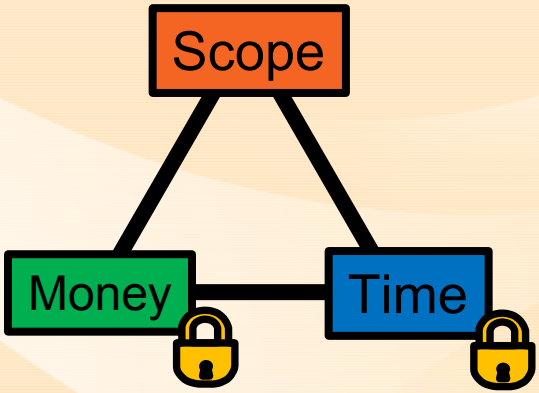
Scheduling

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

Scope

Third: Identify what takes priority

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

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
  - Paper and electronic prototypes are created, tested, and prove the idea works

Phases of Production: 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/Gant charts

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

Phases: Open Production

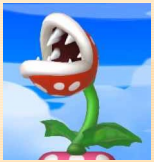
- Planning is done lets get building
- Obvious
  - Complete tasks
  - Verify completeness
- Change requests

Phases: Milestone: First Use/Playable

Useable product

- Play/Use testing can start
  - Product must be able to sell itself
- A completed single level that displays most Global aspects of product in a Local environment.
  - Global: Things necessary for every portion of the product



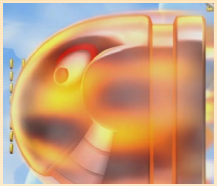



  - Local: Things only necessary for specific portions







Phases of Production: Alpha

Completed the construction of all features

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

Phases of Production: Beta

Finalizing content for the product

- All placeholders and temporary content replaced with final quality versions
- Balancing and play testing completed
- Removed all debugging tools

Phases of Production: Gold!

Released

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

Phases of Production: Post Release

In the users hands

- Lessons learned
- Archiving
- Reviews come out
  - Internet hate machine turns on
- Bug fixes continue

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

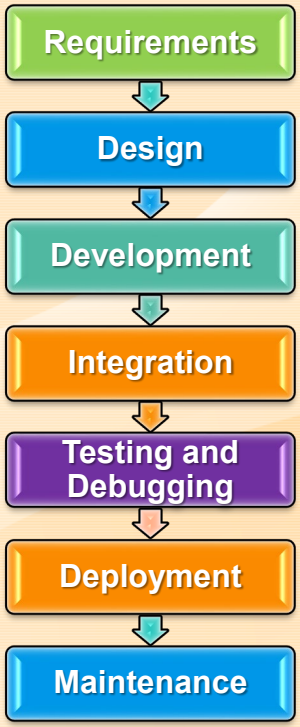
There are many different development methodologies

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

Waterfall (a.k.a. Traditional)

A linear stage based model

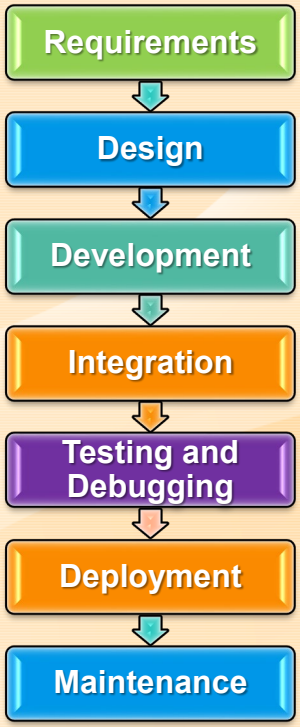
- Requirements are well documented, clear, and fixed
  - Full production and milestone schedule assembled during initial planning
- Move from one phase to the next only after it is reviewed and verified



Waterfall (a.k.a. Traditional)

Pros

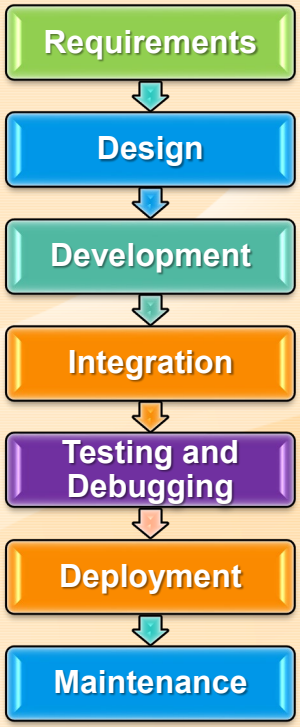
- Simple to understand
- Each stage has a definitive focus that improves the quality of the output
- Little to no methodology overhead



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)

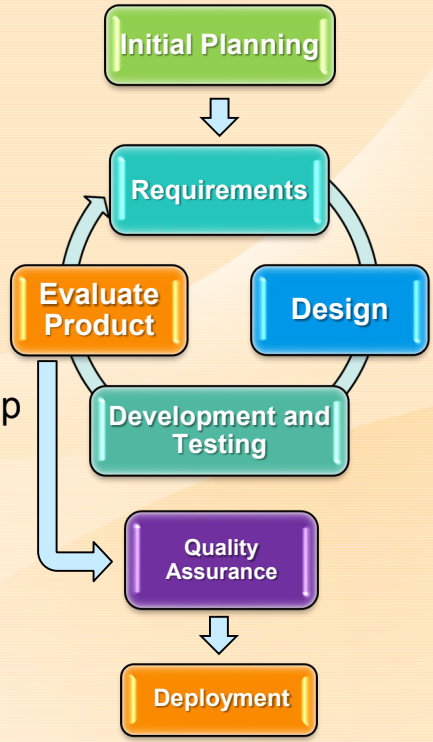
Cons

- Needs modularized code bases to work well
- Harder to budget and manage
  - No predefined end
- Can build technical debt in codebase quickly

Agile/Iterative

A cycling development pattern that builds toward a larger end goal

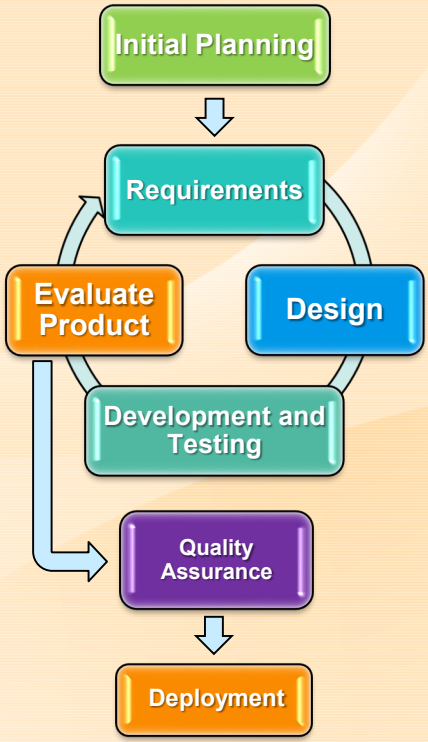
- Breaks the project into incremental builds
- High level design agreed upon up front
- Each iteration intends to be capable of release



Agile/Iterative

Pros

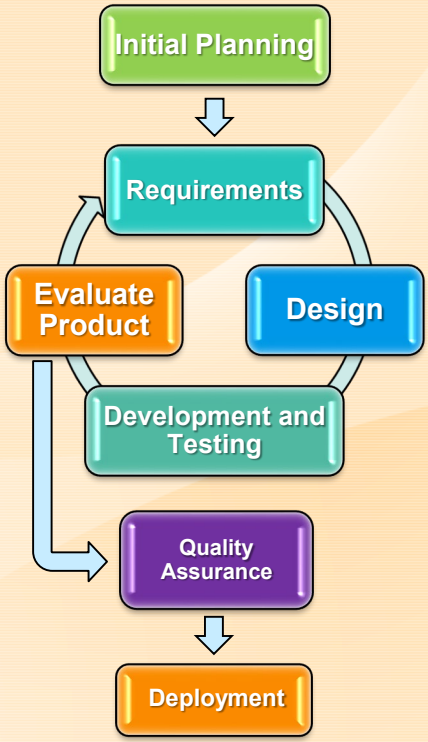
- Flexible
- Higher visibility
  - Everyone is involved with planning and stays informed of project progress
  - Individual effort of developers more visible during process
- Built to respond to unexpected changes
  - Less cost involved in redesigns
- Maintains working builds
  - Working build achieved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



Agile/Iterative

Cons

- Significant methodology overhead
  - Teams take several iterations to learn
- Shifting goals requires comprehensive tests
- Can lead to scope creep
- Can build technical debt in codebase quickly
- Teams frequently don't maintain the methods processes



SCRUM

Scrum main points

Scrum is an agile/iterative process

- Features and tasks are documented in a “Product Backlog”
- The team collaborates to plan the “sprints”
- Teams keep each other informed daily
- At the end of each sprint teams have a marketable product

User Stories

User Story: What is it?

How work is organized in scrum

- 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: What are they for?

User stories reference things more fully designed and laid out in other documentation

- “As a user, I want a ninja enemy “
  - Example use:
    - Spawn a ninja
    - Walk on screen once created
    - Set to an AI patrol state
    - Attack player when seen
    - Vanish when player is near

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

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

User Story: Test cases

“As a user, I want a ninja enemy “

Test cases:

- Can a ninja be created?
- Can the ninja be killed?
- Can the ninja patrol?
- Does the ninja attack the player when it sees it?
- Does the ninja vanish when the player in near with a smoke cloud in its wake?
- Does the ninja animate through all of its states?
- Does the ninja play SFX for its actions?
- …

User Story: Dependencies

All user stories need a list of dependencies.

- What has to exist before work on the userstory can start

User Story: Dependencies

“As a user, I want a ninja enemy “

Dependencies:

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

*All dependencies should have their own userstories responsible for completing those tasks.

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

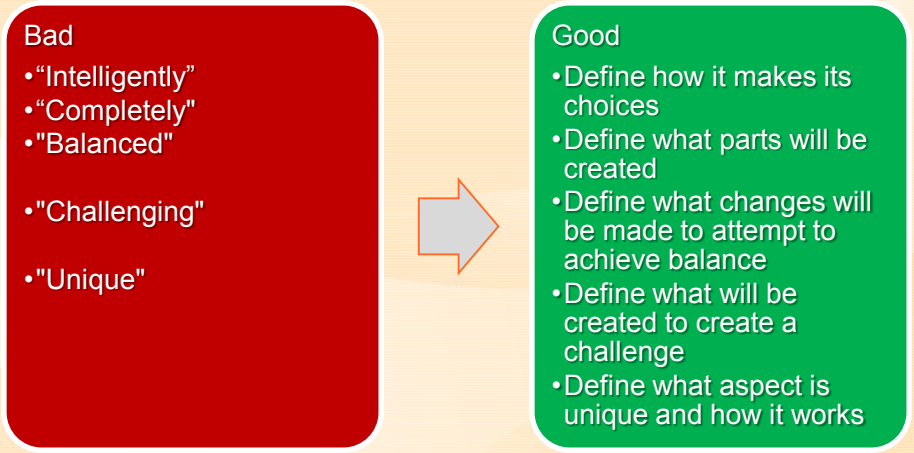
- Every user story needs tests that can be verified

- 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

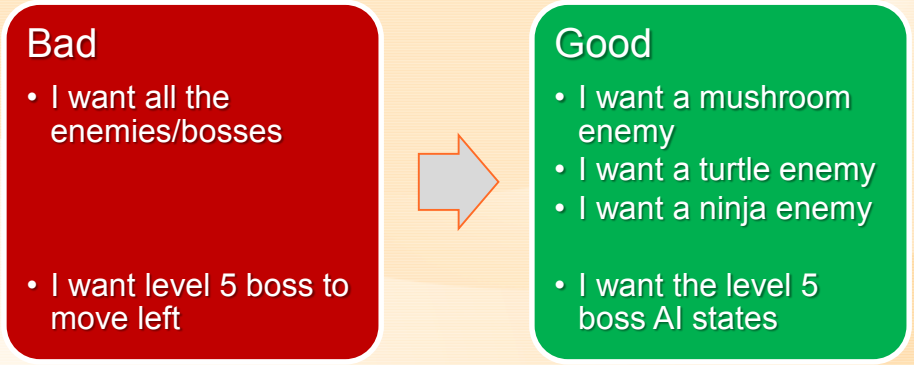


- Focus on what will be created or done that can be verified without opinions

User Story: Most Common problems

Amount of work per userstory

- Overly-encompassing or Under-encompassing userstories



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

<Activity> User Stories

User Story Writing

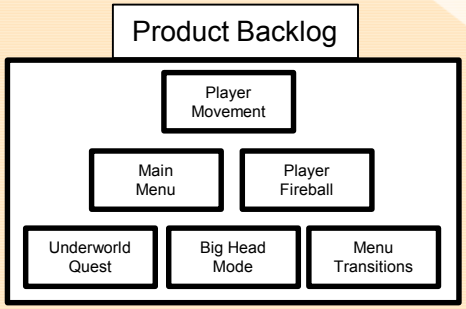
- We have a backlog (though not in user story form yet)
- Let's expand the work in there into userstories

Scrum Setup

Product backlog

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

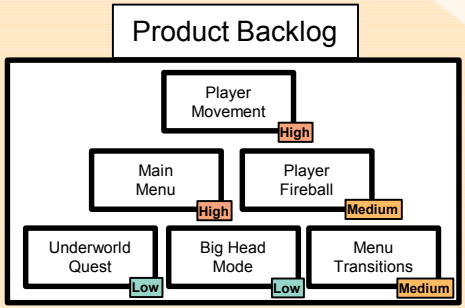
- Things can get added to the product back log as needed
- Only a wish list for now, Not promises that need to be fulfilled



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

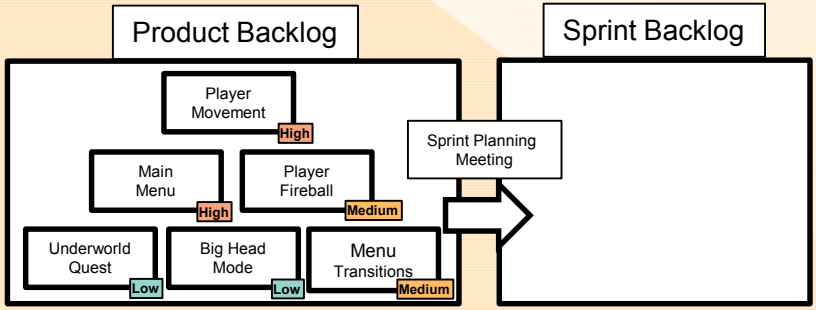


Sprint Planning

Sprint Backlog

Sprint Planning

- Before the sprint the entire team will meet to:
  - Determine an overall sprint goal
  - Select stories from the product back log to achieve that goal
  - Evaluate the difficulty/hours/complexity of the stories selected
  - Distributing the work load among the team



Planning poker

Planning poker

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)

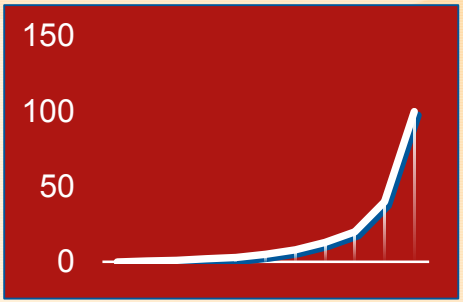
- Each team member
  - Evaluates how difficult they believe the story is to completing, without bias from other members
  - Pick which of the possible bids best represents how difficult they evaluate the task to be

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
  - Bid of 5 = anything above 3 up to 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.

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

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

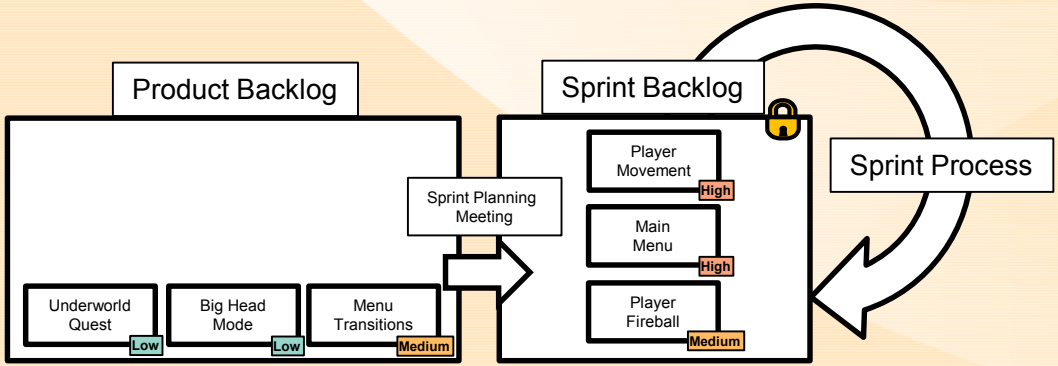
Balance the workload

- Make sure each team member is contributing equally
  - Redistribute stories if they are not
- Make sure the work load matches up with the sprint length
  - Not enough hours to fill the schedule = take more stories from the product backlog
  - Over hours = Discuss with the product owner to return things to 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
- Neither the product owner nor the developers should change a sprint plan once in motion



<Activity> Planning poker

Planning poker estimates

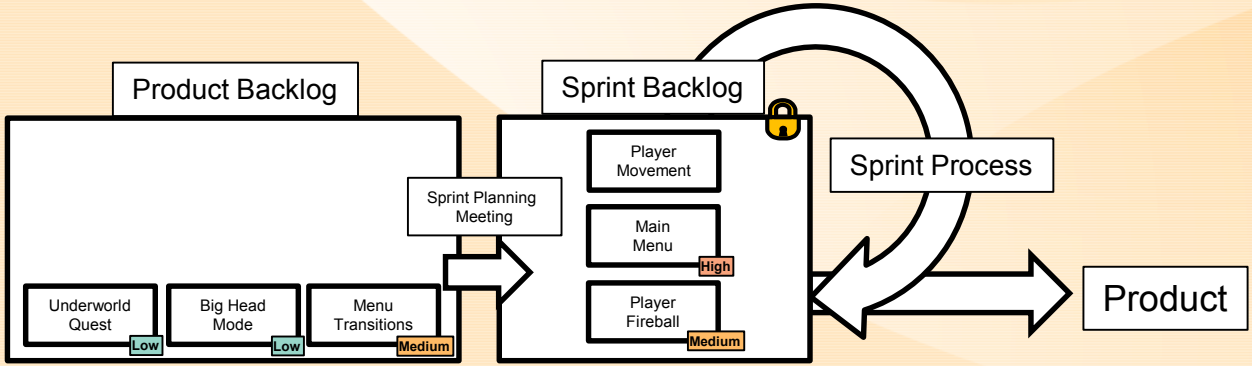
- We have a couple userstories, let's practice estimations on those.

Sprint Process

Sprint Process

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

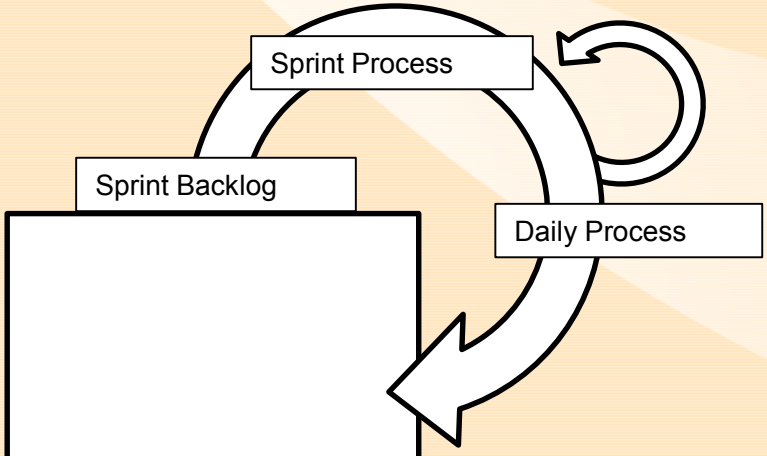
- Completing the tasks
- Integrating into the master build



Work day in scrum

An iteration occurs each day within scrum

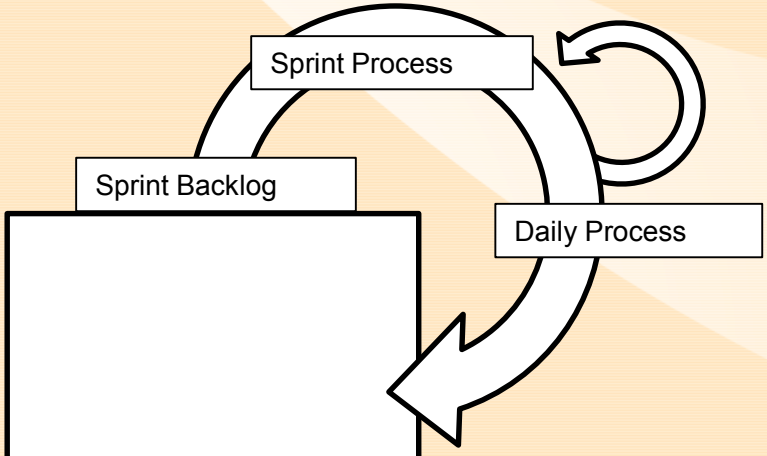
- Daily Meeting
- Work on tasks
- Track Progress



Work day in scrum

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

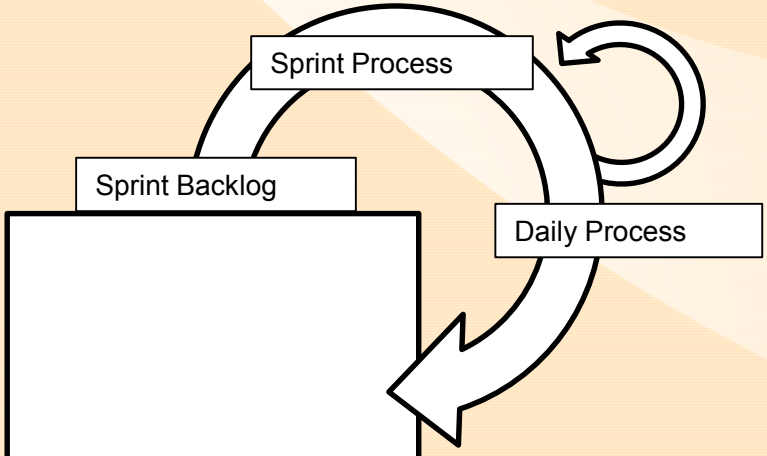
- Maintain transparency
- Hold each other accountable
- Set up help when needed



Work day in scrum

Key points of scrum “stand up” meetings

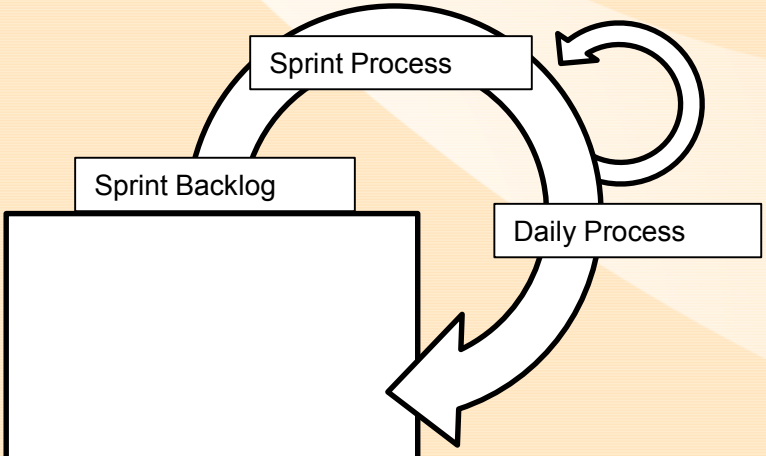
- The meeting should be the start of our working day
- Maximum of 15 minutes.



Work day in scrum

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

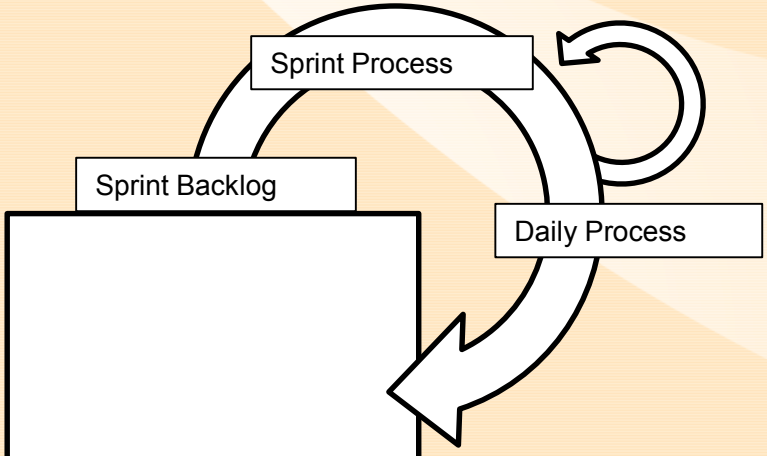
- What did you do?
- What are you about to do?
- What currently stands in your way?



Work day in scrum

After the meeting

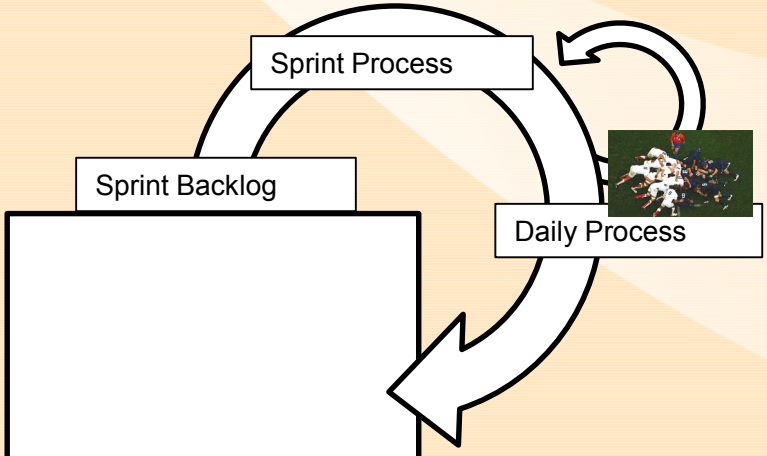
- The team breaks apart to work on assigned tasks
- Longer follow-ups happens individually



Work day in scrum

Continue to work until the end of the agreed upon work day

- Integrate the work
- Update task tracking



Logging Hours with Trello

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

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

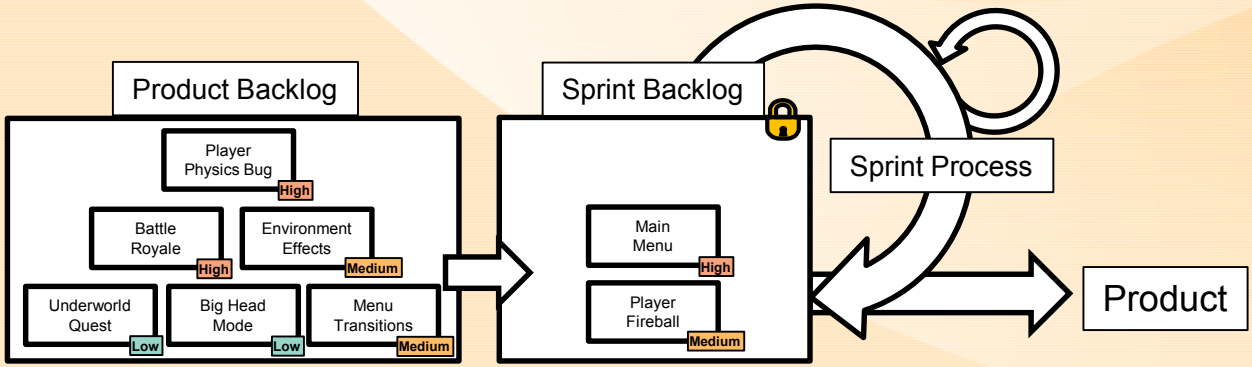
The person who is taking ownership of the task and hours.
Defaults to “me”; the person entering the hours on the card

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

Sprint Process

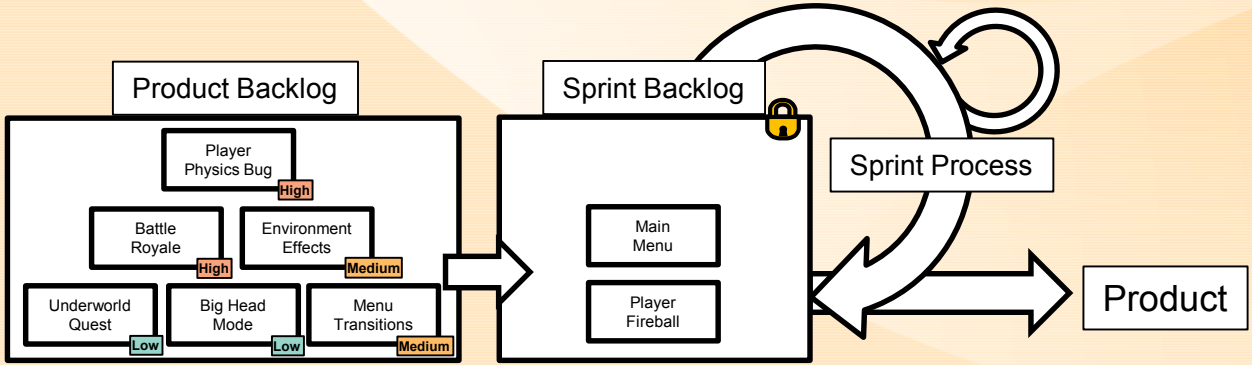
During the sprint, things are added to the product backlog if

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



Sprint Process

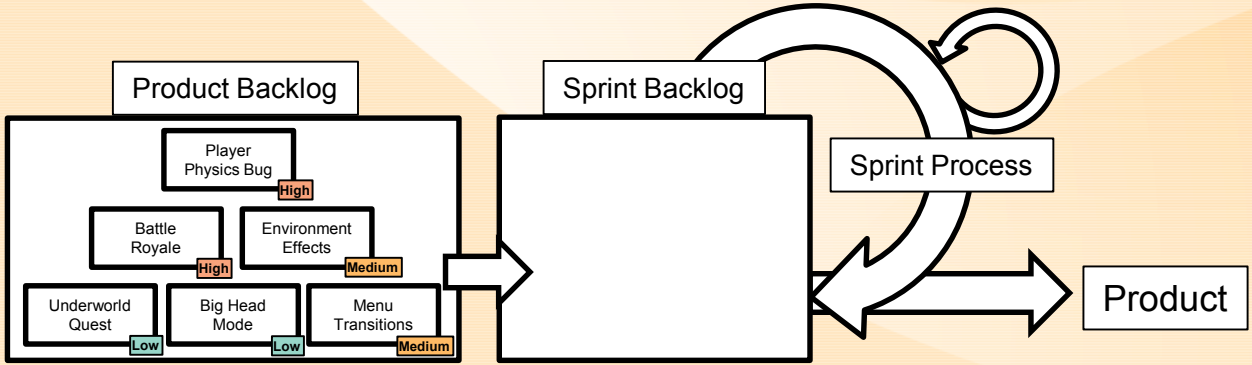
At the end of each sprint the product is delivered in marketable state



Sprint Process

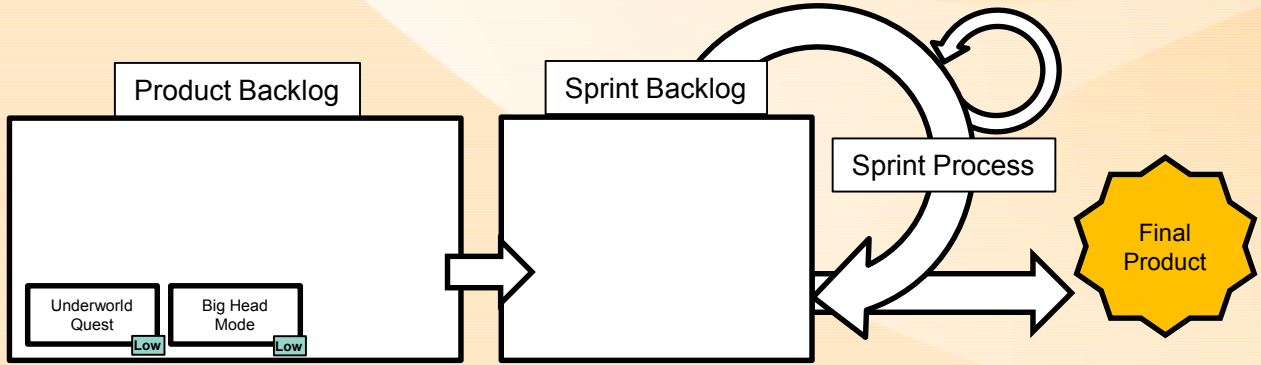
Over the course of multiple sprints

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



Sprint Process

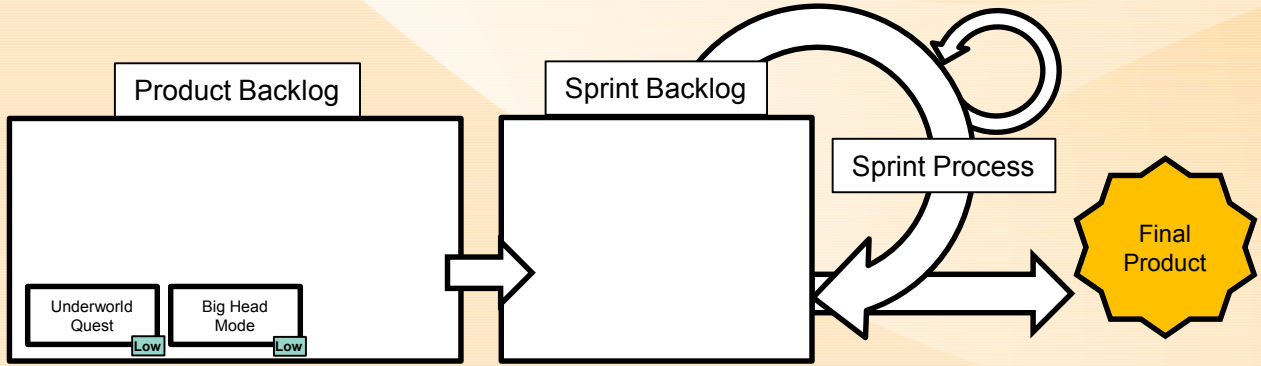
- Eventually this leads to a the product that will be released



Sprint Process

Stories may be left in the back log at release if

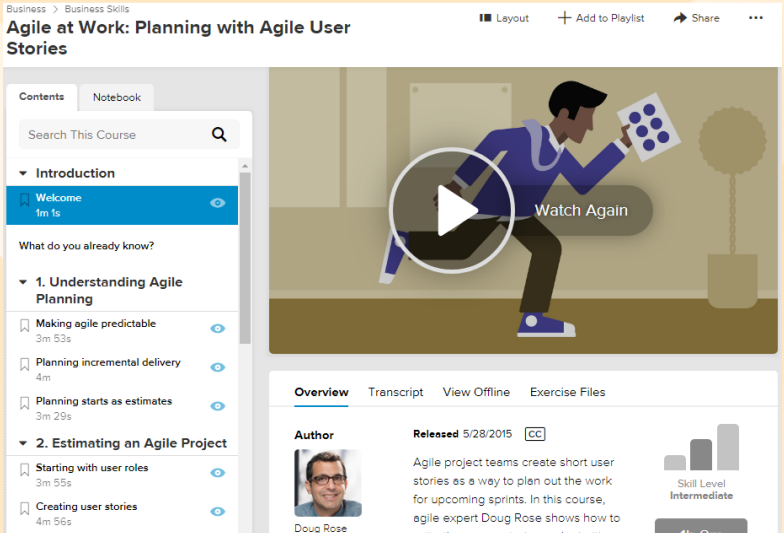
- They are deemed unnecessary
- Planned for further updates/patches
- Put off for a sequel



Additional Resources

Doug Rose: Agile at Work—Planning with Agile User Stories

- http://www.lynda.com/Business-Skills-tutorials/Agile-Work-Planning-Agile-User-Stories/175074-2.html



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

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

Work on product backlog in Trello

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