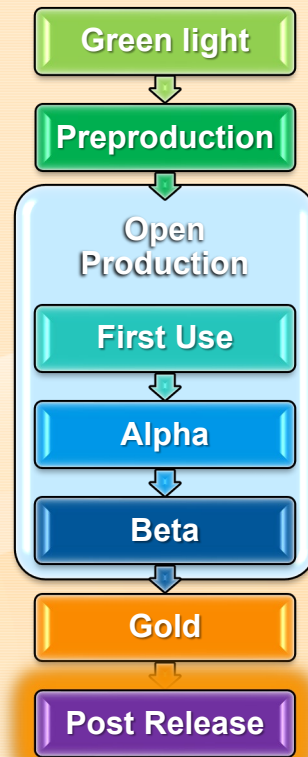
- QA completed
- Final build created
- Passed all Certification requirements
- Manufacturing and shipping completed



# Phases: Post Release

In the user's hands

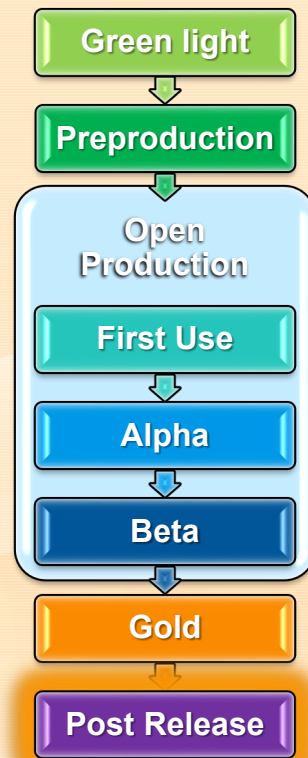
- Postmortems completed
  - What went well
  - What went poorly
  - Lessons learned
- Archiving reusable assets for future projects



# Phases: Post Release

In the user's hands

- Bug fixes continue
  - User will always find bug you missed with pure man hours  
Team of 10 testers testing the game for 2 months gets 3200 man hours of testing (10 people for 8 weeks, 40 hours a week)  
=  
Game purchased by 3200 people in 1 hour



---

# Software Development Methodologies

---



# What methodologies try to fix

---

## Dealing with uncertainty

- Shifting goals
  - From testing/use
  - Client
- Scheduling issues
  - Department down time
  - Milestone/ship dates
- “Technical debt”
  - A debt of time created by implementing something for the short term, without thought or concern with long term ramifications, that will require refactoring and revisions in the future.

# Different methods

---

There are many different development methodologies

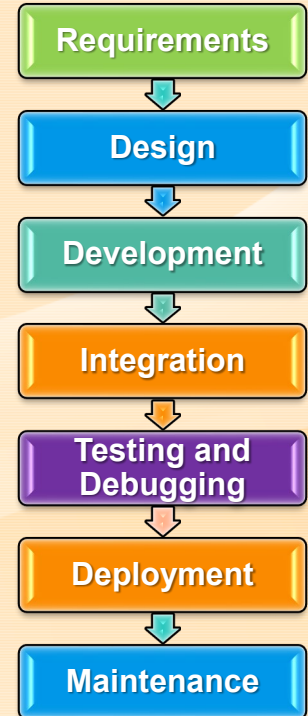
- Most commonly used in game development and Sim industry
  - Waterfall
  - RAD
  - Agile/Iterative

# Waterfall (a.k.a. Traditional)

---

A linear stage-based model

- Requirements are well documented, clear, and fixed
  - Full production and milestone schedule assembled during initial planning
- Move from one phase to the next only after it is reviewed and verified

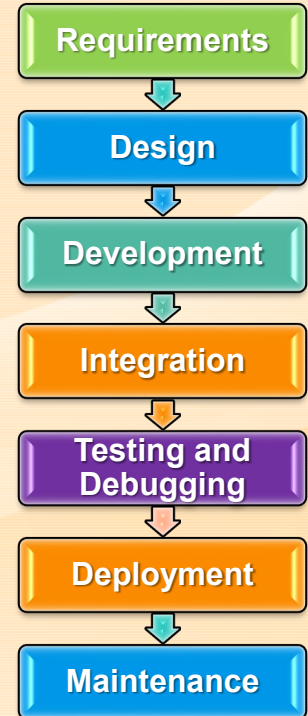


# Waterfall (a.k.a. Traditional)

---

## Pros

- Simple to understand
- Each stage has a definitive focus that improves the quality of the output
- Little to no methodology overhead

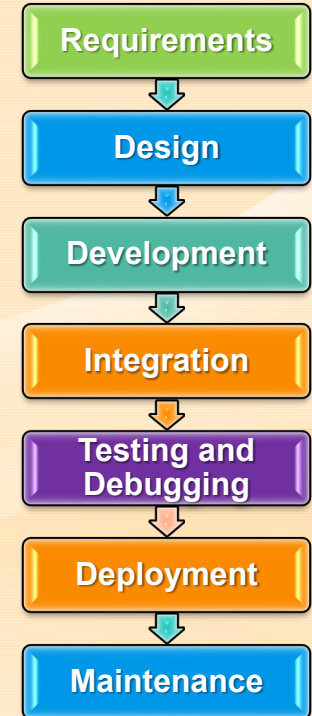


# Waterfall (a.k.a. Traditional)

---

## Cons

- All requirement analysis and design must be done up front
- Hard to estimate times accurately
- Not built to handle changes or revisions during development
- No working build until late in the process
- Can cause disciplines to become idle

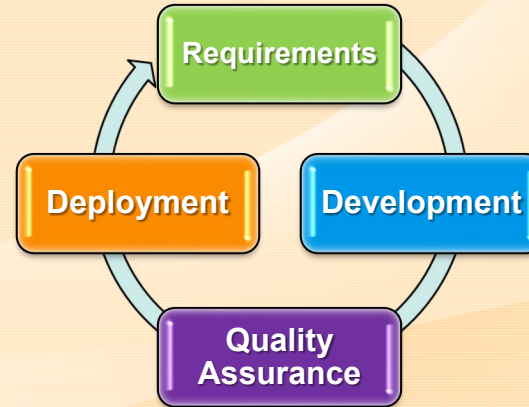


# RAD (Rapid application development)

---

A cycling development pattern  
getting functioning software out as  
fast as possible

- Product is delivered in an incremental manner
  - Single or few features completed on each cycle
- Best for already established products or prototypes



# RAD (Rapid application development)

---

## Pros

- Very fast turn around on feature requests
- Can accommodate changing designs and priorities
  - End user involvement
- Short turnaround on investment
- Consistent integration schedule lowers the risk of large scale integration issues

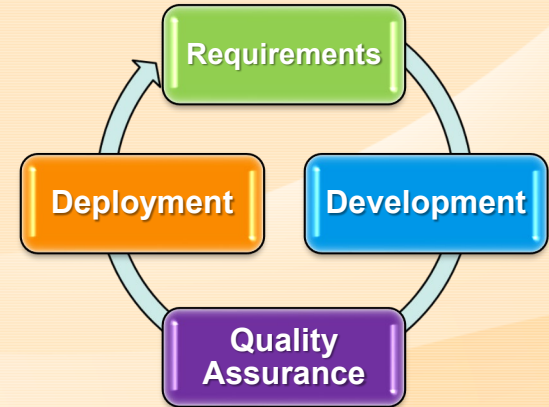


# RAD (Rapid application development)

---

## Cons

- Needs modularized code bases to work well
- Harder to budget and manage overall due to uncertainty
  - No predefined end
  - You don't know how much money is coming in from previous cycles
- Can build technical debt in codebase quickly due to the constant release schedule

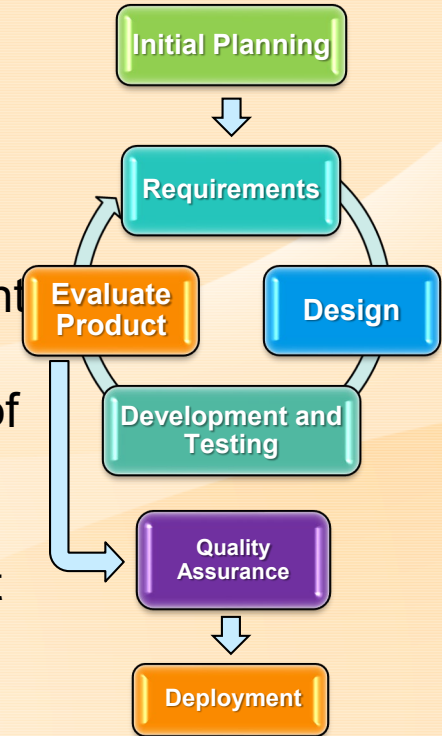




# Agile/Iterative

A cycling development pattern that builds toward a larger end goal

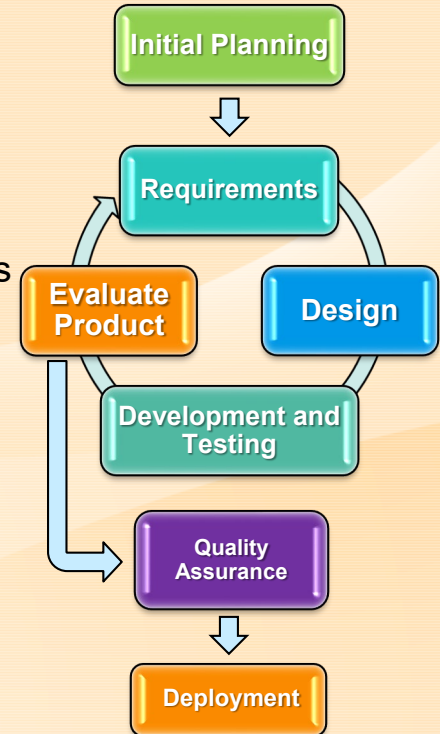
- Breaks the project into incremental builds
- High level design agreed upon up front
- Each iteration intends to be capable of release
- Most common Software Development Methodology at the moment



# Agile/Iterative

## Pros

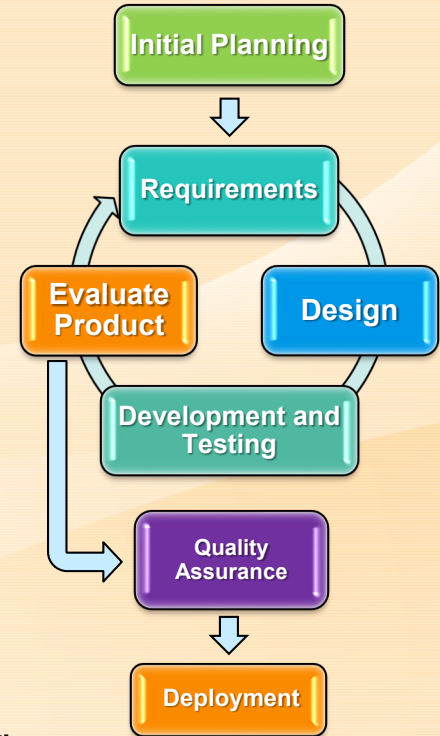
- Flexible, can handle any type of project
- Higher visibility
  - Everyone is involved with planning and stays informed of project progress
  - Individual effort of developers more visible during process
- Built to respond to unexpected changes
  - Less cost involved in redesigns
- Maintains working builds
  - Working build achieved early
  - Development and testing at once



# Agile/Iterative

## Cons

- Significant methodology overhead
  - Teams take several iterations to learn
- Shifting goals requires comprehensive tests on tasks
- Can lead to scope creep
- Can build technical debt in codebase quickly (but not as quickly as RAD)
- Teams frequently don't maintain the method's processes and end up doing it in name only



---

# SCRUM

---

# Scrum main points

---

Scrum is an agile/iterative process

- Gets its name from rugby
  - Formation of players/Teamwork
- Focuses on
  - Self-organizing
  - Daily face-to-face communication
  - Response to changing expectations and unpredictable challenges



# Scrum main points

---

## Scrum Vocab

- “User Story” potential feature to be made or tasks to be completed
- “Product Backlog” a breakdown of the work to be done on the project
  - In userstory form
- “Sprint” the timebox for iterating on builds for the product

# Scrum main points

---

## People

- **Product owner:**  
The person with the role of defining what the product will be and how it will be verified to be completed.
- **Development team:**  
3-9 member team that is creating the product
- **Scrum master:**  
Person who ensures the scrum process is being followed and assists the development team as well as being a buffer between the development team and product owner

---

# User Stories

---



# User Story: What is it?

---

How work to be completed is organized in scrum

As a [USER], I want [FEATURE] to [PURPOSE/VALUE]

- The agreement between the development team and the product owner on what will be created.
- Can be shown to be completed just by using the product

# User Story: Test cases

---

All user stories need a list of test cases / acceptance criteria.

- What will be on screen to prove the work is complete
- Only yes or no questions confirming the state of the product

# User Story: Dependencies

---

All user stories need a list of dependencies.

- What has to exist before work on the userstory can start
- All dependencies should have their own userstories responsible for completing those tasks.

# User Story: Example

---

- “As a user, I want a ninja enemy for the forest levels “
  - Example use:
    - Spawn a ninja off screen when the player comes to that point of the level
    - Walk on screen once created
    - Set itself to an AI patrol state
    - Attack player with a ranged attack if player is within attack range
    - Do a “Ninja Vanish” when player gets in melee attack range



# User Story: Test cases

---

Convert the decided upon aspects into test cases

- Plus any other aspects of work for that feature

Test cases:

- Can a ninja be created though level spawn triggers?
- Can the ninja be killed by the player?
- Can the ninja patrol between two points of a level?
- Does the ninja attack the player with a ranged attack when it sees it?
- Does the ninja vanish when the player is near with a smoke cloud in its wake then the player gets within melee range?
- Does the ninja animate through all of its states?
- Does the ninja play SFX for its actions?

# User Story: Dependencies

---

Decide what groundwork has to be in place before you begin working

Dependencies:

- Player character to respond to
- Path-finding system
- Patrol AI
- Ninja sprite sheet\*
- Ninja SFX\*
- Smoke Particle effect\*

# User Story: Most Common problems

---

Not knowing what you want before writing the userstory

- If the design hasn't been decided upon yet, it must be now.
  - We want a boss. But what does that boss do?

Not having test cases

- Every user story needs tests that can be verified
- Having useless test cases is just as bad
  - User story: Make X
  - Test cases: "Does X function correctly"

# User Story: Most Common problems

---

- Using ambiguous terminology in test cases
- “-ly” words and vague ideas rather than verifiable tasks

## Bad

- “Intelligently”
- “Completely”
- “Balanced”
- “Challenging”
- “Unique”



## Good

- Define how it makes its choices
- Define what parts will be created
- Define what changes will be made to attempt to achieve balance
- Define what will be created to create a challenge
- Define what aspect is unique and how it works

- Focus on what will be created or done that can be verified without opinions



# User Story: Most Common problems

---

- Amount of work per userstory
- Overly -encompassing or Underencompassing userstories

## Bad

- I want all the enemies/bosses
- I want level 5 boss to move left



## Good

- I want a mushroom enemy
- I want a turtle enemy
- I want a ninja enemy
- I want the level 5 boss AI states

- Each userstory should be enough work to be 1 integration and commit

# <Activity> User Stories

---

## User Story Writing

- We have a backlog (though not in user story form yet)
- Let's expand the work in there into userstories

---

# Scrum Setup

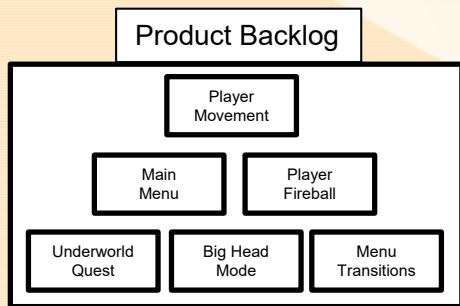
---

# Product backlog

---

Everything that could be in the product is collected into a list called the product back log

- Things can get added to the product back log as needed
- Only a wish list for now, Not promises that need to be fulfilled
- In userstory format

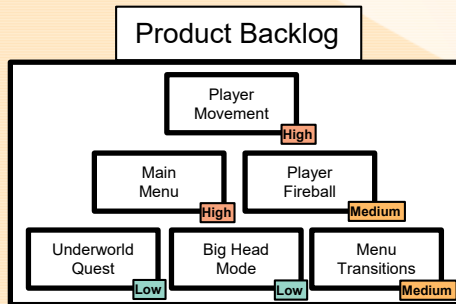


# Product backlog

---

The back log is prioritized according to overall importance to the product, stake holders, and dependencies

- Highest priority things get worked on first
- The things unnecessary get pushed down

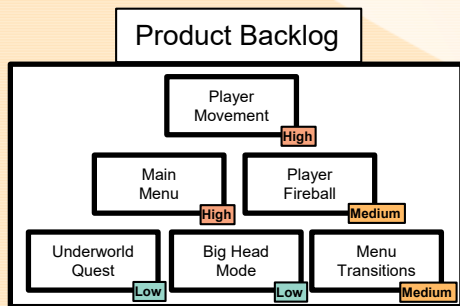


# Product backlog

---

When is this work done?

- The writing of the product backlog initially gets completed during the pre-production phase
- Once complete sprints can start
- The product backlog continues to get developed during open production based on change requests.



---

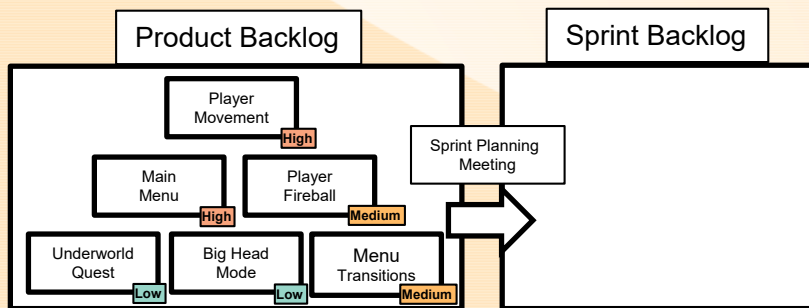
# Sprint Planning

---

# Sprint Backlog

## Sprint Planning

- The entire team will meet to:
- First
  - Determine an overall sprint goal
  - Select stories from the product back log to achieve that goal

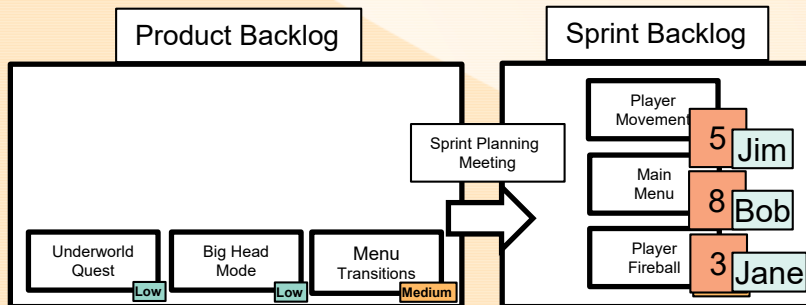




# Sprint Backlog

## Sprint Planning

- The entire team will meet to:
- Second
  - Evaluate the difficulty/hours/complexity of the stories selected
  - Distributing the workload among the team



---

# Planning poker

---

# Planning poker

---

The process of sprint planning

After the userstories have been selected each userstory is evaluated individually by the group

- Estimating workload
- Understanding dependencies
- Assigning tasks

# Planning poker: Step 1: Bidding

---

## Step 1: Bidding

- Userstory and test cases is read out to the team
  - Answers questions if there are any
  - Modify test cases where needed
    - (Client is involved in this for externally produced projects)
- Each team member
  - Evaluates how difficult they believe the story is to completing, without bias from other members
  - Pick which of the possible bids best represents how difficult they evaluate the task to be
  - Done with cards placed face down in person
  - Done with web tools when remote ([planitpoker.com](https://planitpoker.com))

# Planning poker: Step 1: Bidding

---

Bid Value : Estimated Work

- |                 |            |
|-----------------|------------|
| ● 0             |            |
| ● $\frac{1}{2}$ |            |
| ● 1             |            |
| ● 2             | ● 20       |
| ● 3             | ● 40       |
| ● 5             | ● 100      |
| ● 8             | ● Unknown  |
| ● 13            | ● Infinite |

- Normally these are treated as abstract “points” with 0 being the easiest thing the complete and 100 being the hardest
- Without previous experience with making and living with estimations this can be hard to grok and instead we will be estimating in hours

# Planning poker: Step 1: Bidding

Bid Value : Estimated Work

- 0 hr.
- ½ hr.
- 1 hr.
- 2 hrs.
- 3 hrs.
- 5 hrs.
- 8 hrs.
- 13 hrs. (1 day and a half)
- 20 hrs. (half a week)
- 40 hrs. (1 week)
- 100 hrs. (2 weeks)
- Unknown
- Infinite

- The number pattern reflects one of the faults in making estimates
- The larger the estimate the more room for error



# Planning poker: Step 1: Bidding

Bid Value : Estimated Work

- 0 hr.
- ½ hr.
- 1 hr.
- 2 hrs.
- 3 hrs.
- 5 hrs.
- 8 hrs.
- 13 hrs. (1 day and a half)
- 20 hrs. (half a week)
- 40 hrs. (1 week)
- 100 hrs. (2 weeks)
- Unknown
- Infinite

- Each value should be through as a range from the bid below it up
  - Can I get this done in 3? No. Can I get this done in 5?...

1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	3	5		8			13				

# Planning poker: Step 1: Bidding

---

Bid Value : Estimated Work

- 0 hr.
- ½ hr.
- 1 hr.
- 2 hrs.
- 3 hrs.
- 5 hrs.
- 8 hrs.
- 13 hrs. (1 day and a half)
- 20 hrs. (half a week)
- 40 hrs. (1 week)
- 100 hrs. (2 weeks)
- Unknown
- Infinite

Special bids:

- 0: There is zero or an inconsequential amount of work to be done to have this completed.



# Planning poker: Step 1: Bidding

---

Bid Value : Estimated Work

- 0 hr.
- ½ hr.
- 1 hr.
- 2 hrs.
- 3 hrs.
- 5 hrs.
- 8 hrs.
- 13 hrs. (1 day and a half)
- 20 hrs. (half a week)
- 40 hrs. (1 week)
- 100 hrs. (2 weeks)
- Unknown
- Infinite

Special bids:

- Unknown: When there is not enough information to make a bid.
- Cannot be assigned to a sprint without that question answered first.

# Planning poker: Step 1: Bidding

---

Bid Value : Estimated Work

- 0 hr.
- ½ hr.
- 1 hr.
- 2 hrs.
- 3 hrs.
- 5 hrs.
- 8 hrs.
- 13 hrs. (1 day and a half)
- 20 hrs. (half a week)
- 40 hrs. (1 week)
- 100 hrs. (2 weeks)
- Unknown
- Infinite

Special bids:

- Infinite: The user story is completely understood, but will never be able to be completed during a sprint.

# Planning poker: Step 1: Bidding

---

## Avoid Bias

- This first step (bidding) must be done in a vacuum
  - Allows everyone to think about the story
  - Gives people a place to defend and forces them to make their estimate for a reason
- Helps avoid “group think”
  - Bob thinks it is X so I guess it must be X

# Planning poker: Step 2: Negotiation

---

## Step 2: Negotiation

- Each team member reveals what bid they decided upon on the previous step at the same time
- If bids differ the team must discuss why and come to an agreement on the task's value

# Planning poker: Step 3: Allocation

---

## Step 3: Allocation

- After every user story has agreed upon values, user stories must have owners committed to them.
- The story's owner will be the person
  - Best equipped to tackle the story
  - Responsible for completing all task related to the story before the end of the sprint

# Planning poker: Step 3: Allocation

---

## Balance the workload

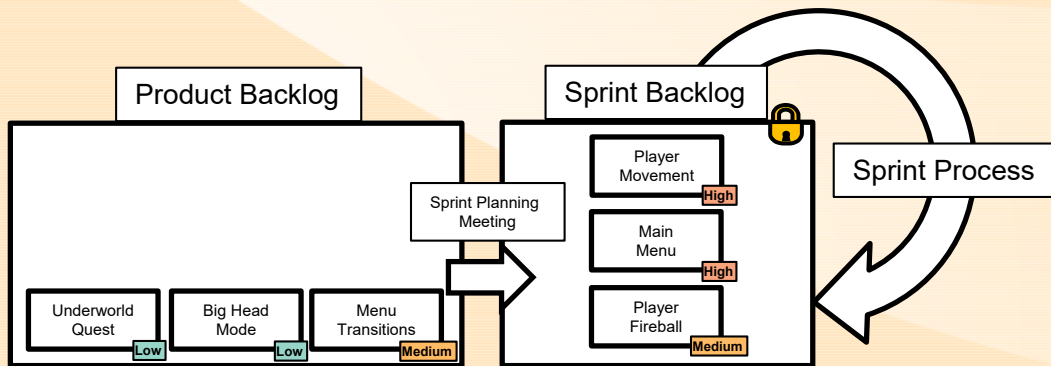
- Make sure each team member is contributing equally
  - Redistribute stories if they are not
- Make sure the workload matches up with the sprint length
  - Not enough hours to fill the schedule = take more stories from the product backlog
  - Over hours = Over hours = Discuss with the product owner about returning things to the product back log or pull back on the sprint goal

# Sprint Backlog

## Sprint Planning

- Once the sprint planning is completed and the sprint has started a commitment has been made for those tasks

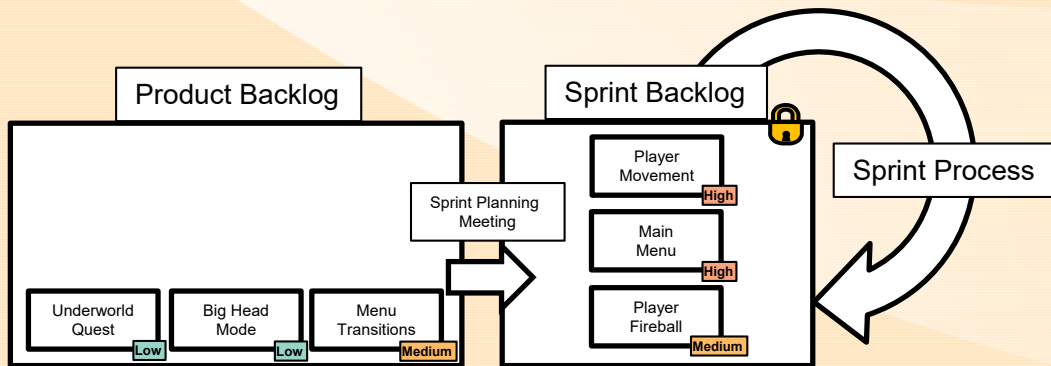
Important note: This is the agreement between the product owner and the developer on what will be done and how it will be verified.



# Sprint Backlog

## Sprint Planning

- Neither the product owner nor the developers should change a sprint plan once in motion
  - Change requests get handled through userstories for future sprints





---

# Sprint planning with our tools

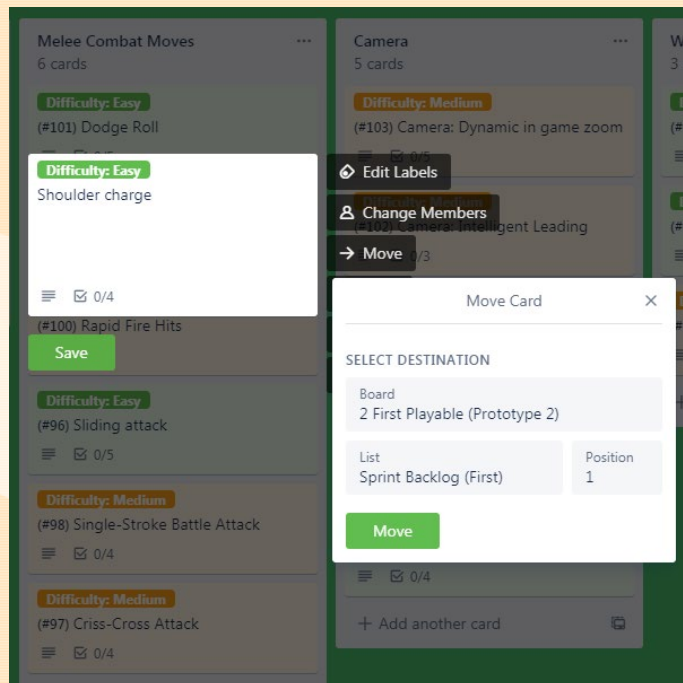
---

# Process in Trello

- First

Move userstories from the product backlog to the upcoming sprint

- Making edits where needed
- Writing new userstories if we are missing the ones we need



# Process in Trello

- Second
  - Go through the planning poker process on each userstory

The screenshot displays a Trello board with a user story card and a planning poker session in progress.

**User Story Card:**

- Title:** As a user I want, the game to contain a completion scenario (winning the game, achieving achievements, high score)
- Labels:** in list *Sprint Backlog (Alpha)*
- Buttons:** ☐ Recurring, Add #tags, S/E & More
- Description:** Intent: Create the win condition and all of the experience we intend to provide with it
- Dependencies:** 0% progress bar, Add an item button
- Task List:** 0% progress bar, Add an item button
- Actions:** 00:00:00

**Planning Poker Session:**

- Card Title:** As a user I want, the game to contain a completion scenario (winning the game, achieving achievements, high score)
- Grid:** A 4x4 grid of cards with values: 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100, and a question mark (?). A coffee cup icon is at the bottom center.
- Players:** Waiting on 8 players to vote. A list of 8 player avatars is shown with a 00:00:24 timer.
- Buttons:** Reset Timer, Flip Cards, Clear, Skip

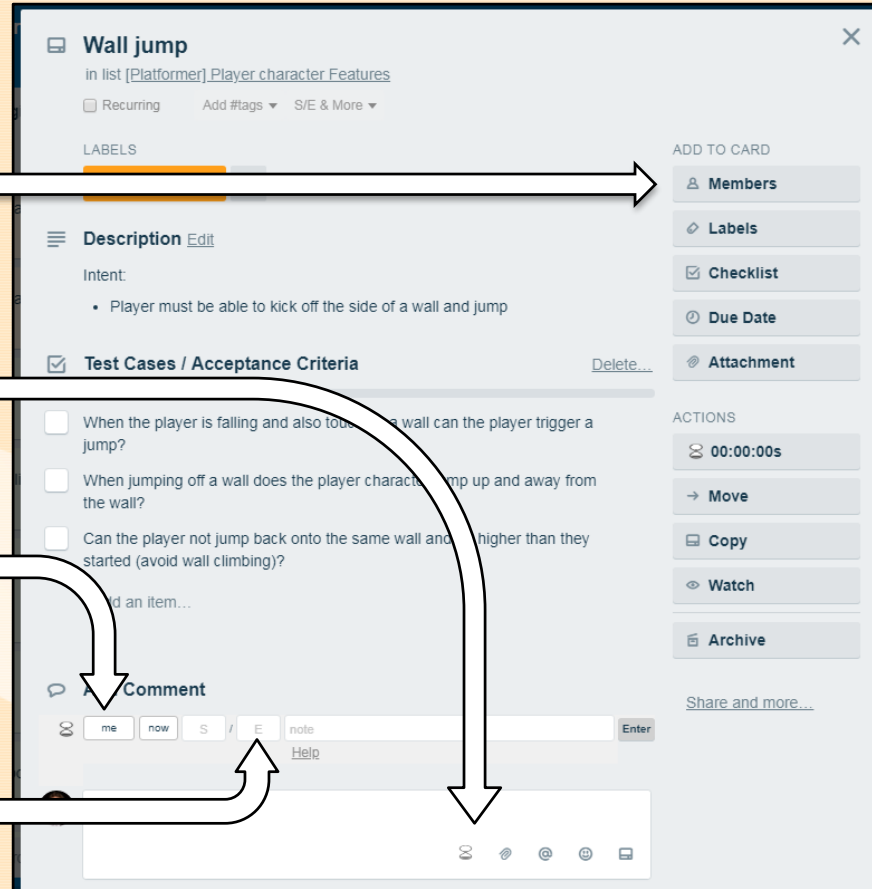
# Logging Hours with Trello

Add yourself as a member of any card you are responsible for

Click the hourglass to start logging hours if interface isn't already visible

The person who is taking ownership of the task and hours.  
Defaults to "me"; the person entering the hours on the card

Log the hours here  
E for estimate (sprint planning)  
S for time spent (tracking your progress)





# <Activity> Planning poker

---

Planning poker estimates

- We have a couple userstories, let's practice estimations on those.

---

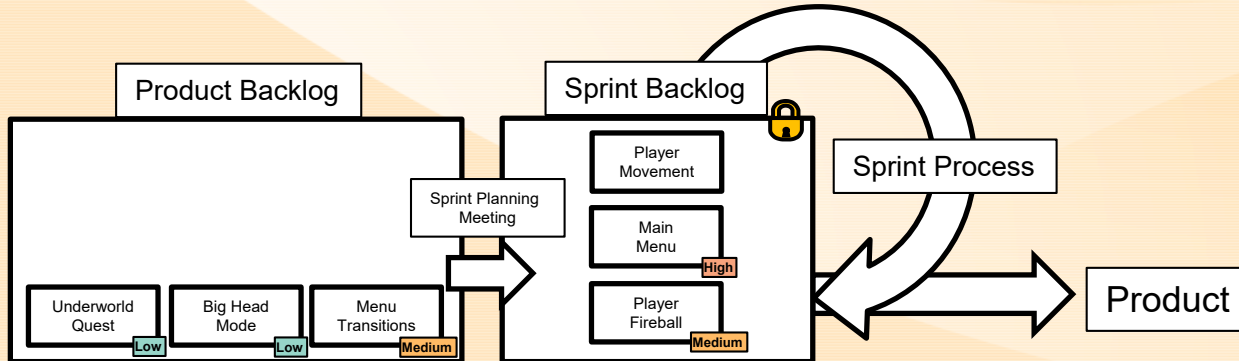
# Sprint Process

---

# Sprint Process

Teams then work through the sprint to complete the agreed upon tasks

- Completing the tasks
- Integrating into the master build

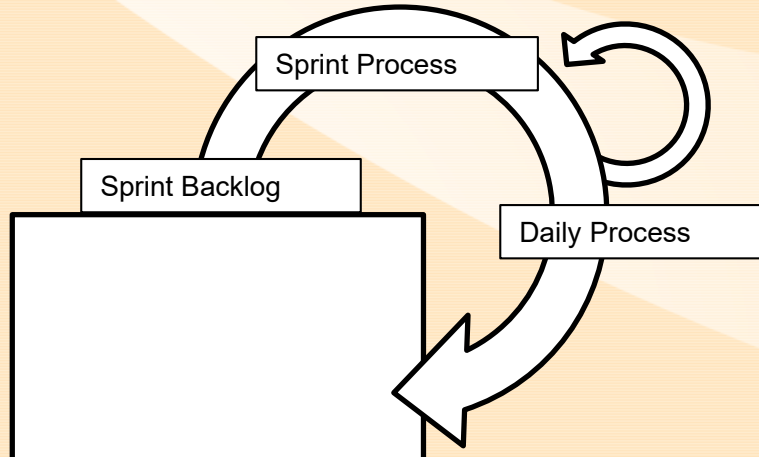


# Workday in scrum

---

An iteration occurs each day within scrum

- Daily Meeting
- Work on tasks
- Track Progress



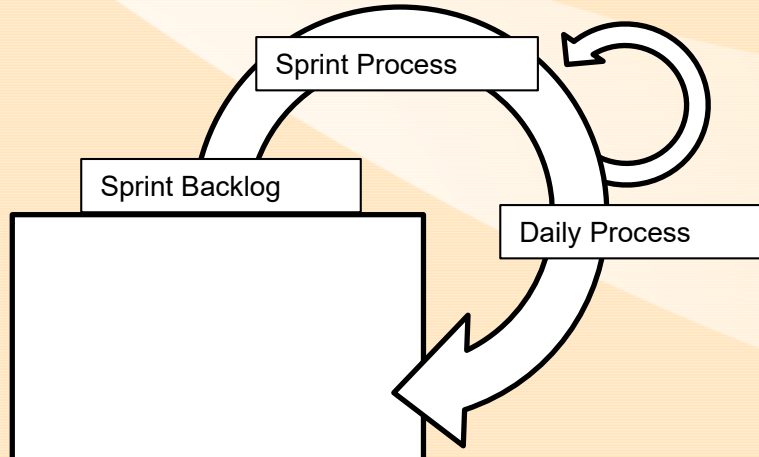


# Workday in scrum

---

Teams meet every day for a scrum “stand up” meeting

- Maintain transparency
- Hold each other accountable
- Set up help when needed

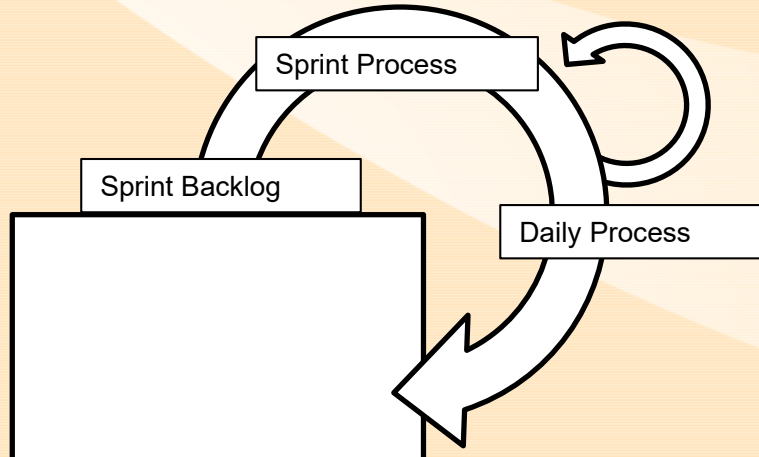


# Workday in scrum

---

Key points of scrum “stand up” meetings

- The meeting should be the start of our working day
- Maximum of 15 minutes.

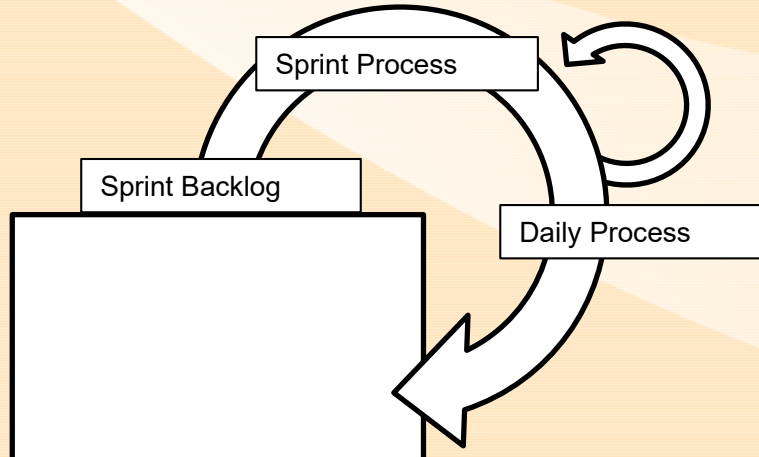


# Workday in scrum

---

The daily meeting needs to answer the following for each team member

- What did you do?
- What are you about to do?
- What currently stands in your way?

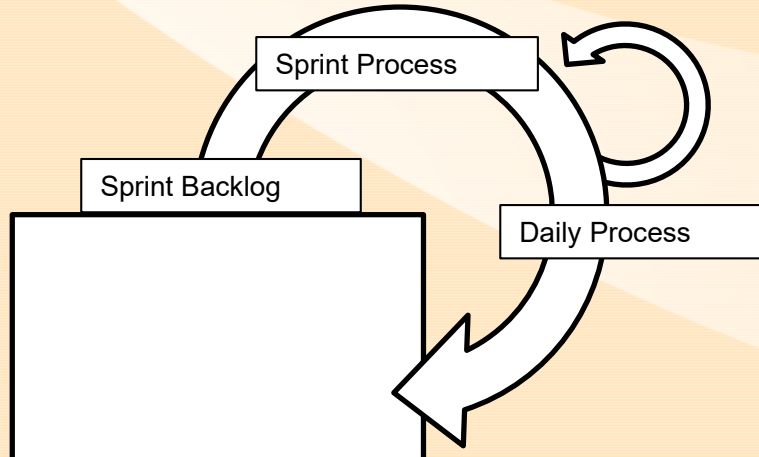


# Workday in scrum

---

After the meeting

- The team breaks apart to work on assigned tasks
- Longer follow-ups happens individually

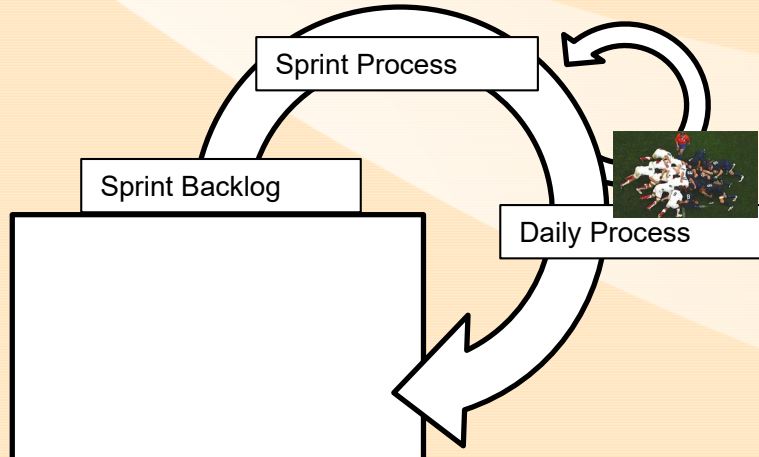


# Workday in scrum

---

Continue to work until the end of the agreed upon workday

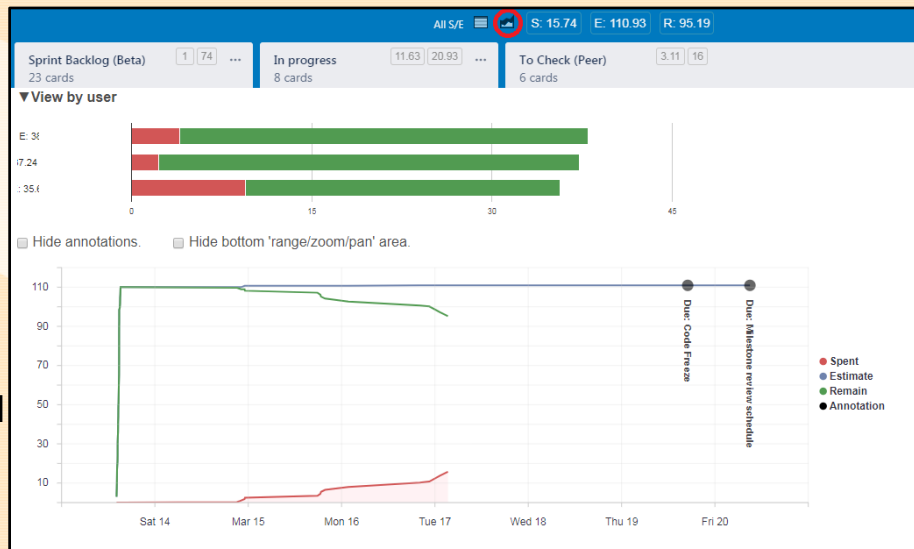
- Integrate the work
- Update task tracking



# Stand up meeting: Trello

Keep everything up to date

- Update the task board (trello) while working
- Enter hours spent on any userstory worked on
- The burn down chart will be updated automatically as hours spent get entered

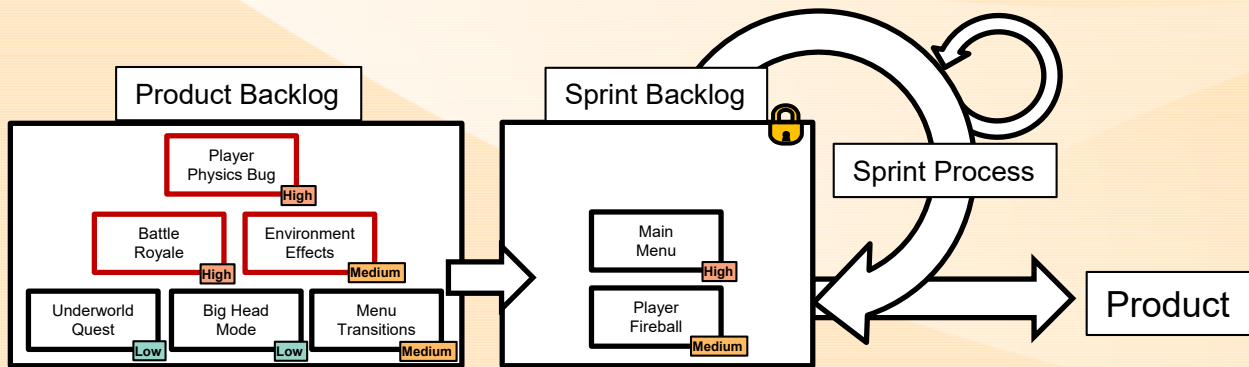


This team is behind, and they should know it so they can respond to it

# Change Requests

During the sprint, things are added to the product backlog if

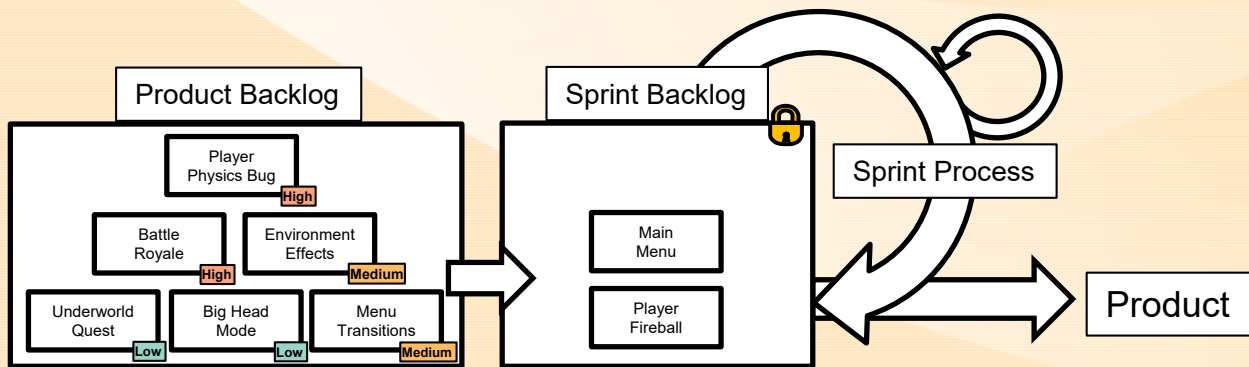
- Discovered to make the product better
- Added from outside influences
- Changes in product expectation from client



# Sprint Review

At the end of each sprint the product is delivered to stakeholders in marketable state

- Review the work that was completed
- Discuss the work that was not completed
- Incomplete work cannot be demonstrated





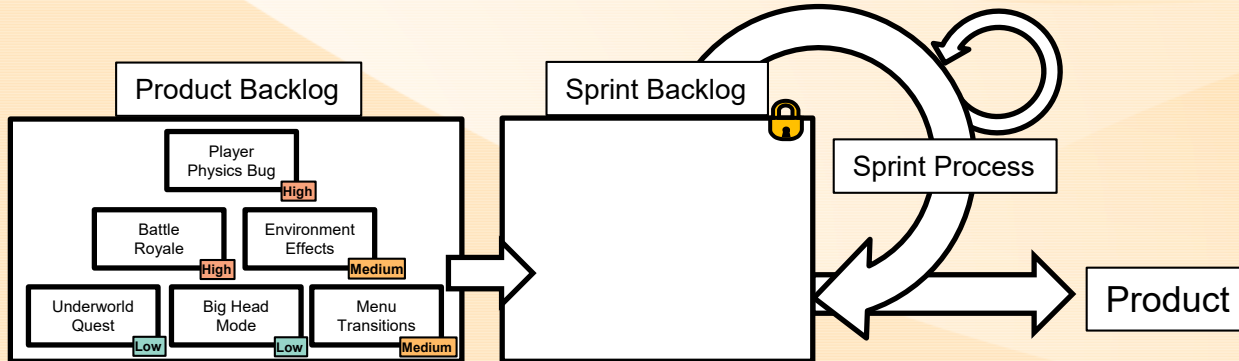
# Sprint Retrospective

Reflects on the past sprint

What went well?

What went poorly?

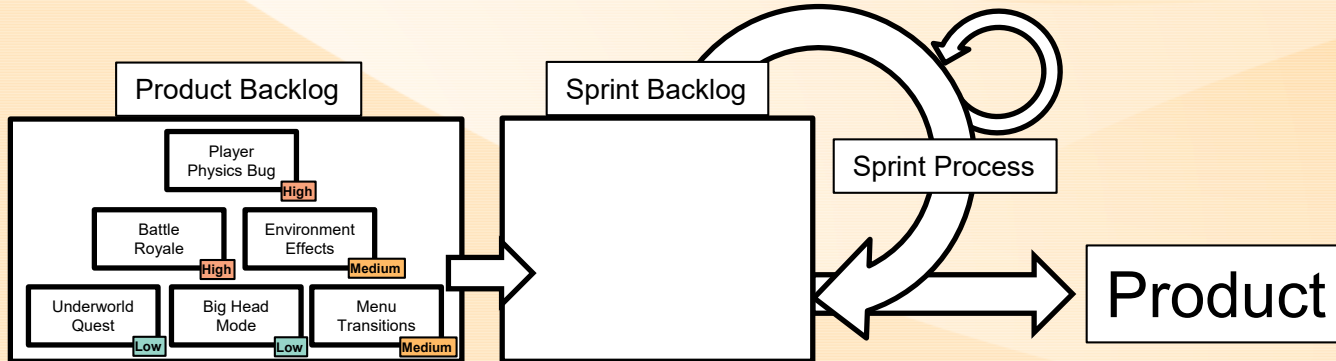
How can we improve?



# Sprint Process

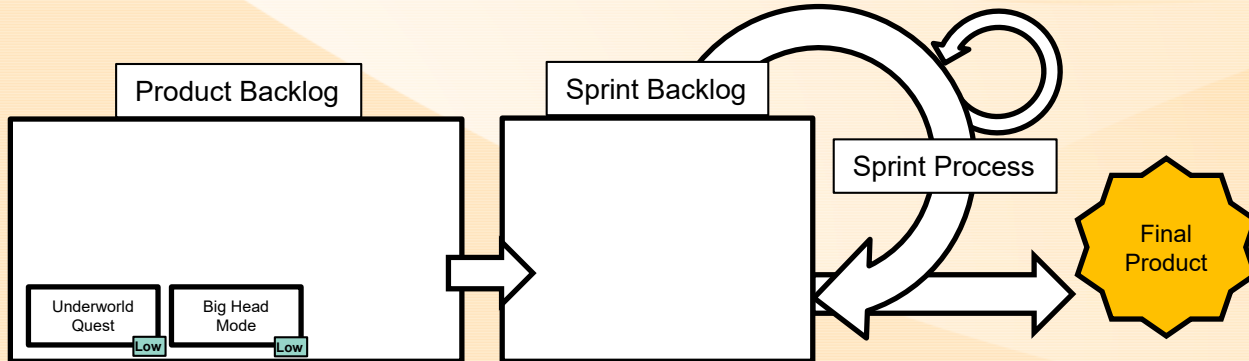
Over the course of multiple sprints

- The product backlog get smaller
- The end product gets better and more feature rich



# Sprint Process

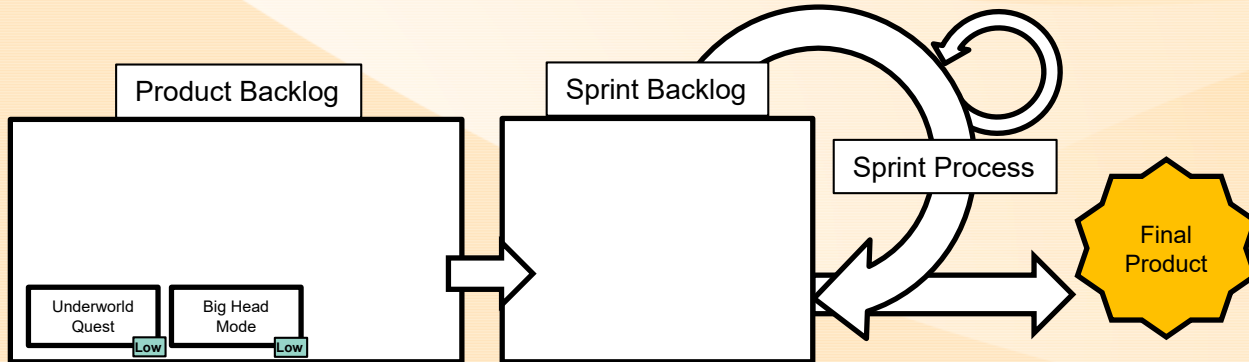
- Eventually this leads to the product that will be released



# Sprint Process

Stories may be left in the back log at release if

- They are deemed unnecessary
- Planned for further updates/patches
- Put off for a sequel



---

# Assignments

---

# Pre-Pro Assignments

---

## Design Document

- Rework and revise the document based on feedback

## Product backlog (core and extended)

- Continue to break down tasks
- Full user story format
- Test Cases
- Dependencies

## ● Code Architecture

- Research the engine
- Plan the major sections

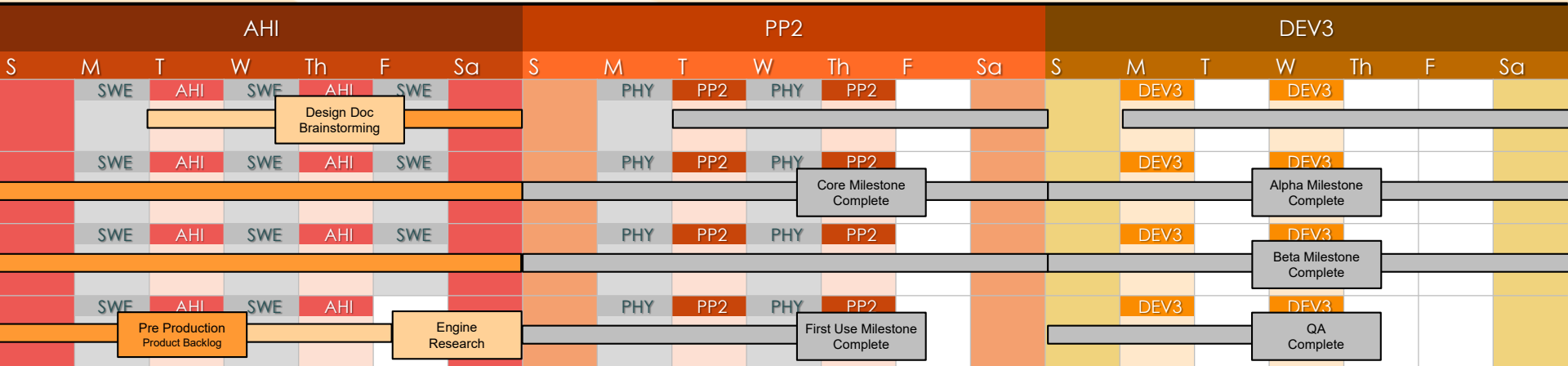
# Pre-Pro Assignments

By the start of lecture 7

- Design documentation rework
- Product back log filled completely
  - Story card
  - Test cases and Dependencies

Before PP2

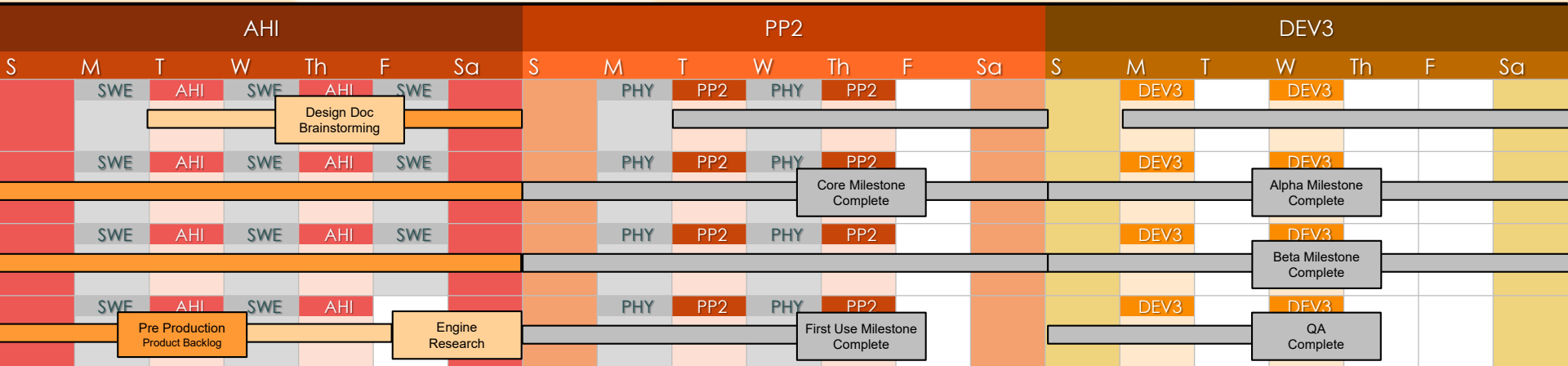
- Unity engine research



# Pre-Pro Assignments

Why so long?

- It is a lot of work
- You will have other assignments as well
  - Those assignments will help you get better understanding that will affect the design
- Gives you the opportunity for multiple rounds of feedback and revision





# Additional Resources

---

This was the basics

Scrum has an entire industry around teaching and use.

- Consulting firms teaching it
- You can be scrum certified

If you would like to know more there is a lot of

- [scrumguides.org](https://www.scrumguides.org)
- [scrum.org](https://www.scrum.org)
- [scrumalliance.org](https://www.scrumalliance.org)

# Engine Research

---

- Familiarize yourself with Unity engine
  - <https://docs.unity3d.com/Manual/>
  - <https://docs.unity3d.com/Manual/ExecutionOrder.html>
- Make prototypes and experiment
  - Unity has really well-made tutorials and documentation
  - <https://unity3d.com/learn/tutorials>
  - (Roll-a-ball tutorial, Space Shooter tutorial, Mini Tutorials are good starts)
- Have an idea on how things will be assembled for that game you are making
  - Code architecture understanding