SOFTWARE DEVELOPMENT METHODOLOGIES

Methodologies

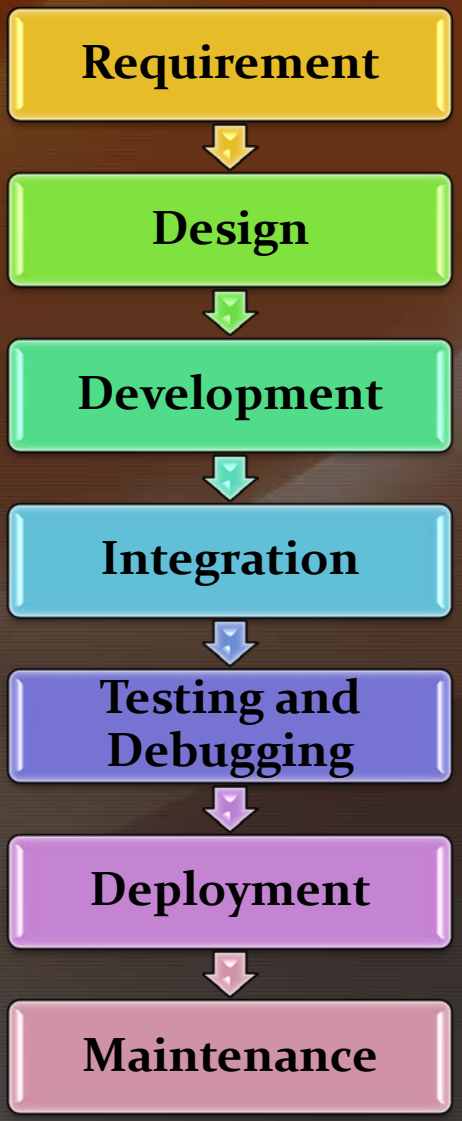
Different methods

- There are many different development methodologies
- Most popular seem to be
  - Waterfall
  - RAD
  - Agile/Iterative

Waterfall (a.k.a. Traditional)

Methodologies

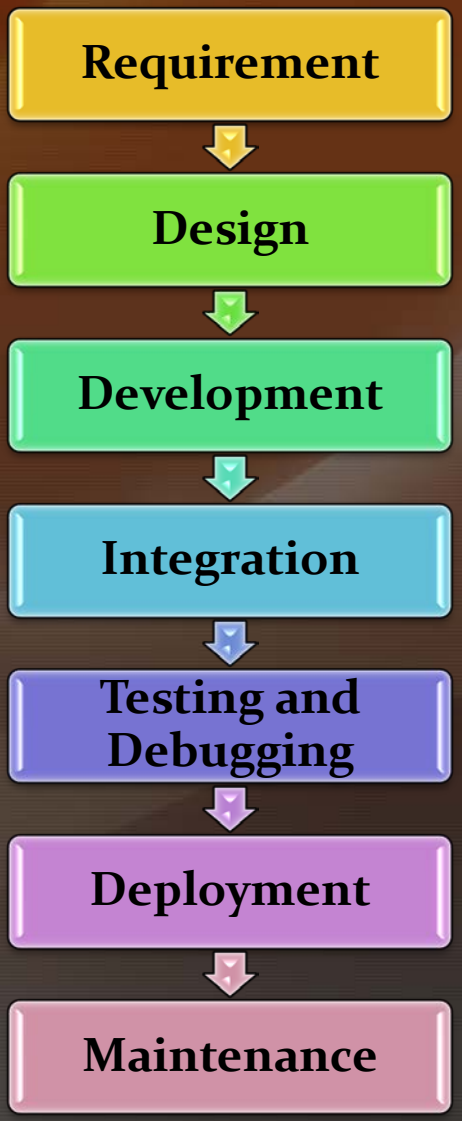
- A linear stage based model
  - One single straight path to the goal.
- Requirements are well documented, clear, and fixed
  - Full production and milestone schedule assembled during initial planning
- Emphasis on
  - Planning
  - Target dates
  - Budgets



Waterfall (a.k.a. Traditional)

Methodologies

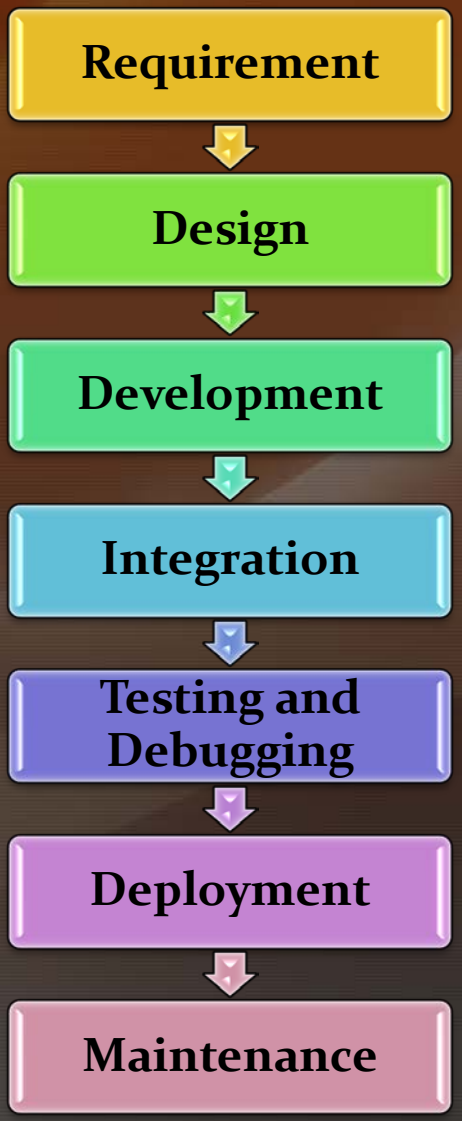
- Pros
  - Simple to understand
  - Clearly defined stages
  - Each stage has a definitive focus that improves the quality of the output
  - Little to no methodology overhead



Waterfall (a.k.a. Traditional)

Methodologies

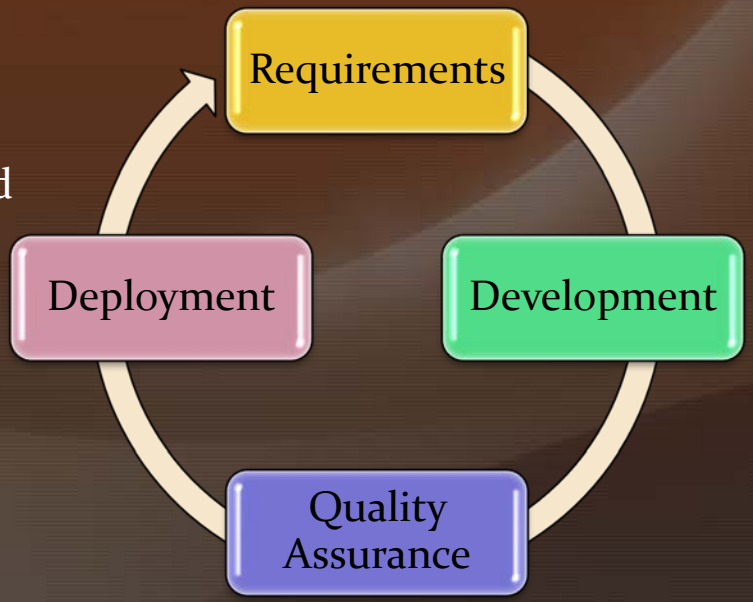
- Cons
  - All requirement analysis and design must be done up front
  - Hard to estimate times accurately
    - Everything is large scale
  - Does not handle changes or revisions during development well
    - No built in course correcting
  - No working build until late in the process
    - No built in integration requirements until the very end
  - Can cause disciplines to become idle



Methodologies

RAD (Rapid application development)

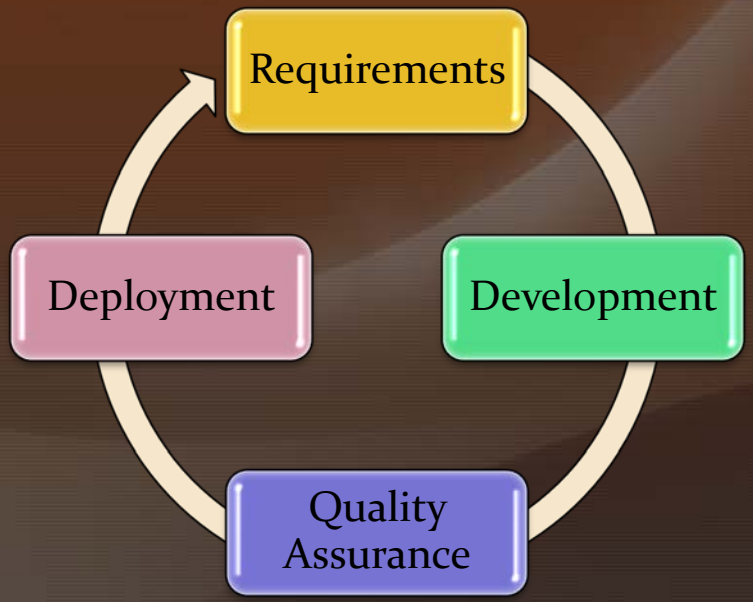
- A cycling development pattern getting functioning software out as fast as possible
- Product is delivered in an incremental manner
  - Single or few features completed on each cycle
- Best for small or already established products or prototypes
- Emphasis on
  - Customer feedback
  - Product modifications
  - Fast delivery of features



Methodologies

RAD (Rapid application development)

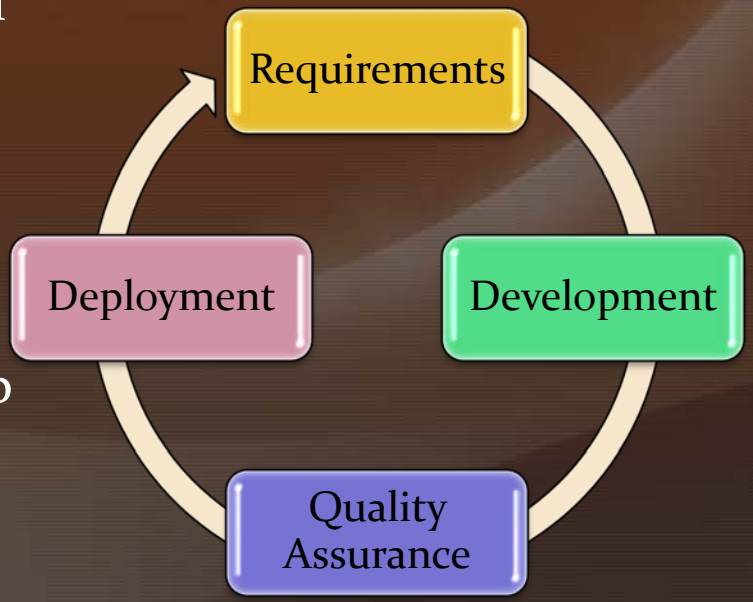
- Pros
  - Very fast turn around on feature requests
  - Little methodology overhead
  - Can accommodate changing designs and priorities
    - End user involvement
  - Short turnaround on investment
  - Consistent integration schedule
    - Lowers the risk of large scale integration issues



Methodologies

RAD (Rapid application development)

- Cons
  - Needs modularized code bases to work well
  - Can't handle complex long term tasks well
  - Harder to budget and manage
    - No predefined end
    - Budgets tend to end up higher due to the addition need to development tools
  - Can build *technical debt* in codebase quickly



Methodologies

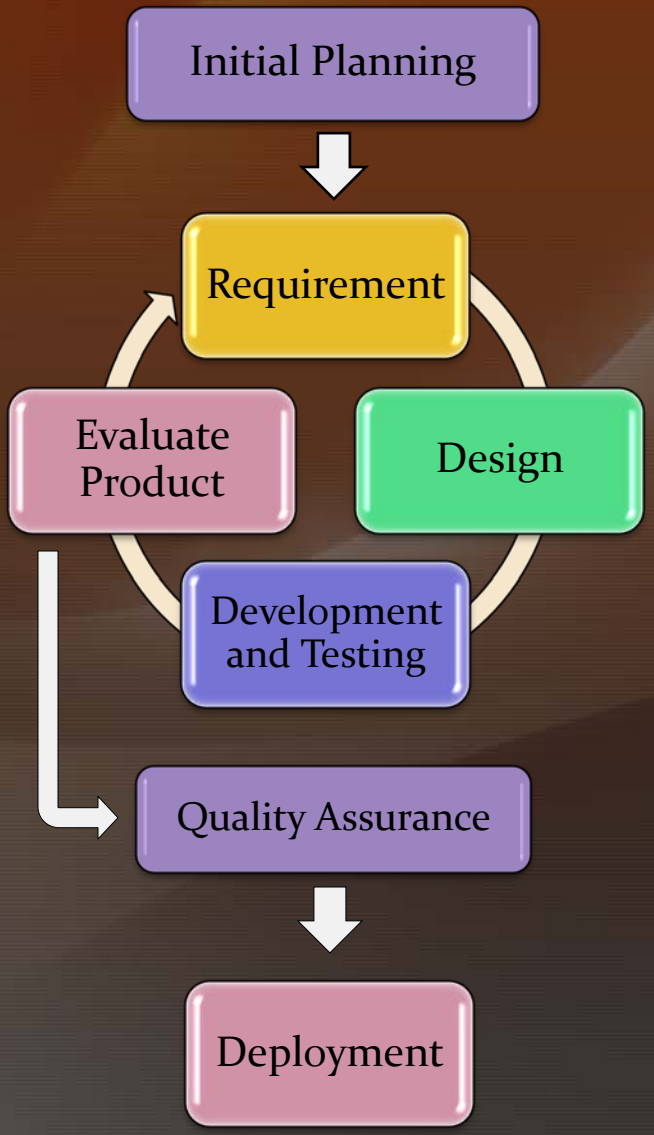
RAD (Rapid application development)

- 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.

Agile/Iterative

Methodologies

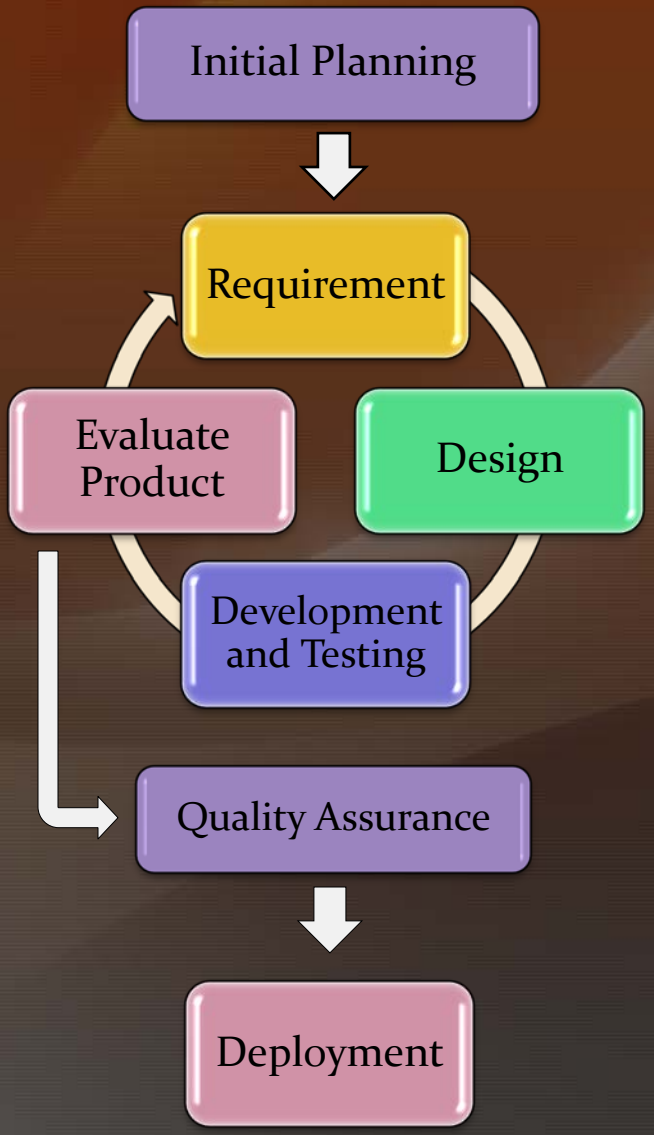
- A cycling development pattern that builds toward a larger end goal
- Breaks the project into incremental builds
- High level design agreed upon up front
- Each iteration intends to be capable of release
- Emphasis on
  - Self organization
  - Responding to feedback
  - Maintaining working software throughout development



Agile/Iterative

Methodologies

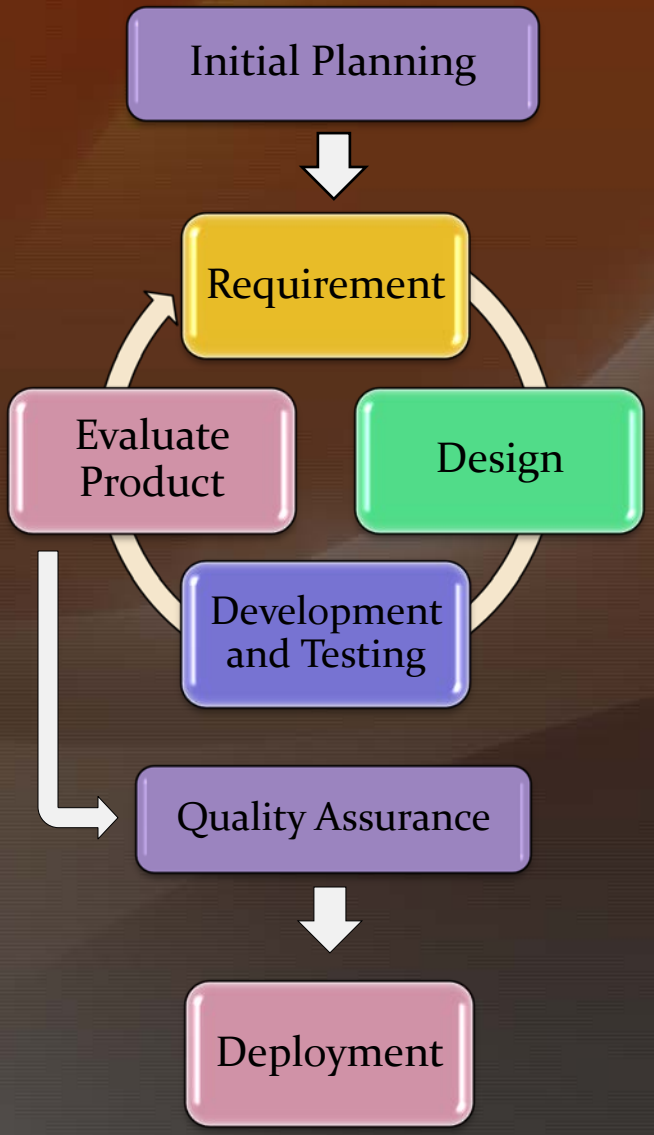
- Pros
  - Flexible
    - Can handle most project types
  - Built to respond to unexpected changes
    - Less cost involved in redesigns
  - Working build achieved early in the process
  - Feedback is gathered quickly and consistently
  - Development and testing happen at the same time



Agile/Iterative

Methodologies

- Cons
  - Significant methodology overhead
    - Requires a leader to maintain agile practices
    - Teams take several iterations to learn
  - Needs comprehensive tests
  - Can lead to scope creep
  - Can have problems managing complex dependencies
  - Can build technical debt in codebase quickly



SCRUM

Main points

- Scrum is an agile/iterative process
  - Each iteration of planning, implementing, and review is referred to as a sprint
  - At the end of each sprint teams have a marketable product (MVP)

Scrum

USER STORIES

User Stories

What is it?

- How work is organized in scrum
  - Short one or two sentence thoughts on how to focus the product
  - Individual tasks that can be shown to be completed just by using the product

User story format

- Who wants it done?
  - What person is demanding it out of the product
  - What person is waiting for it to be developed so they can start with their own work
- What do they want done?

"As a user, I want a ninja enemy "

What are they for?

- The format helps focus the development toward the priorities
- "As a level designer, I need..."
  - Dependency that is needed for him to get his job done
- "As a player of COD, I expect..."
  - Target audience that wants the feature
- "As a user, I want..."
  - Something all users want

What are they for?

- The format helps with plain English communication between disciplines
  - Everyone, across disciplines, should be able to
    - Request functionality
    - Track and quantify progress
    - Confirm completeness

What are they for?

User Stories

- 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

DEFINING COMPLETE

Defining complete

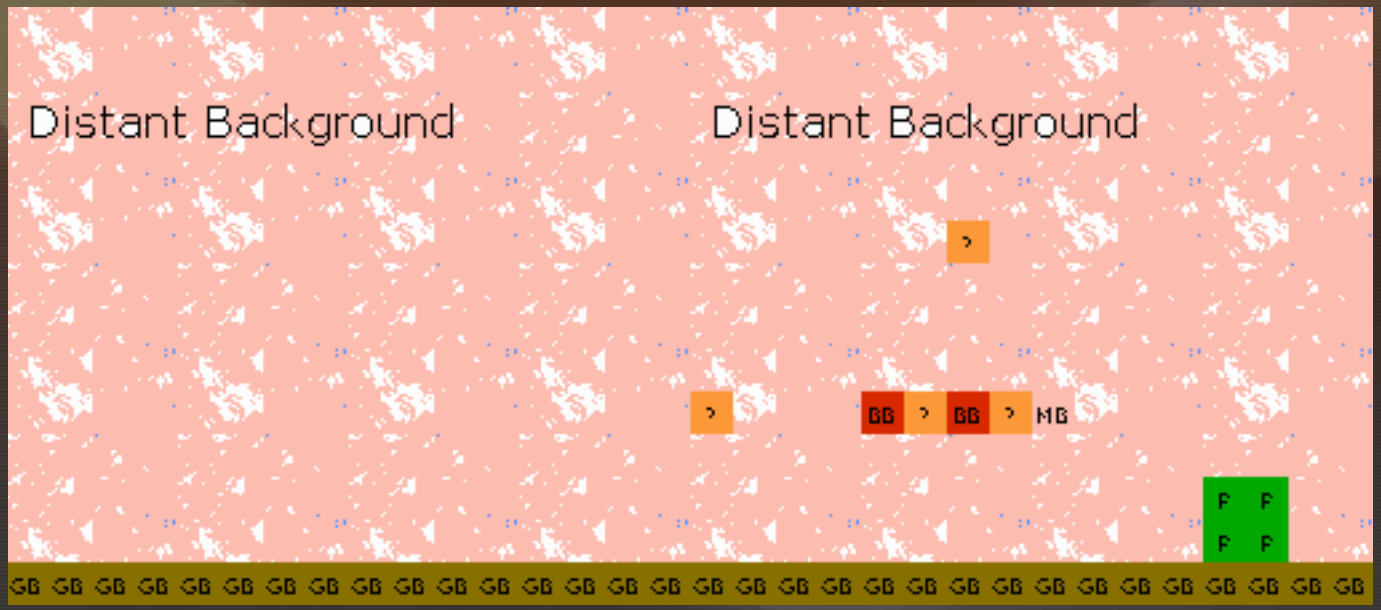
Defining complete

- One of the most common issues with agile development is defining “done”
- Example: As a user I want to be able to play the first level of the game.
  (This is left ambiguous on purpose)

Defining complete

Defining complete

- Proof of concept functionality
  - Having one instance of the functionality in the game to demonstrate



Defining complete

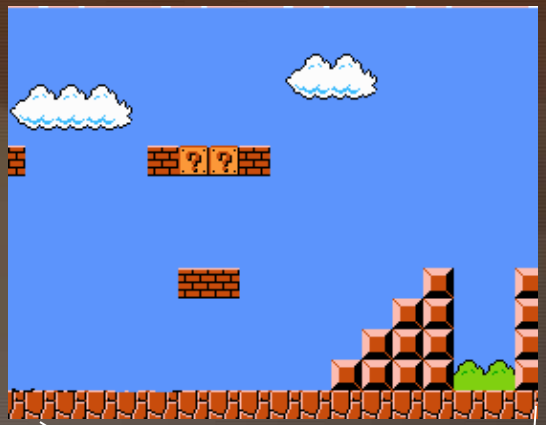
Defining complete

- Placeholder
  - Majority of the functionality is integrated throughout the code, though not complete
  - Using placeholder art

Defining complete

Defining complete

- Fully functional
  - Full functionality as described by the design is in the game
  - No placeholder art is used



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

Test cases

User Stories

- “As a user, I want a ninja enemy “

  Test cases:

  - Can a ninja be created?
  - Can the ninja be killed?
  - Does the ninja use AI states?
  - Can the ninja patrol?
  - Does the ninja animate through all of its states?
  - Does the ninja play SFX for its actions?
  - 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?
  - …

Task lists

- List out everything that has to exist to get all test cases to be true

- Help understand the scope of the feature
- Help recognize the difficulty of implementing the story
- Discover dependencies

Task lists

User Stories

"As a user, I want a ninja enemy for the shadowy forest levels"

Tasks:

- Game Objects
- Ninja game object
  - Creation
  - Death
- Player
- Ninja AI states to set objects to
  - Patrol state
  - Attack state
  - Flee state
- Ninja animations
  - Walk
  - Attack
  - Vanish
- Ninja sound effects
  - Attack sound
  - Damage sound
  - "Whiff" sound

Dependencies?

Task lists

User Stories

"As a user, I want a ninja enemy for the shadowy forest levels"

Tasks:

- Game Objects [D]
- Ninja game object
  - Creation
  - Death
- Player [D]
- Ninja AI states to set objects to
  - Patrol state
  - Attack state
  - Flee state
- Ninja animations
  - Walk
  - Attack
  - Vanish
- Ninja sound effects
  - Attack sound
  - Damage sound
  - "Whiff" sound

Dependencies?

Task lists

- Watch out for userstories with lots of test cases or dependencies
  - Stories with lots of them can't be completed until late in the process
  - We need to ensure that those dependencies have their own stories elsewhere
  - The problem exaggerates if anything is also dependent on this story

Most Common problems

- Not knowing what you want before writing the userstory
  - If the full design hasn't been decided upon yet. It must be now.
- Using ambiguous terminology
  - -ly words

Most Common problems

- Overly-encompassing userstories
  - I want all the enemies/bosses – bad
- Under-encompassing userstories
  - I want level 5 boss to move left - bad
- Each userstory should be enough work to be 1 integration and commit

Most Common problems

User Stories

- To many or to few test cases
  - 10+ test cases
    - Userstory most likely need to be broken down into parts
  - 1 test case
    - Userstory is probably only a test of a larger userstory
    - Test cases don't cover all testable aspects of the feature

Additional Resources

Additional

- 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

Class Activity

- User Story Writing
  - We have a backlog
  - Let's expand the work in there into userstories

Scrum Process

Scrum Process

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
Main Menu
Object Collision
Cheat codes
Options menu
Player
Player Jump
Big Head Mode

Scrum Process

Product backlog

- The back log is prioritized according to overall importance to the product, stake holders, and dependencies
- This ensures
  - Only the highest priority things get worked on
  - The things unnecessary to the game keep getting pushed down

	Product Backlog
A	Main Menu
A	Object Collision
..C	Cheat codes
.B	Options menu
A	Player
A	Player Jump
...D	Big Head Mode

Scrum Process

Sprint backlog

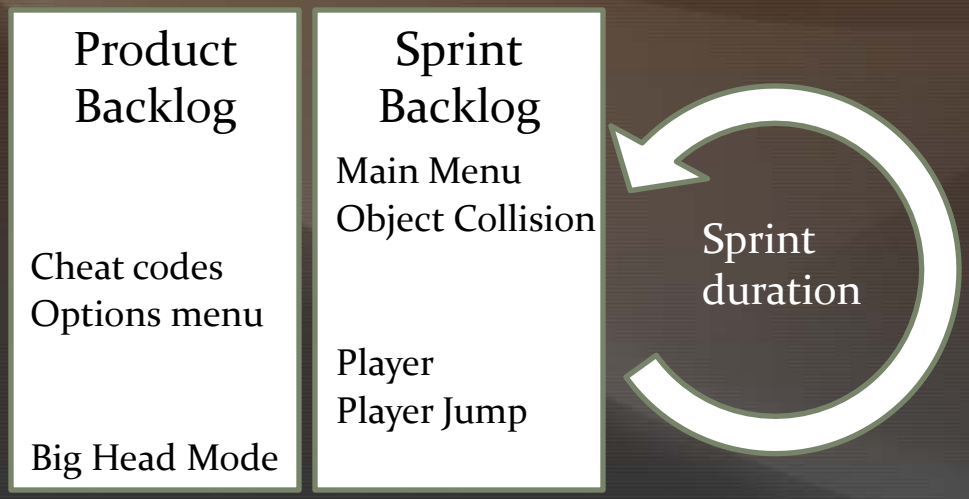
- Each iteration a sprint backlog is filled with the tasks that will be completed in the upcoming sprint
- Tasks are selected based on priority to the project and the upcoming sprint goal.

	Product Backlog	Sprint Backlog
A	Main Menu	Main Menu
A	Object Collision	Object Collision
..C	Cheat codes	
.B	Options menu	
A	Player	Player
A	Player Jump	Player Jump
...D	Big Head Mode	

Scrum Process

Sprint backlog

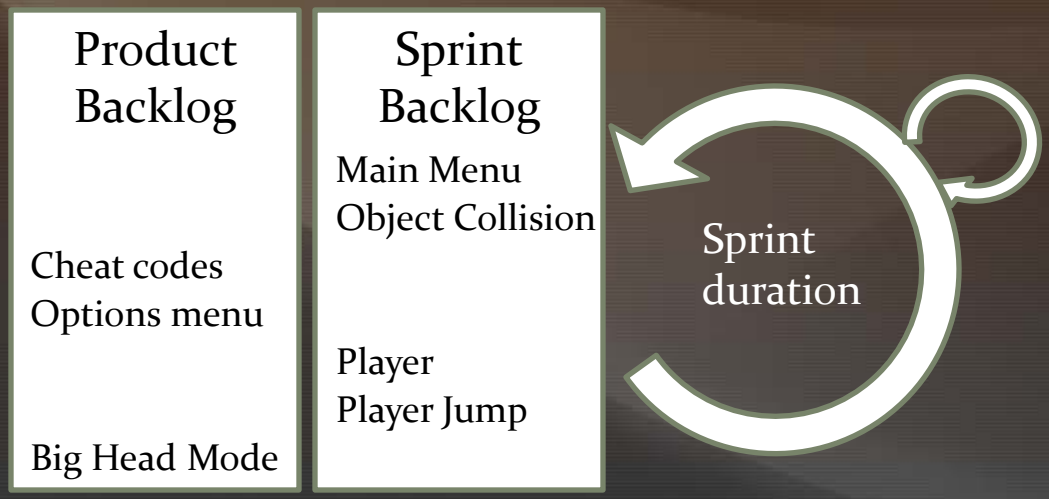
- Once the sprint has started a commitment has been made to complete the tasks in the sprint backlog



Scrum Process

Work day in scrum

- An iteration occurs each day within scrum



Work day in scrum

Scrum Process

- Teams meet every day for a scrum "stand up" meeting
  - Maintain transparency
  - Hold each other accoun
  - Set up help when neede

Daily scrum meeting

Sprint duration

Work day in scrum

Scrum Process

- Key points of scrum “stand up” meetings
  - The meeting should be the start of our working day

Work day in scrum

Scrum Process

- Key points of scrum “stand up” meetings
  - The daily meeting should answer the following for each team member
    - What did you do?
    - What are you about to do?
    - What stands in your way?

Work day in scrum

Scrum Process

- Key Points of scrum “stand up” meetings
  - Everyone is engaged in the meeting
    - Laptops down
    - Preferably in front of the scrum board
      - (we will get into that later)

Work day in scrum

Scrum Process

- Key Points of scrum “stand up” meetings
  - Not just the status reports for management
  - Commitments in front of peers

Daily scrum meeting

Sprint duration

Work day in scrum

Scrum Process

- Key Points of scrum “stand up” meetings
  - Maximum of 15 minutes.

Work day in scrum

Scrum Process

- The team breaks apart to work on assigned tasks and help the peers when needed

Daily scrum meeting

Sprint duration

Work day in scrum

Scrum Process

- Continue to work until the end of the agreed upon work day
  - Integrate the work
  - Update task tracking

Daily scrum meeting

Sprint duration

Work day in scrum

- This practice repeats each day until the end of the sprint

Scrum Process

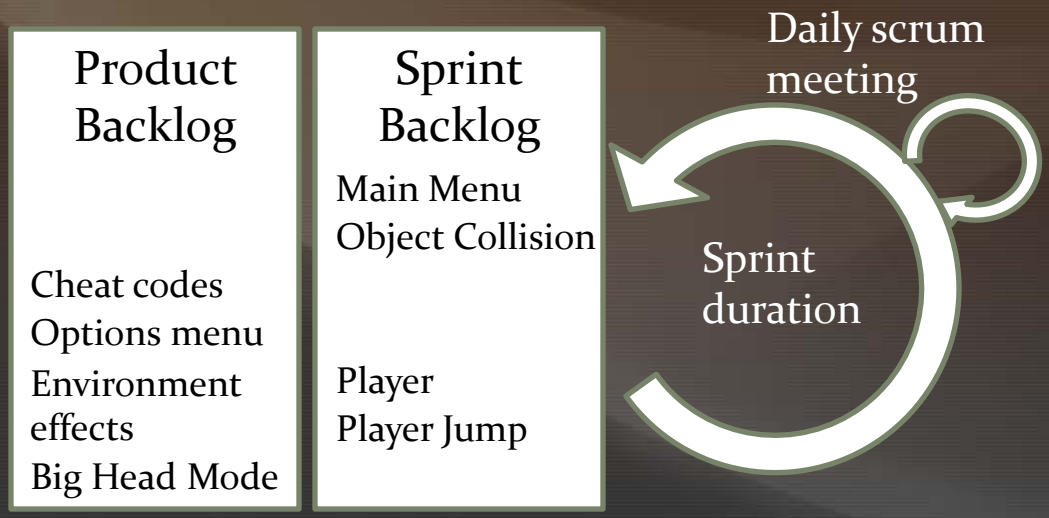
Daily scrum meeting

Sprint duration

Scrum Process

Creating a product

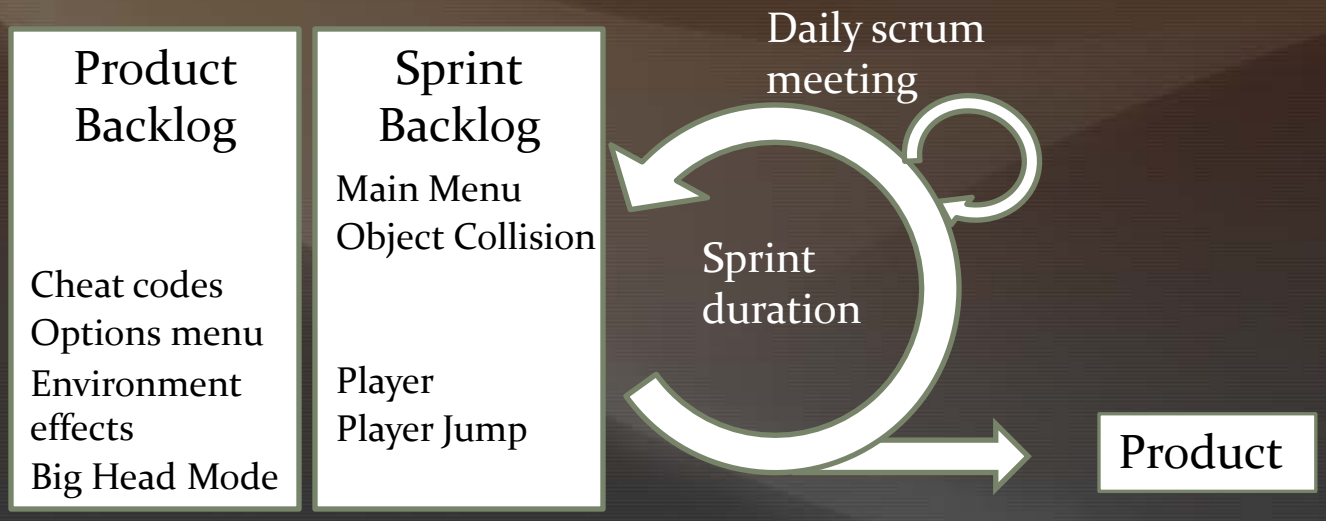
- During the sprint, things are added to the product backlog if
  - Discovered to make the product better
  - Added from outside influences
  - Changes in product expectation from client



Creating a product

Scrum Process

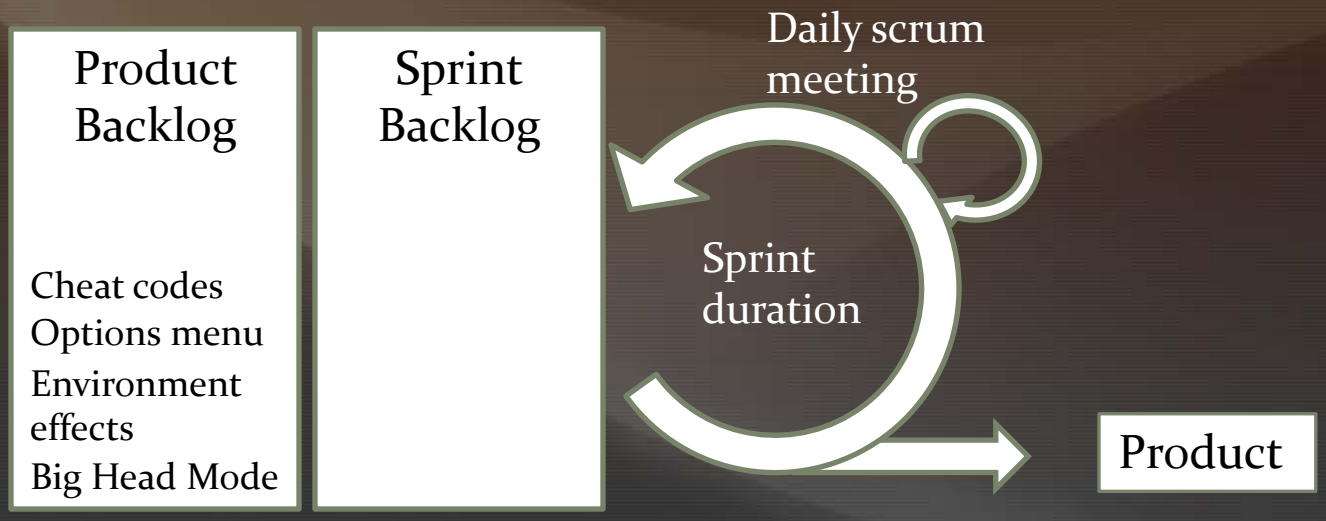
- At the end of each sprint the product is delivered in marketable state



Scrum Process

Creating a product

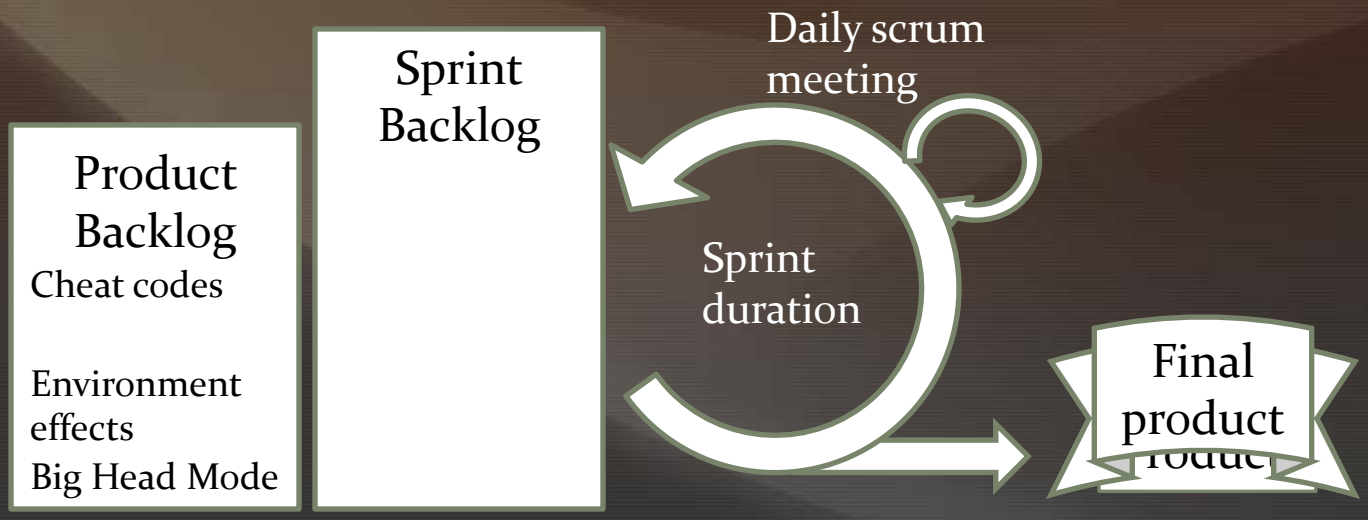
- Over the course of multiple sprints
  - The product backlog get smaller
  - The end product gets better and more feature rich



Scrum Process

Creating a product

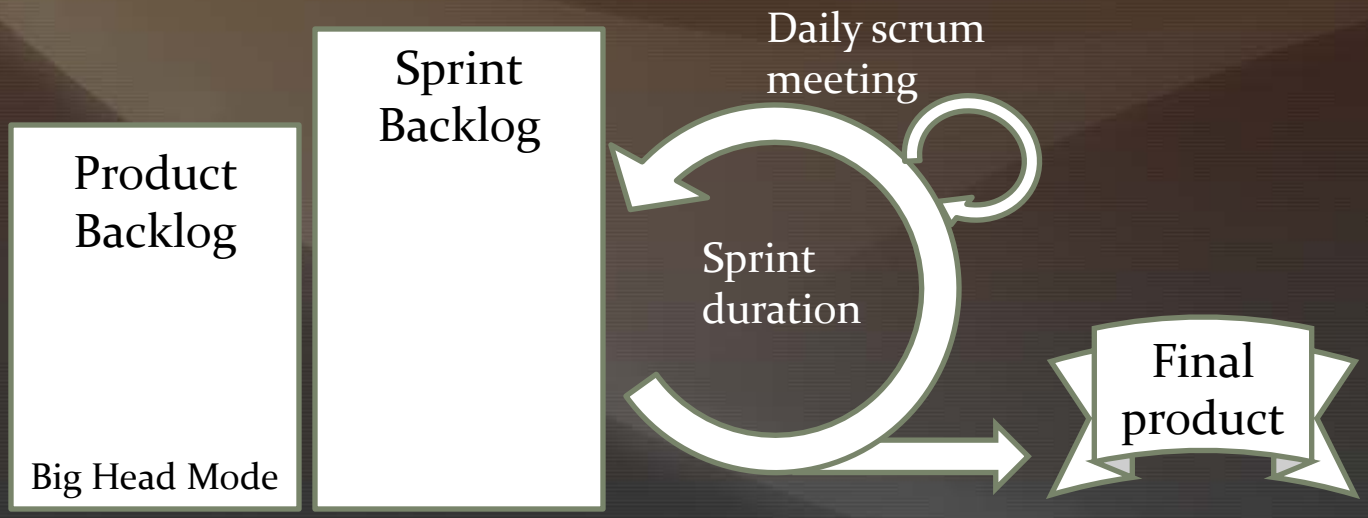
- Eventually this leads to a the productr that will be released



Creating a product

Scrum Process

- Stories may be left in the back log at release
  - They are deemed unnecessary
  - Planned for further updates/patches
  - Put off for a sequel



Task Board

How we keep things straight

Task board

- A task board is used to maintain a constant vision of the progress of the specific user stories and their tasks

- Traditionally a task board is made up of a cork board and index cards
- There are lots of digital solutions to this as well

User story card
(Formatting can vary widely)

User Story Card

Owner	Story #	Estimated	Actual
	1	2	

User Story

As a user I want to have the ability to pause the game [TRC requirement]

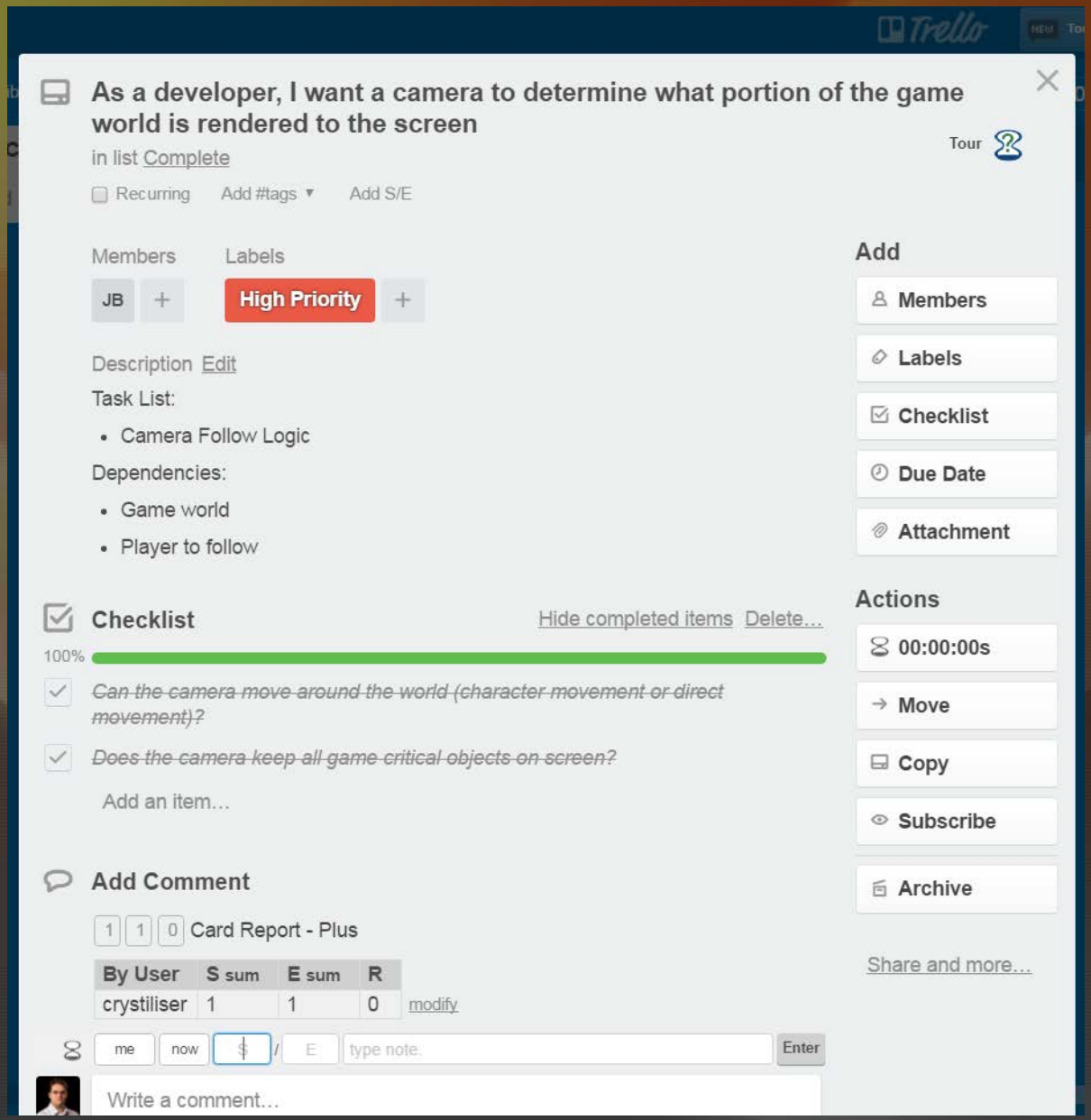
Test Cases

[ ] Can the player pause the game at any point during gameplay?
[ ] Does the game clearly indicate to the user that it is paused?
[ ] Does all of the gameplay stop while the game is paused?
[ ] Does gameplay resume correctly when the game is un-paused?
[ ] Does pressing the escape key while in game pause the game or bring up an in game pause menu?

Peer review sign off

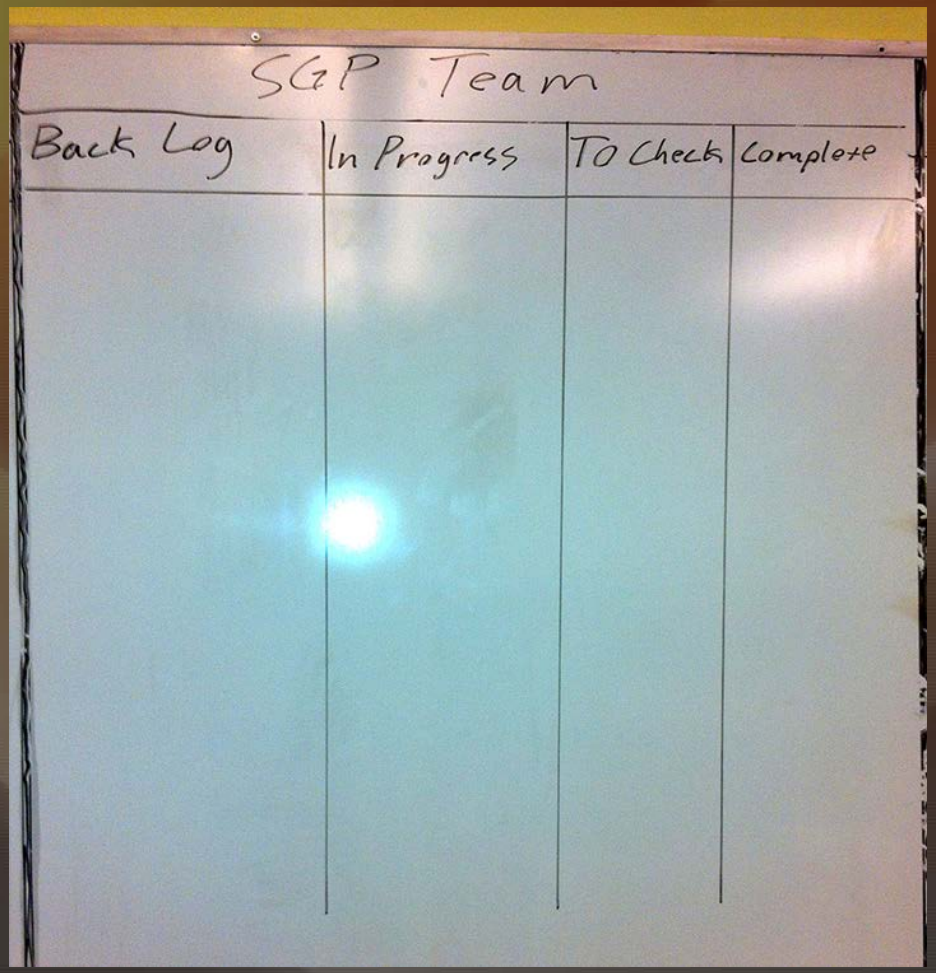
Our Cards on Trello

User Story Card



Task board

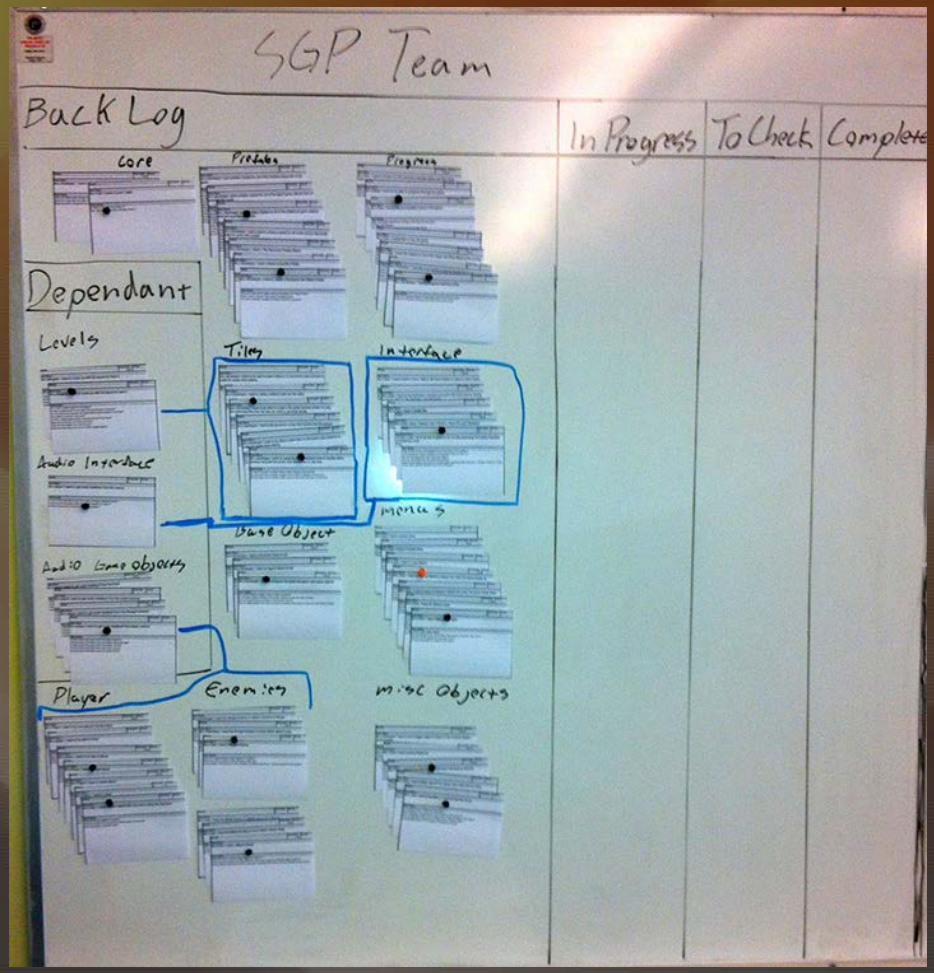
Example task board



Task board

No team is the same, most teams customize the board to suit their needs.

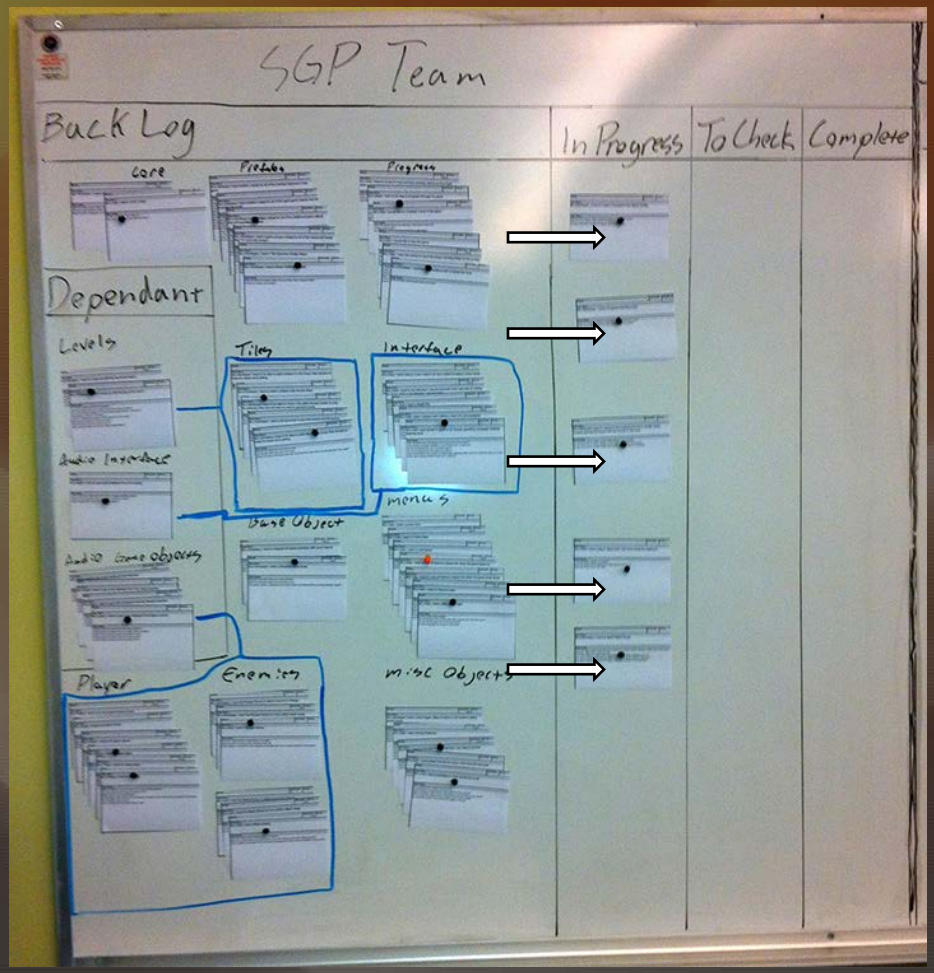
Example: Adding a dependent category to see more clearly if a story can be worked on at the moment



Task board

Rules for the task board:

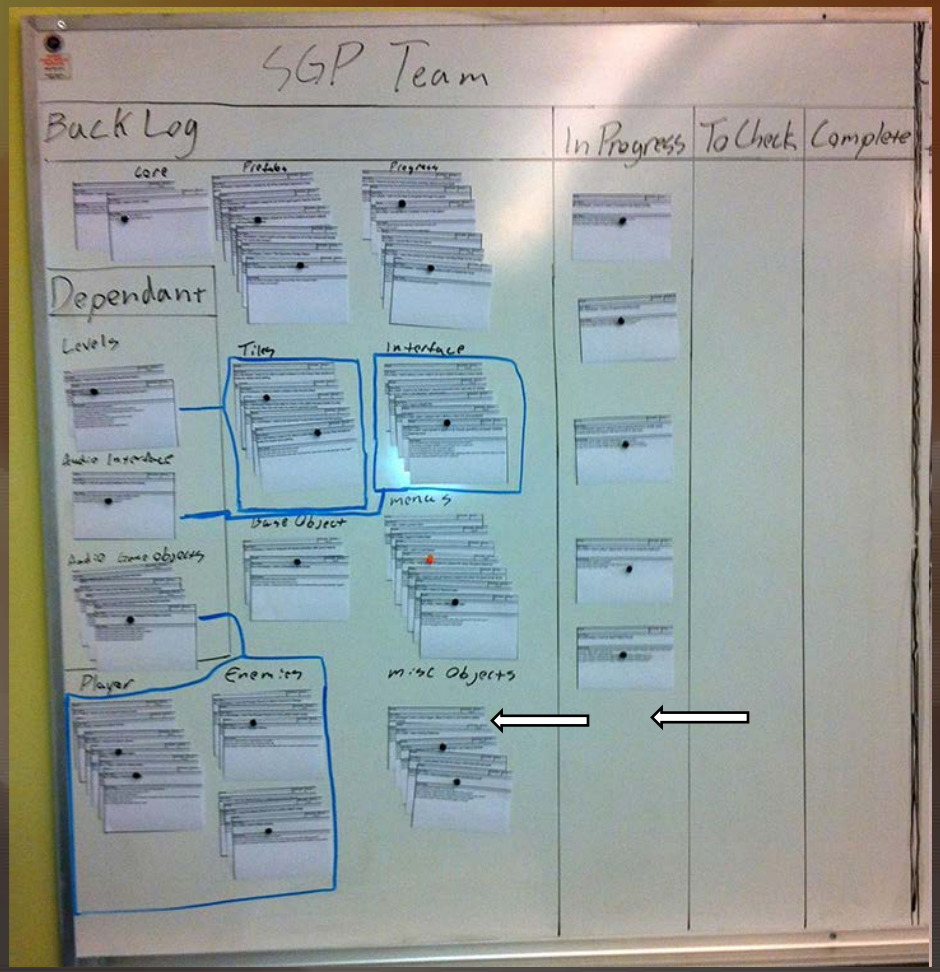
User stories should only move forward



Task board

Rules for the task board:

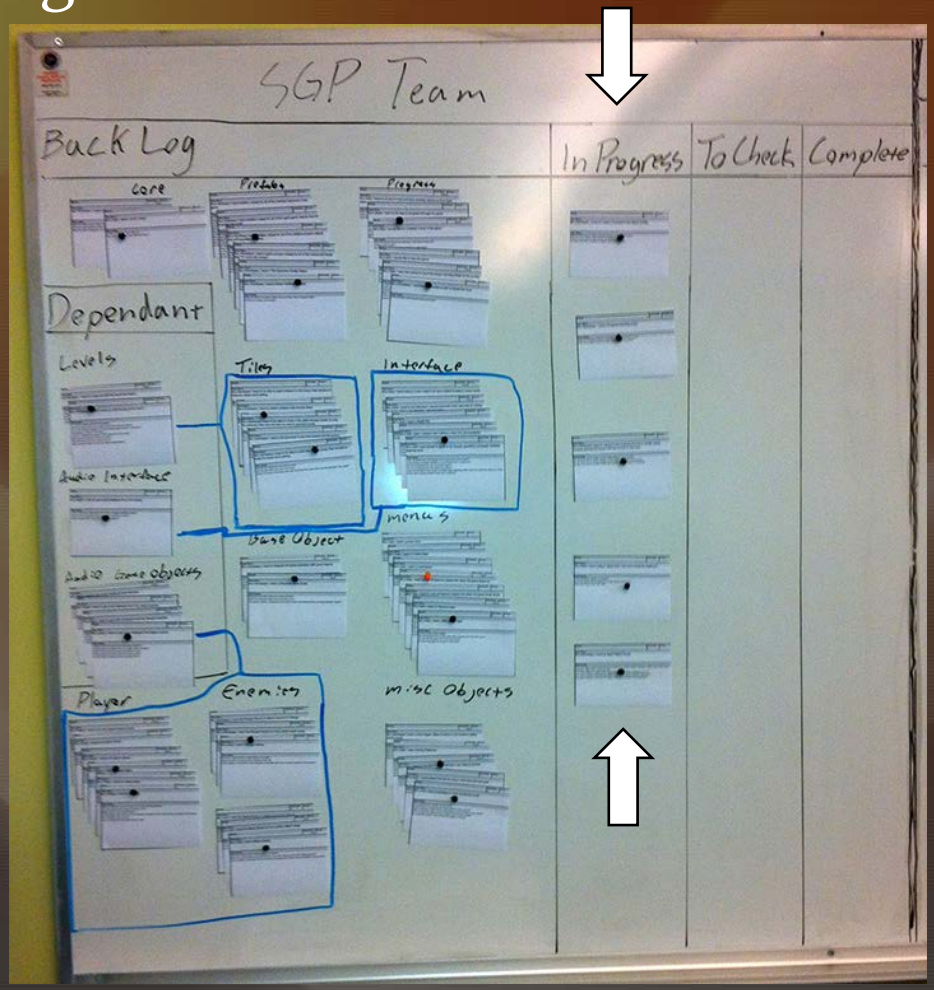
If a user stories ever moves backward there was a failure in communication somewhere



Task board

Rules for the task board:

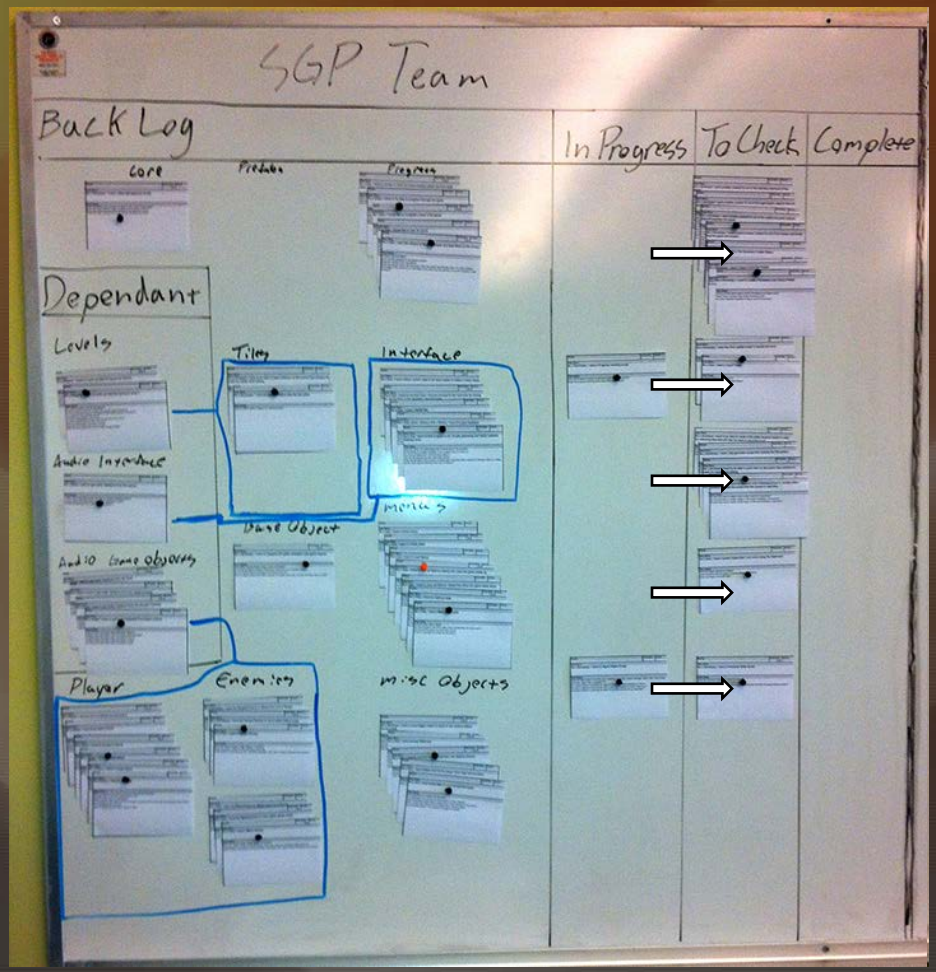
Each developer only has one user story in progress at a time



Task board

Rules for the task board:

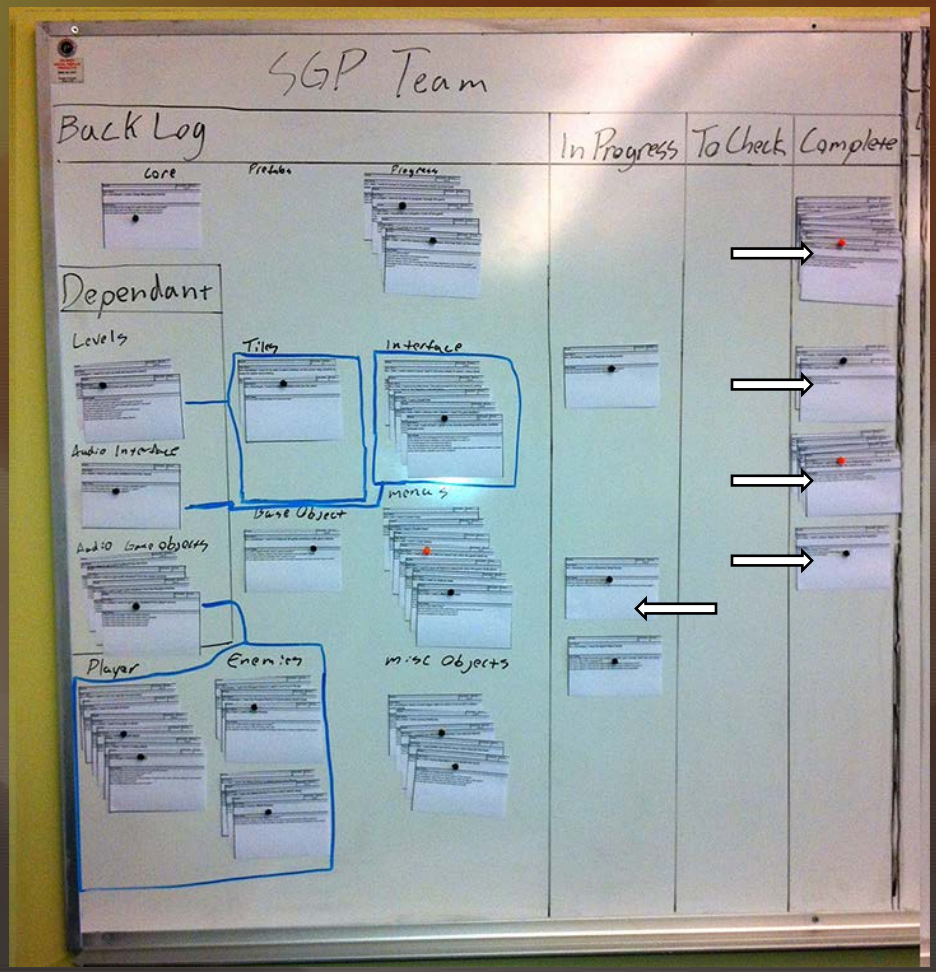
Things only move into “to check” when everything about the user story is thought to be complete



Task board

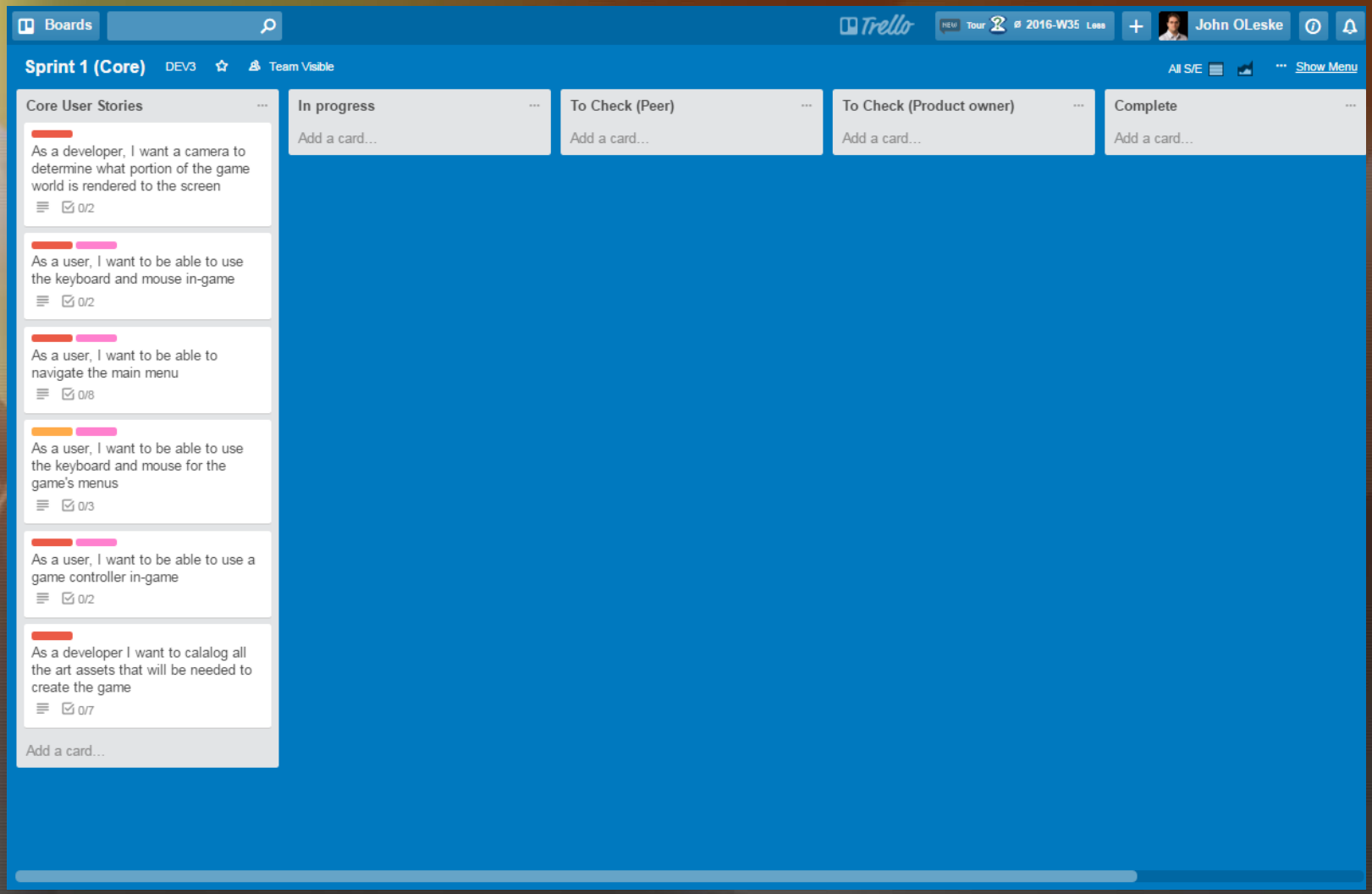
Rules for the task board:

- Everyone must agree that the story is complete



Our Board on trello

Task board



Burn down chart

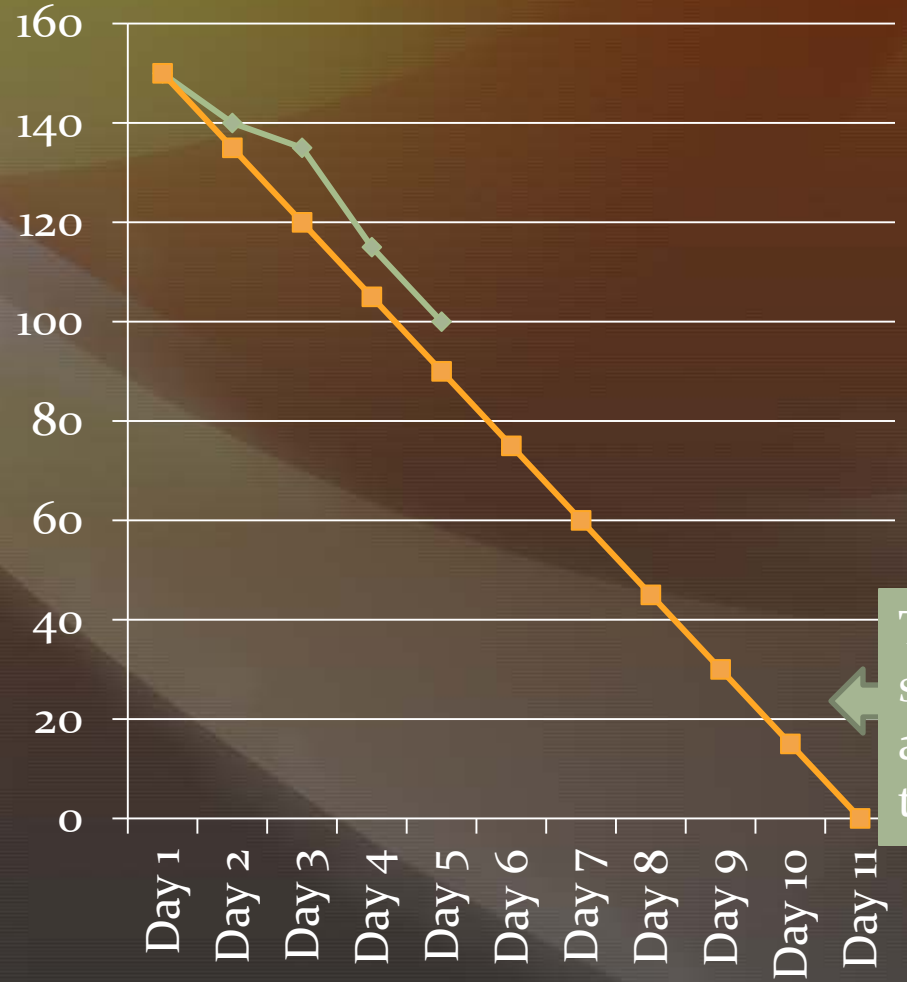
Burn down chart

How we keep things straight

- A burn down chart is used to maintain a constant vision of the overall progress of the sprint
  - Are we ahead?
  - Are we behind?
  - Are we getting enough done day to day?

Parts of the chart

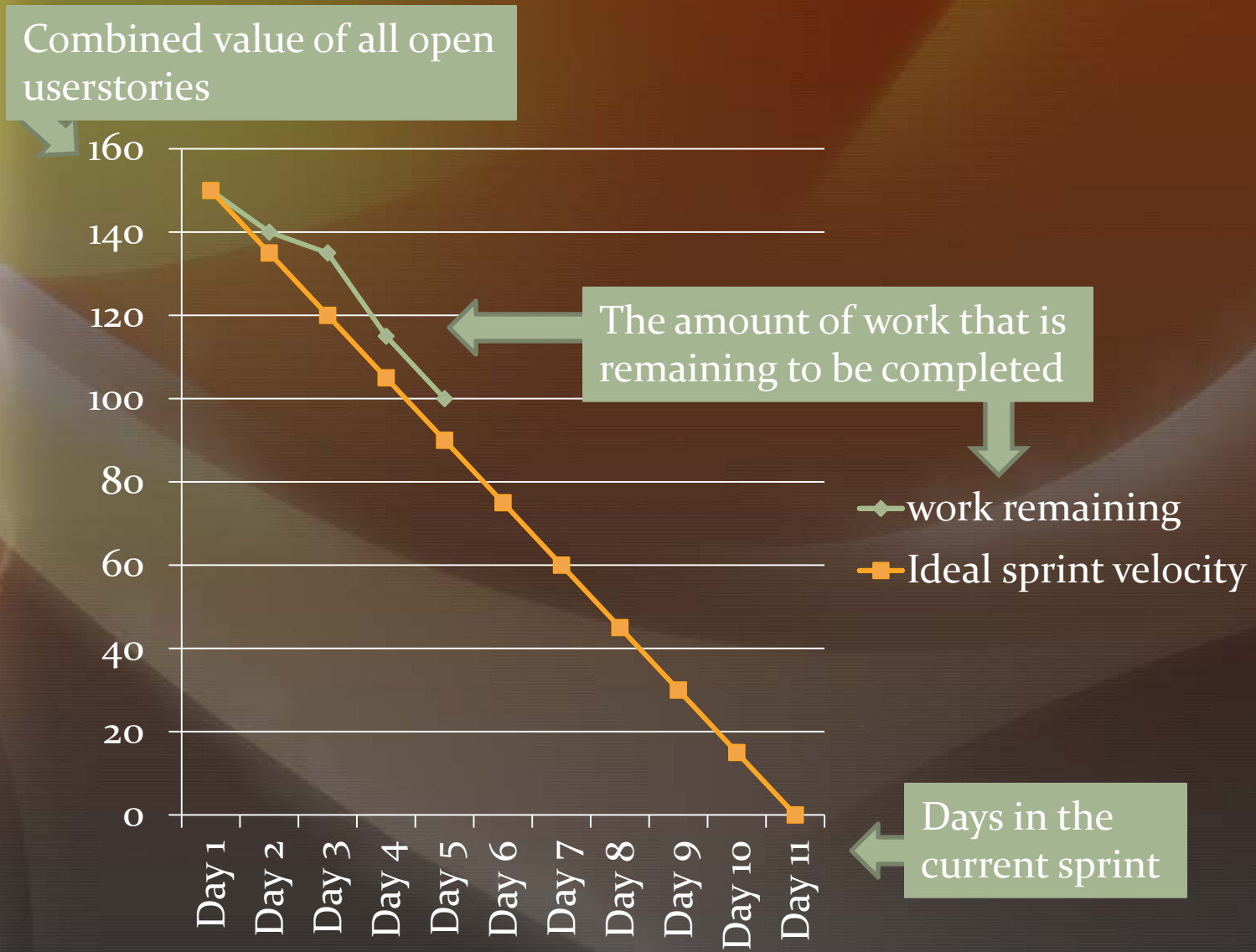
Burn down chart



The relative speed things should be completed to get all work done by the end of the sprint

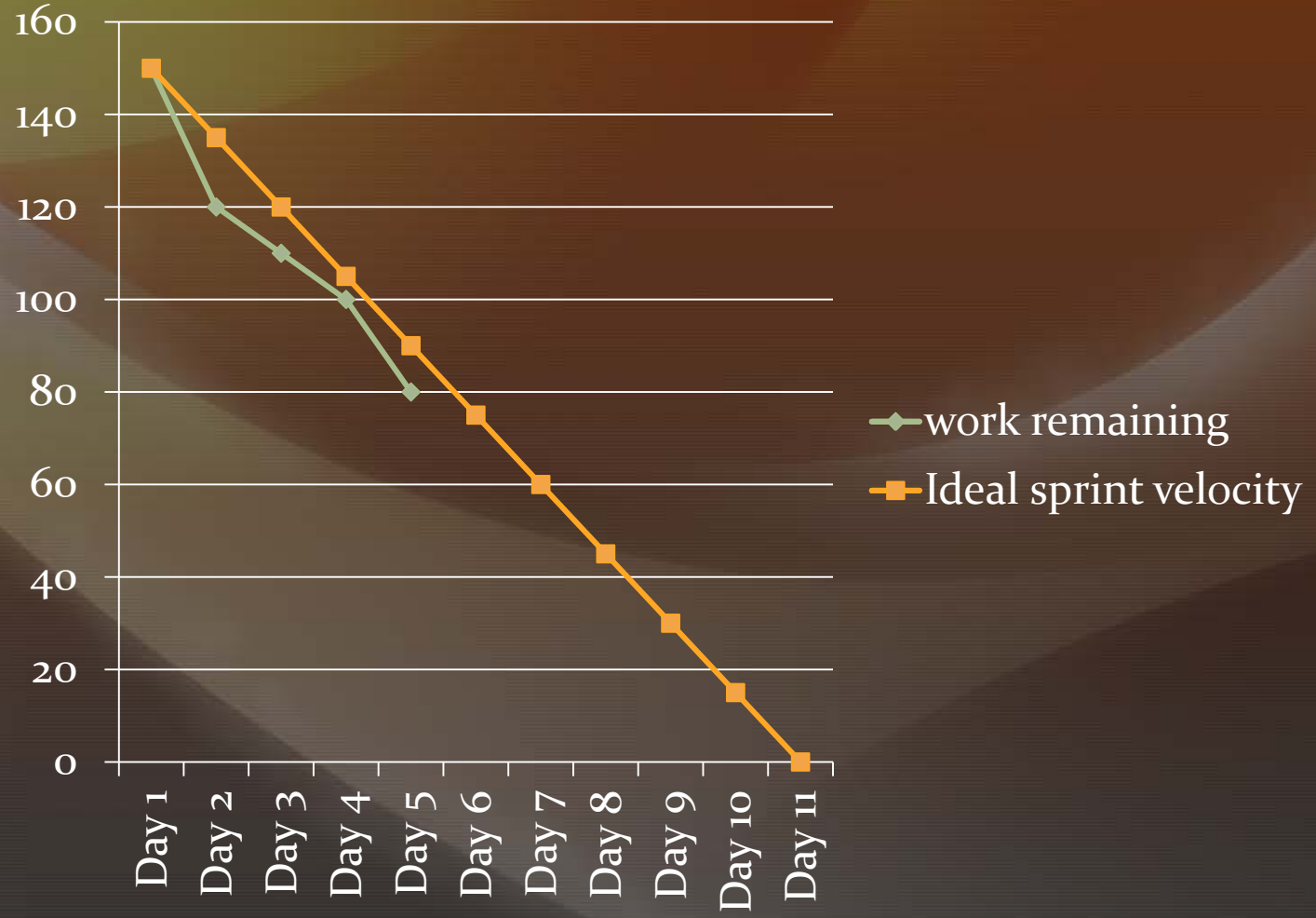
Parts of the chart

Burn down chart



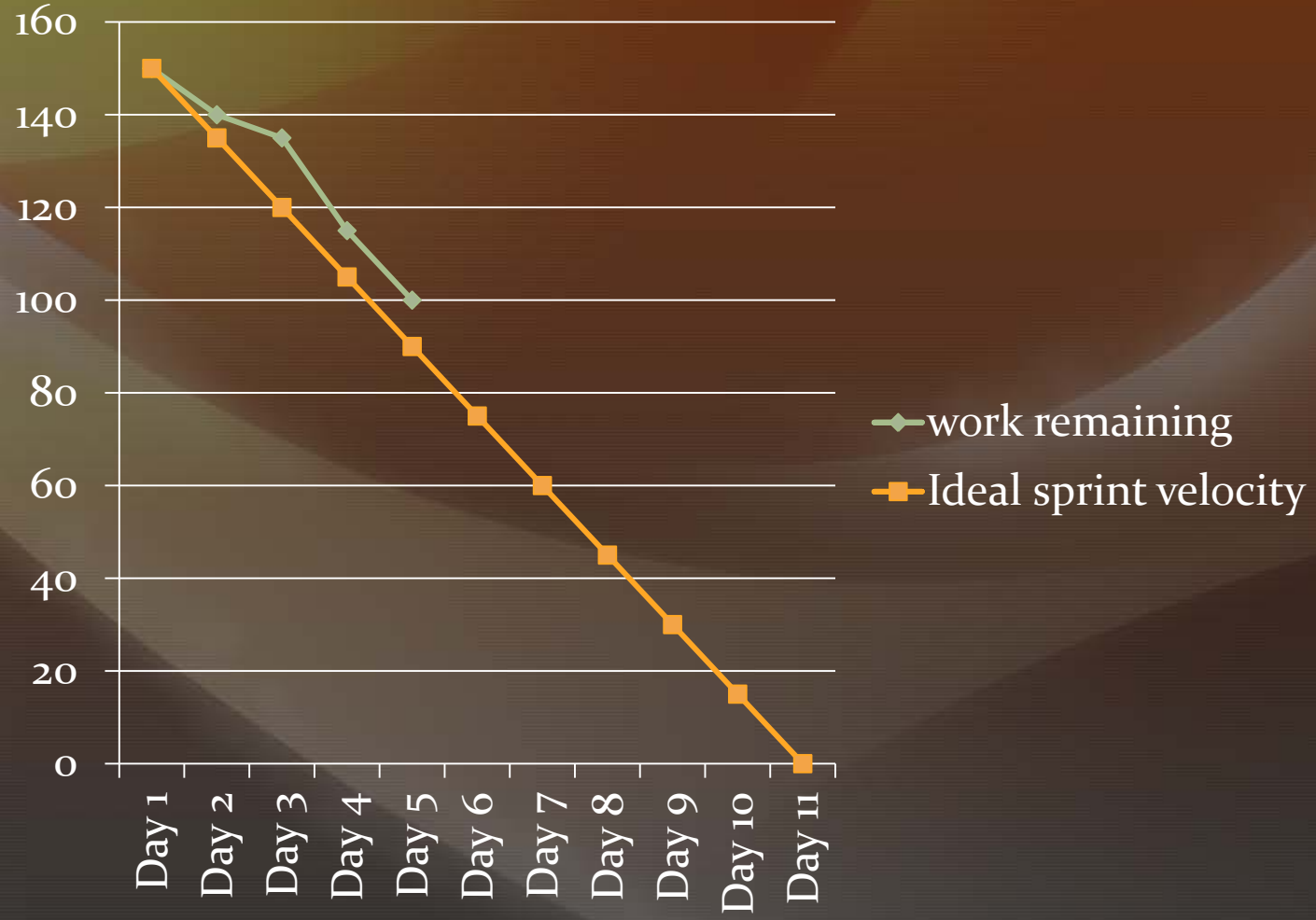
Example of a chart that is ahead of schedule

Burn down chart



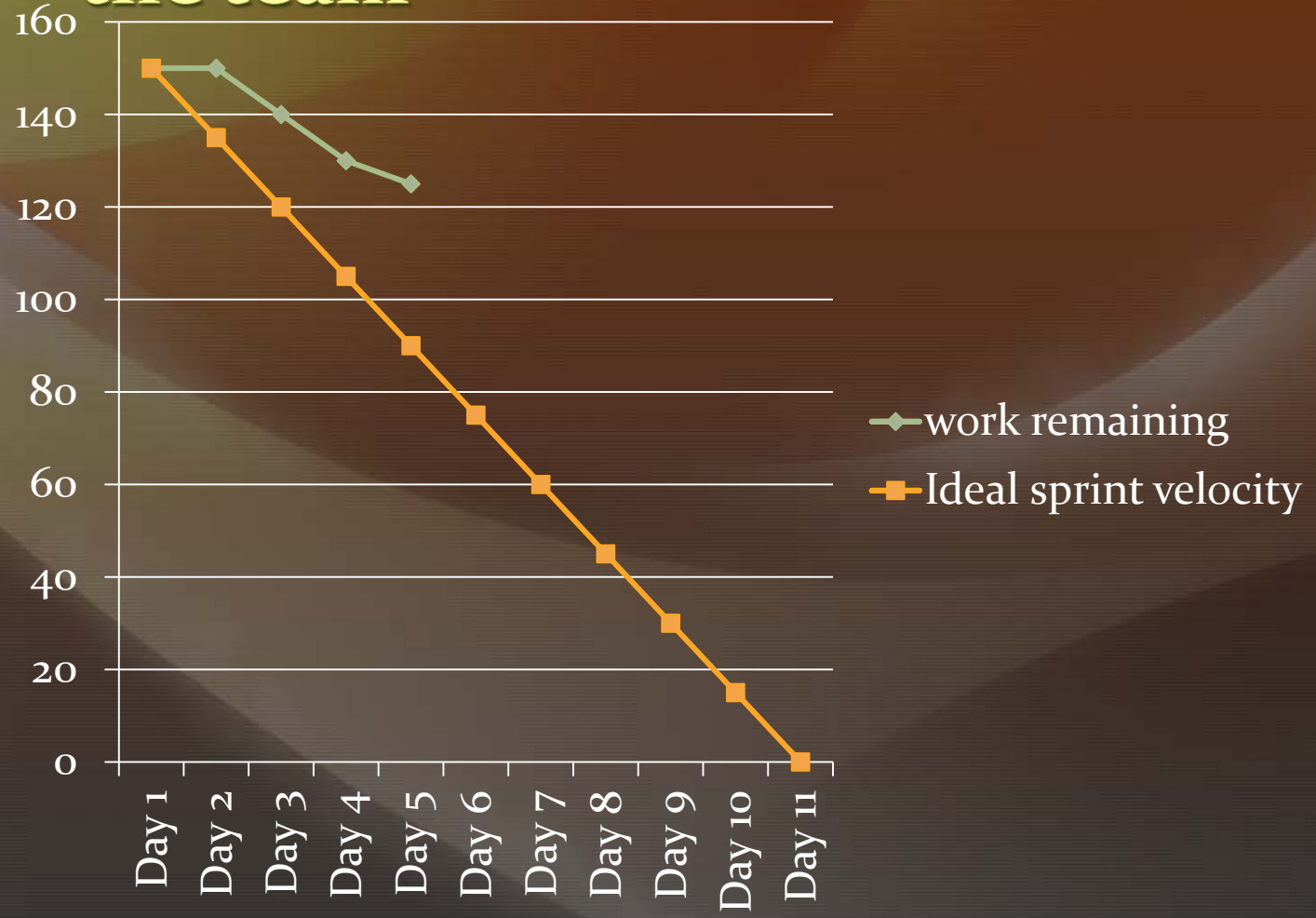
Example of a chart that is nearly on target

Burn down chart



Burn down chart

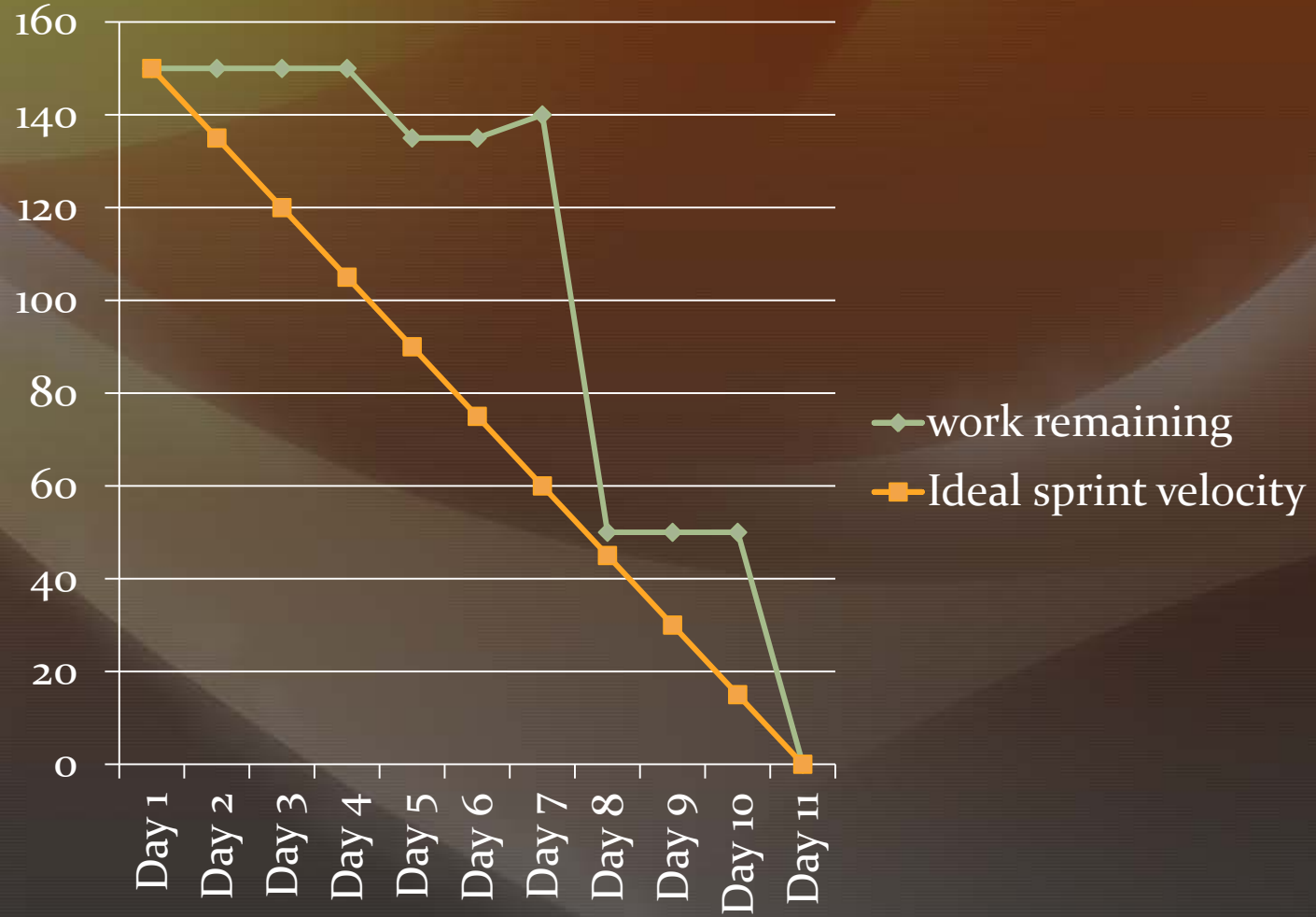
Example of a chart that is showing the sprint is getting away from the team



Example unfortunate events

What could cause a chart like this?

Burn down chart



Visibility

Only as good as you make them

- All of these tools are only as useful as you allow them to be
  - Wait to check off user stories causes the boards and charts to lie to you
  - If the information is wrong they aren't useful
    - Won't know when you are behind and need to crunch

Class Activity

- Sprint Tracking with Trello

Trello

Assignments

Pre Pro Assignments

Assignments

- Product backlog
  - 1 feature != 1 userstory
  - Everything we might want to add to the game
  - Average of 50-100 userstories
    - (depending on scope of story)
    - (in addition to the stories we gave you)

Plan

Things that must be done today

- Review design documentation portion of trello
  - Revise documentation for feedback received
- Work on product backlog in Trello
- Greenlight products