CSC 3380 Aymond

Section 1

1/22/2020

Announcement

- Saturday makeup class for National Championship Days
  - We do not need to make up these days
- Project kickoff next class
  - Try to have your team of 6 set by that time
  - Make every effort to not miss the class!!

Team Diversity

- While it is tempting to put together a team of friends, research shows that you will have better success if you don't
- Your goal is to construct a team of diverse individuals with respect to knowledge, experience, and cognitive preferences
  - How Diverse Teams Produce Better Outcomes
    - https://www.forbes.com/sites/sianbeilock/2019/04/04/how-diversity-leads-to-better-outcomes/#4ad116c865ce
  - Why diverse teams make better business decisions
    - https://thenextweb.com/entrepreneur/2019/04/10/why-diverse-teams-make-better-business-decisions/
  - Do diverse teams produce more creative results?
    - https://the-innovation-race.com/media/do-diverse-teams-produce-more-creative-results/
  - The Truth About Diverse Teams
    - https://www.inc.com/greg-satell/science-says-diversity-can-make-your-team-more-productive-but-not-without-effort.html
  - Diverse Teams Feel Less Comfortable — and That's Why They Perform Better
    - https://hbr.org/2016/09/diverse-teams-feel-less-comfortable-and-thats-why-they-perform-better

Structured Programming vs OO Programming

- Structure Programming
  - Focus on logic and process flow
  - Defines operations on data manipulation
  - "Do something to data"
- OO Programming
  - Focus on data abstraction
  - Hides how data manipulation operations are performed
  - "Tell data to do something"

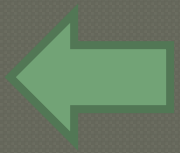
OO Development Activities

- Conceptualization
  - Establish a vision and core system requirements
- OO Analysis and Modeling
  - Build models the system's desired behavior
  - Unified Modeling Language (UML)
- OO Design
  - Create an architecture for implementation
- Implementation
  - Coding, debugging, and unit testing
  - Integration and integration testing
  - Regression and system testing
  - Deployment and deployment testing
- Maintenance
  - Fixing issues
  - Enhance functionality
  - Adapting the system to evolving needs and environments

Jia, Xiaoping. Object Oriented Software Development Using Java: Principles, Patterns, and Frameworks. Addison Wesley, 2003.

The Software Development Process

- Intro to the software development process
- Software Lifecycle Models
- Agile Software Development
- Software Requirements



What is a development process?

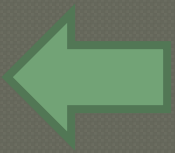
- A way of dividing software development into phases such as:
  - Implementation
  - Design
  - Testing
  - Etc.
- A way of defining the activities of developers

The "natural" development process

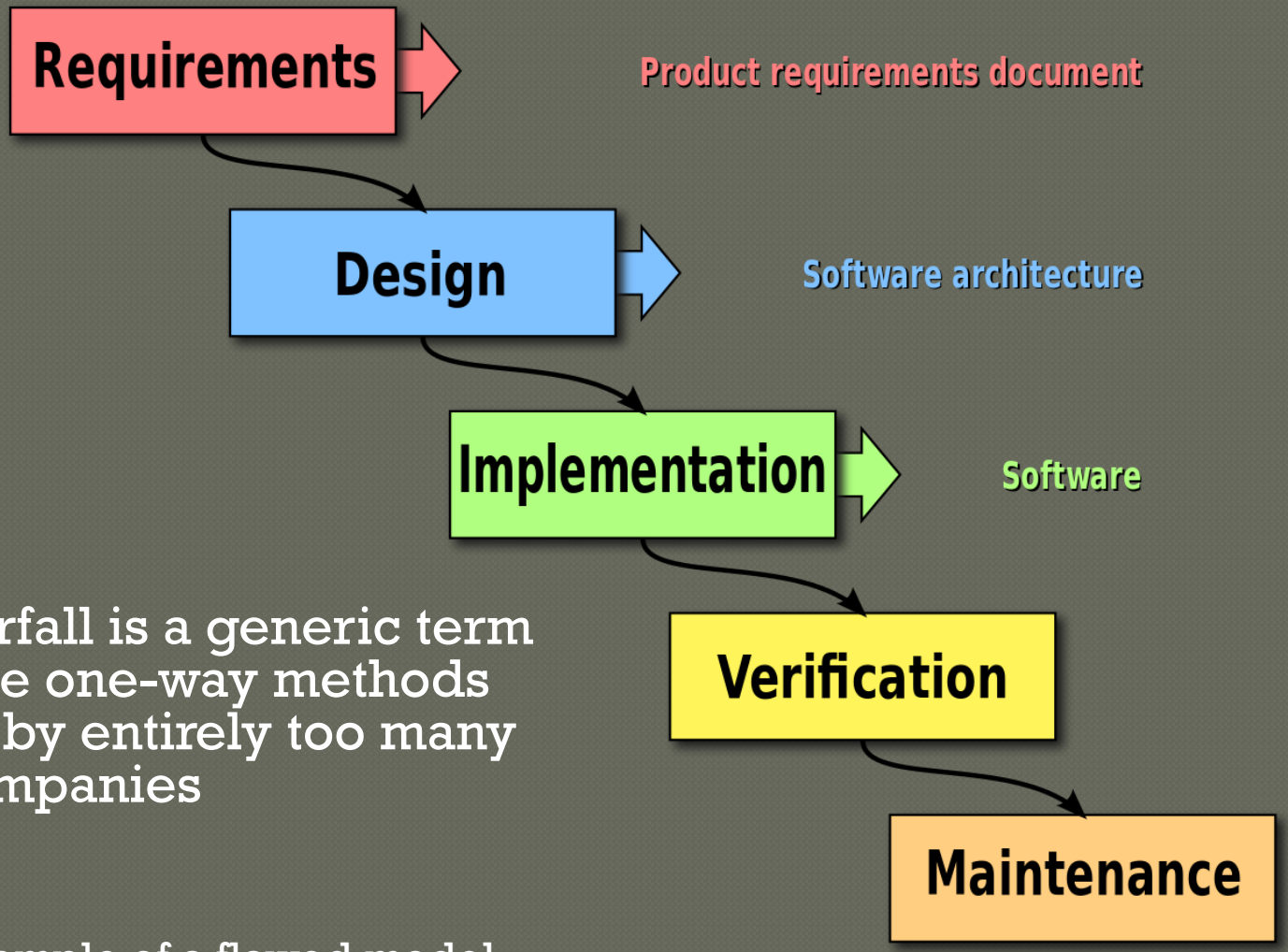
- Get project requirements
- Disappear and start coding immediately
- Abruptly stop coding and start testing
- Emerge from cave to demo project
- Haphazardly fix bugs as they emerge

The Software Development Process

- Intro to the software development process
- Software Lifecycle Models
- Agile Software Development
- Software Requirements



Waterfall: An Engineering Process



Waterfall is a generic term for the one-way methods used by entirely too many of companies

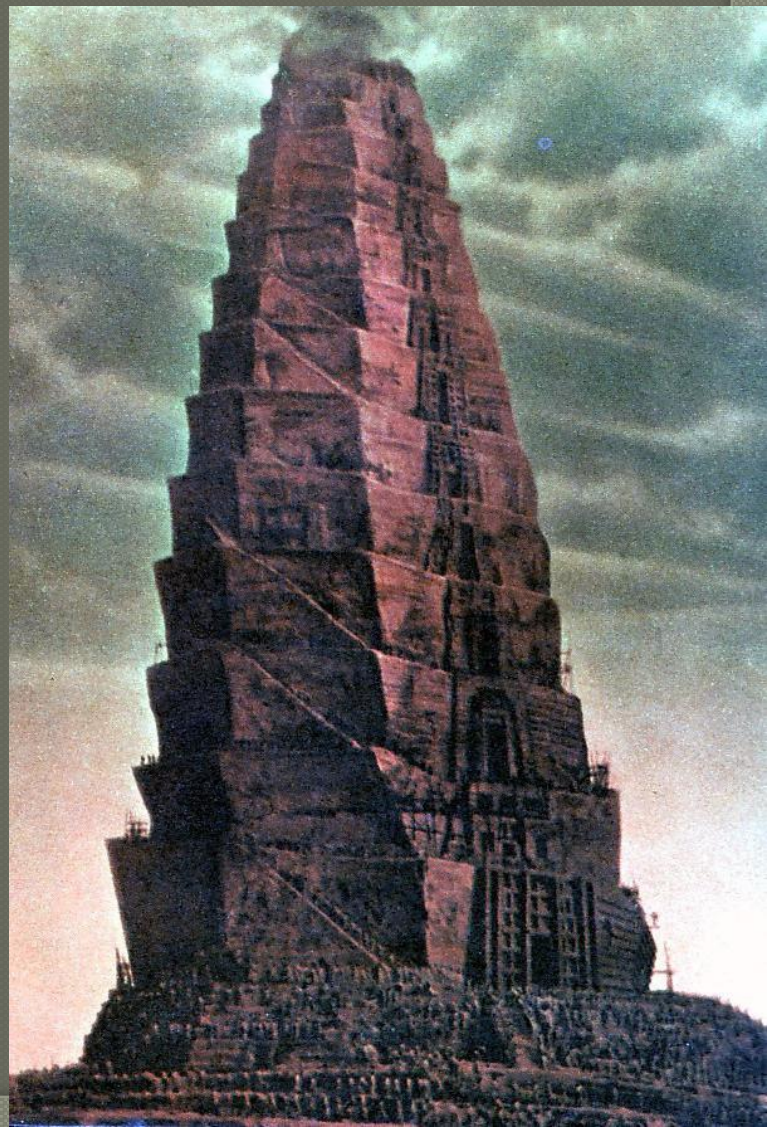
* An example of a flawed model

Informal Motivation

- Simple to understand and manage
- Engineers can specify completely
- Fixing problems in earlier phases is cheap(er)

What's Wrong?

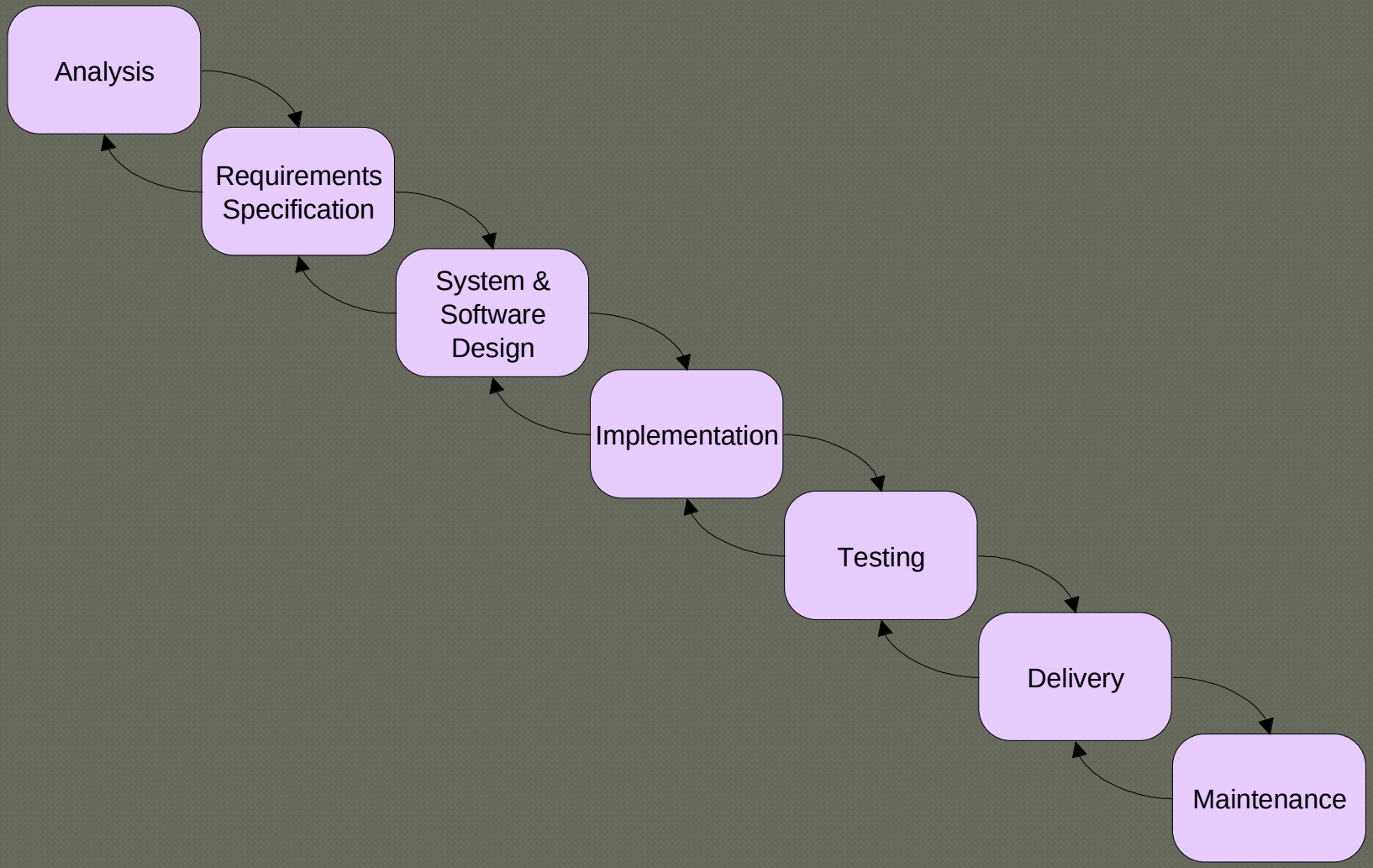
- Software engineering is not like other disciplines
- Requirement specification is never complete (or unambiguous)
- Stakeholders change their minds often
- Nearly infinite complexity in software



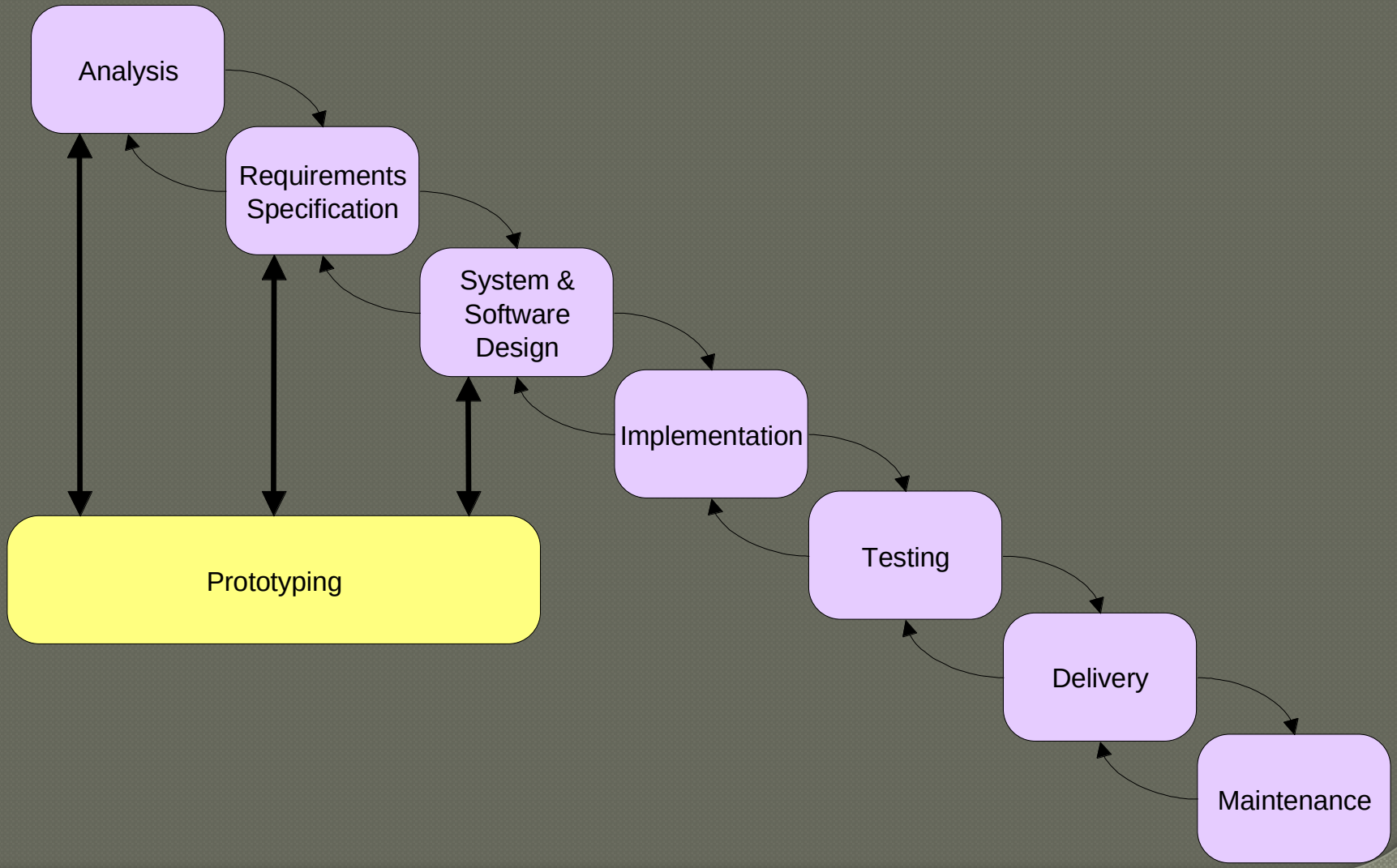
Effects

- Software that is
  - Heavily delayed
  - Significantly over budget
  - Does not meet the need

Alternative Lifecycle: Modified Waterfall Lifecycle

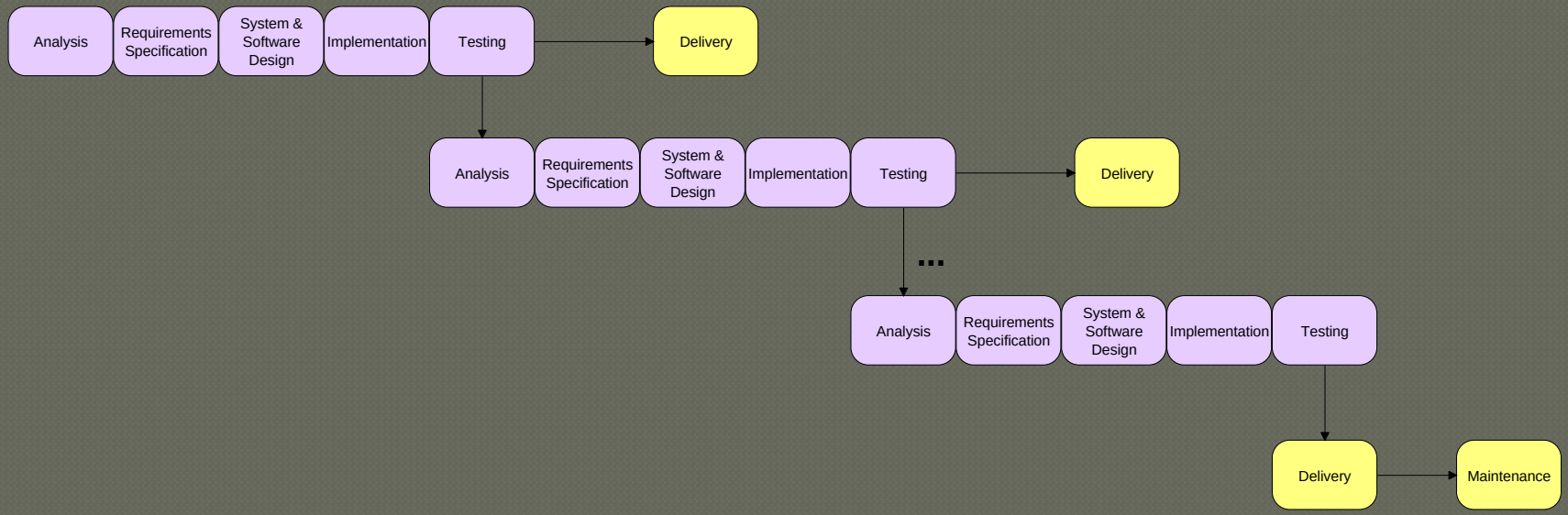


Alternative Lifecycle: Rapid Prototyping

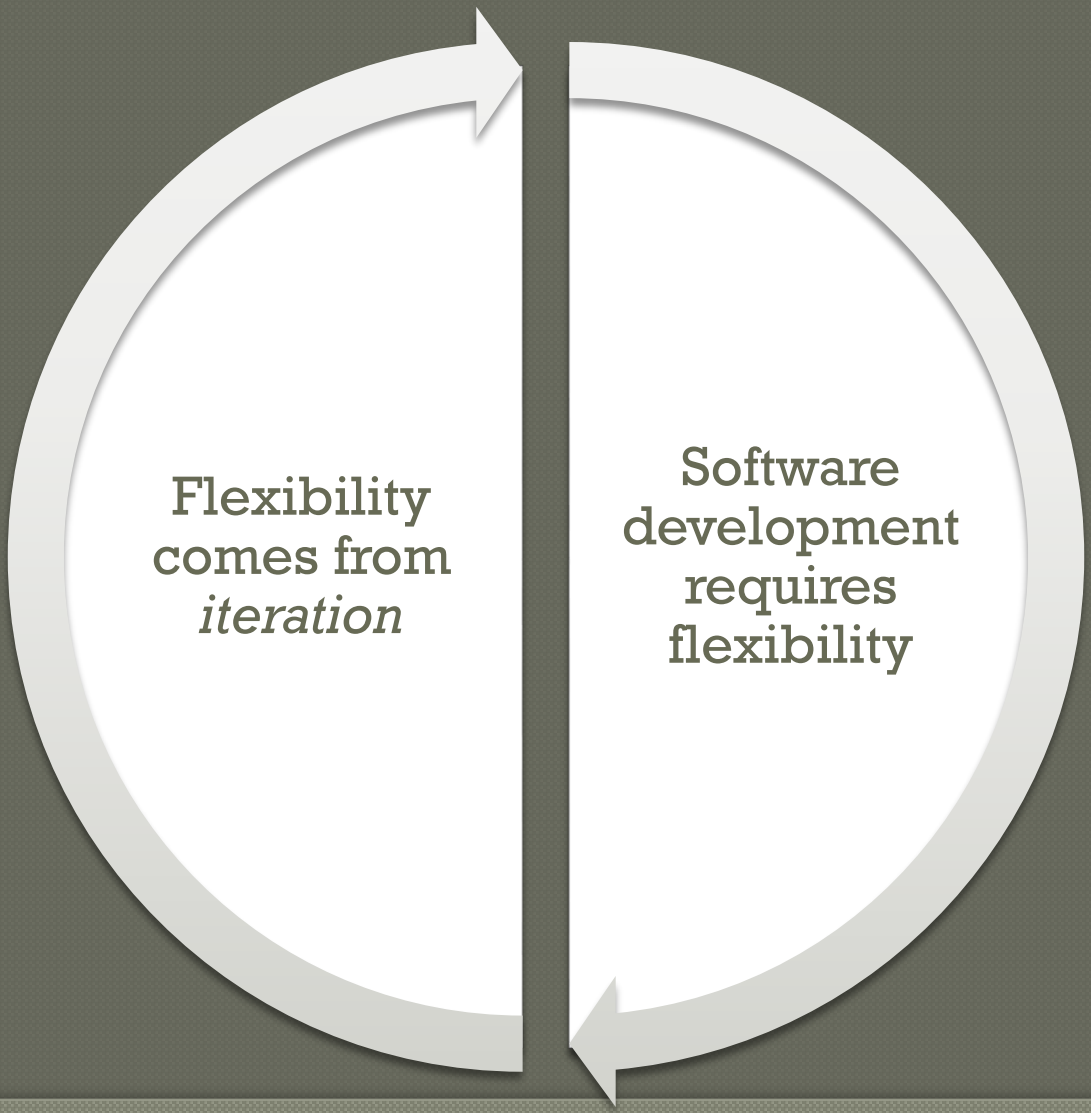


Phased Development

- Incremental
  - Initial release has limited functionality
  - Each release adds new subsystem
- Iterative
  - Each release delivers full system
  - Subsystem changes with new release



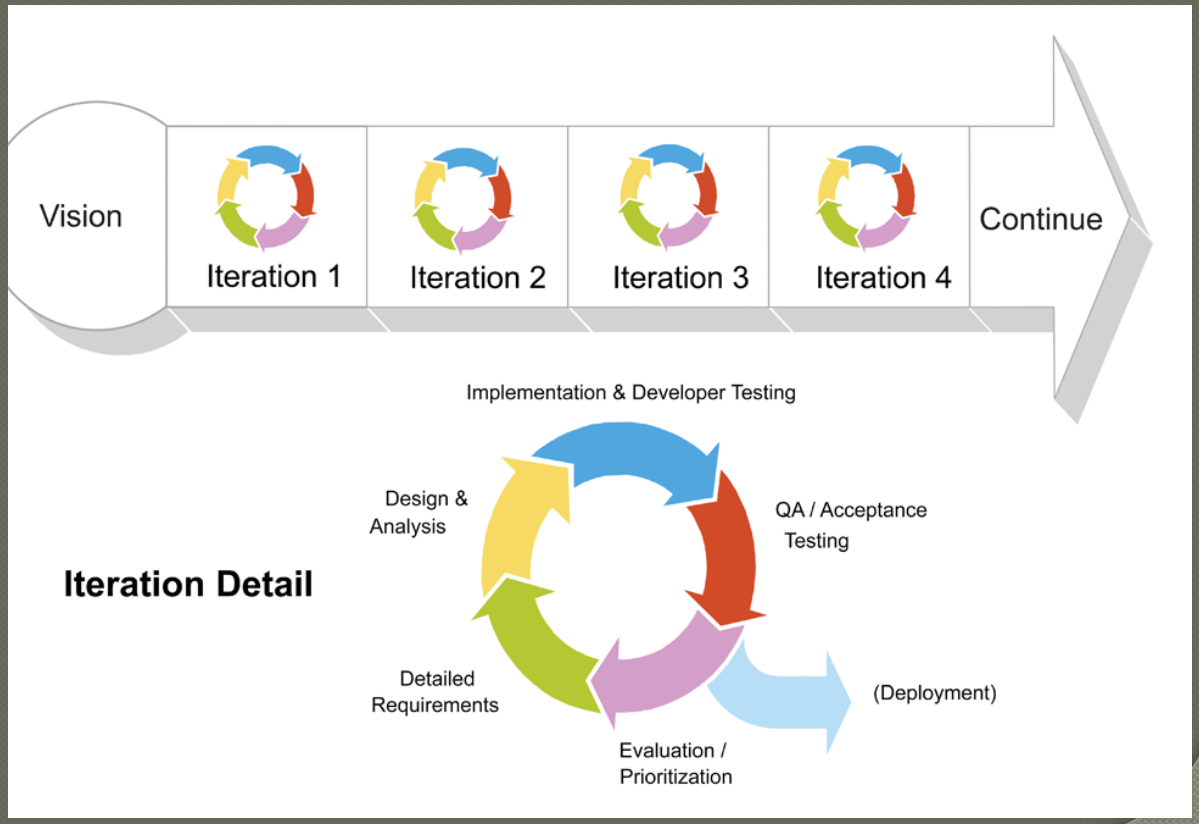
Iterative Development



An Iterative Development Lifecycle

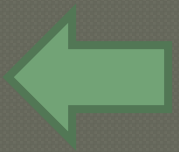
- **Vision is established**
- Development cycles through iterations
- Continuous deployment

- User-focus



The Software Development Process

- Intro to the software development process
- Software Lifecycle Models
- Agile Software Development
- Software Requirements



Agile Software Development

- An approach to software development that emphasizes:
  - Small, collaborative development teams
    - Usually 4-6 members of the team
  - Working closely with customers/stakeholders
    - Have a user representative, that interfaces regularly with customers/users/stakeholders, on the team
  - Developing software in an short iterative cycles
    - requirements / code / testing / documentation
  - Building working versions of code often
    - Incremental Development
    - Continuous Integration/Continuous Deployment (CI/CD)

Agile Manifesto

- "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - **Individuals and interactions** over processes and tools
  - **Working software** over comprehensive documentation
  - **Customer collaboration** over contract negotiation
  - **Responding to change** over following a plan
- That is, while there is value in the items on the right, we value the items on the left more."

* https://www.agilealliance.org/agile101/the-agile-manifesto/

12 Principles Behind the Agile Manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.

* https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

12 Principles Behind the Agile Manifesto

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

* https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

12 Principles Behind the Agile Manifesto

9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

* https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

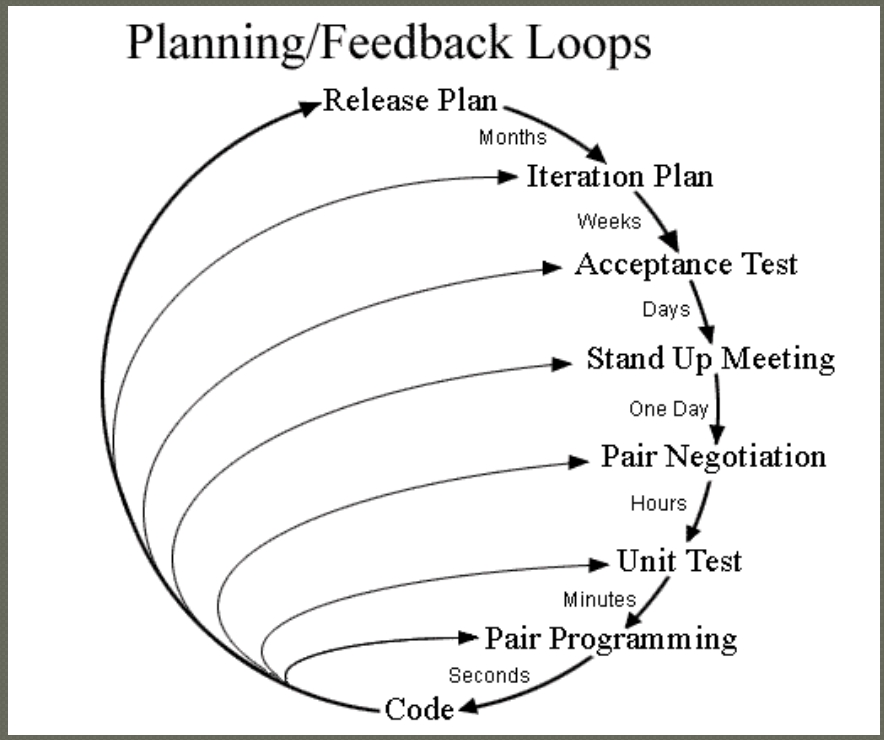
Agile Processes

- **Extreme Programming**
- **SCRUM**
- **Kanban**
- Other Agile Processes (not discussed here)
  - Dynamic Systems Development Method (DSDM)
  - Adaptive Software Development
  - Crystal
  - Feature-Driven Development
  - Pragmatic Programming
  - Lean Development

* https://en.wikiversity.org/wiki/Agile_software_development

Agile Method: XP

- On-site customer
- Planning game (with customer)
- Avoid features until needed
- Characterized by feedback loops
- Pair programming



XP Practices

1. **Test Driven Development** – creation of a test case following by writing a code that satisfies the desired test result
2. **Whole Team** – project stakeholders are actively involved and work together
3. **Continuous Integration** - team members should work on the latest release of the software, some team members who work on the delayed parts should integrate the code into the main release repositories as soon as possible
4. **Refactoring** – refactoring should be performed on a regular basis
5. **Small Releases** – functional releases should be built iteratively that form the project. These visible modules encourage customers, also used for evaluation purposes
6. **Sustainable Pace** – also known as a 40 hour week. Well being of the team is important, burnt out programmers do not perform at their best. Productivity is achieved by these means

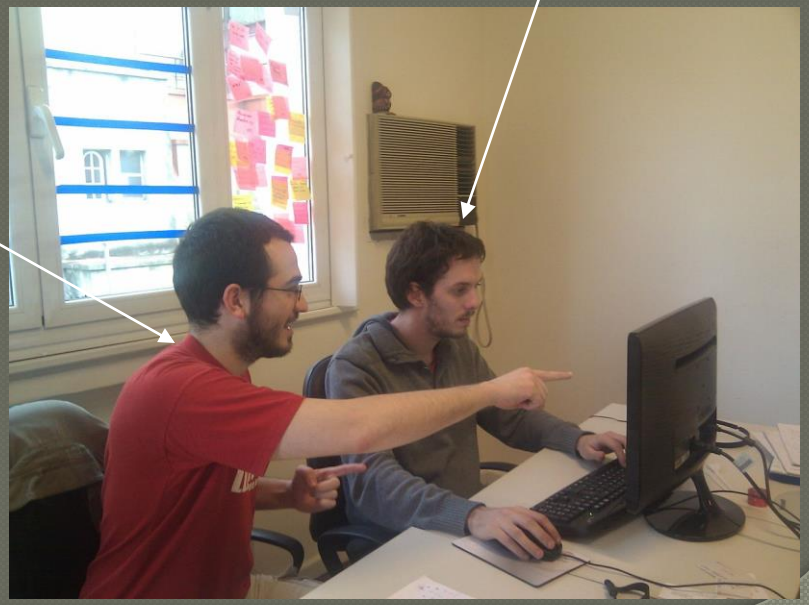
* https://en.wikiversity.org/wiki/Agile_software_development#Extreme_Programming

Pair Programming

- Two programmers, one computer
- One person writes, the other dictates
- Typically writer is less skilled

Driver: writes code

Observer: reviews each line of code as it is typed in.



Pair Programming
It's Just Awkward Sometimes

Agile Method: Scrum

- Scrum is a simple set of roles, responsibilities, and meetings that never change
- Time is divided into short work cadences, known as sprints, typically two to four weeks long
- The product is kept in a potentially shippable (properly integrated and tested) state at all times
- At the end of each sprint, stakeholders and team members meet to see a demonstrated potentially shippable product increment and plan its next steps

* http://scrummethodology.com/

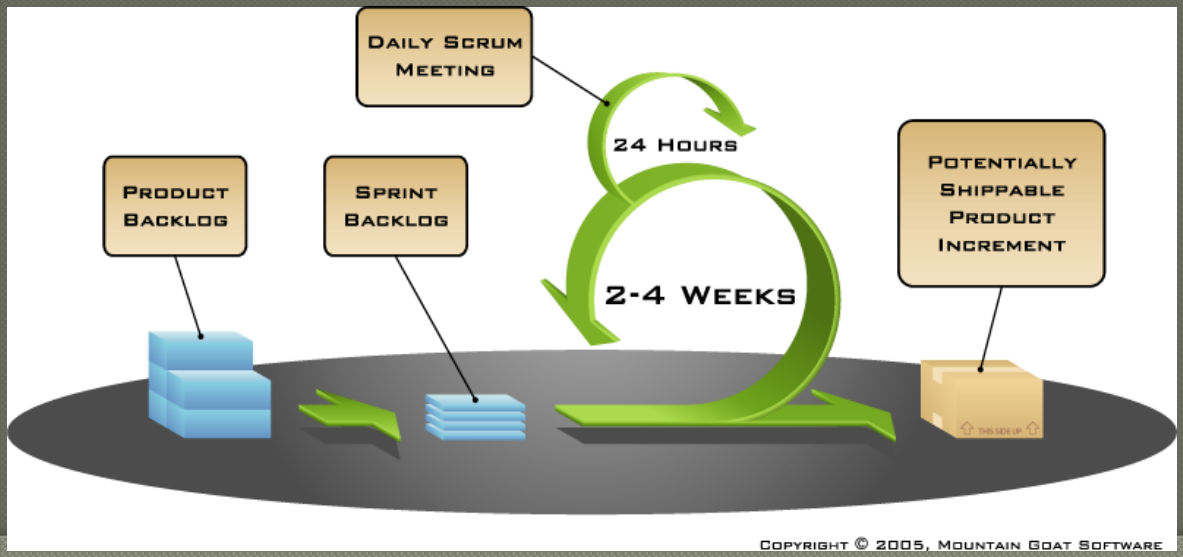
Scrum Roles

- **Product Owner:** The Product Owner should be a person with vision, authority, and availability. The Product Owner is responsible for continuously communicating the vision and priorities to the development team
- **Scrum Master:** The Scrum Master acts as a facilitator for the Product Owner and the team. The Scrum Master does not manage the team. The Scrum Master works to remove any impediments that are obstructing the team from achieving its sprint goals
- **Team:** A Scrum development team contains about 3-9 fully dedicated members, ideally in one team room protected from outside distractions. For software projects, a typical team includes a mix of software engineers, architects, programmers, analysts, QA experts, testers, and UI designers

* http://scrummethodology.com/

Scrum Phases/Steps

- Product Backlog Creation
- Sprints
  - Sprint Planning and Sprint Backlog Creation
    - Team commits to work to be accomplished in the Sprint
  - Work on Sprint/Daily Scrum Meetings
    - Development
    - Testing
    - Documentation
  - Product Demonstration
  - Retrospective



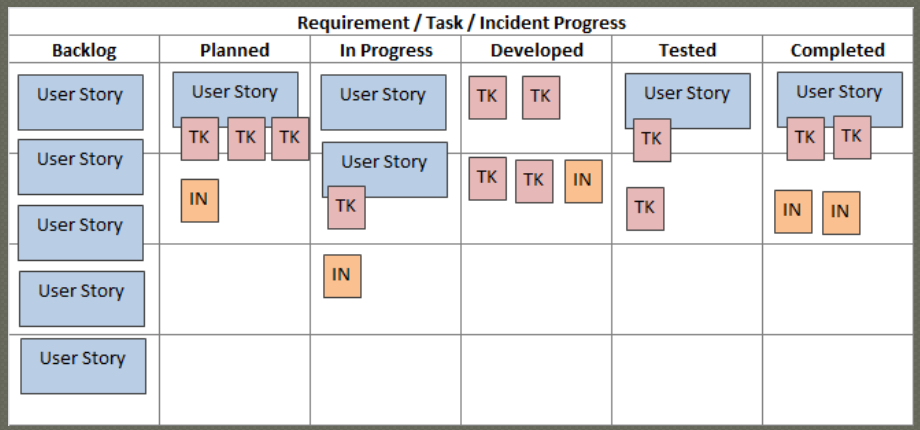
Kanban

- Kanban is a method for managing the creation of products with an emphasis on continual delivery while not overburdening the development team
- Kanban is based on 3 basic principles:
  - Visualize what you do **today** (workflow): seeing all the items in context of each other can be very informative
  - Limit the amount of work in progress (WIP): this helps balance the flow-based approach so teams dont start and commit to too much work at once
  - Enhance flow: when something is finished, the next highest thing from the backlog is pulled into play

* https://www.versionone.com/what-is-kanban/

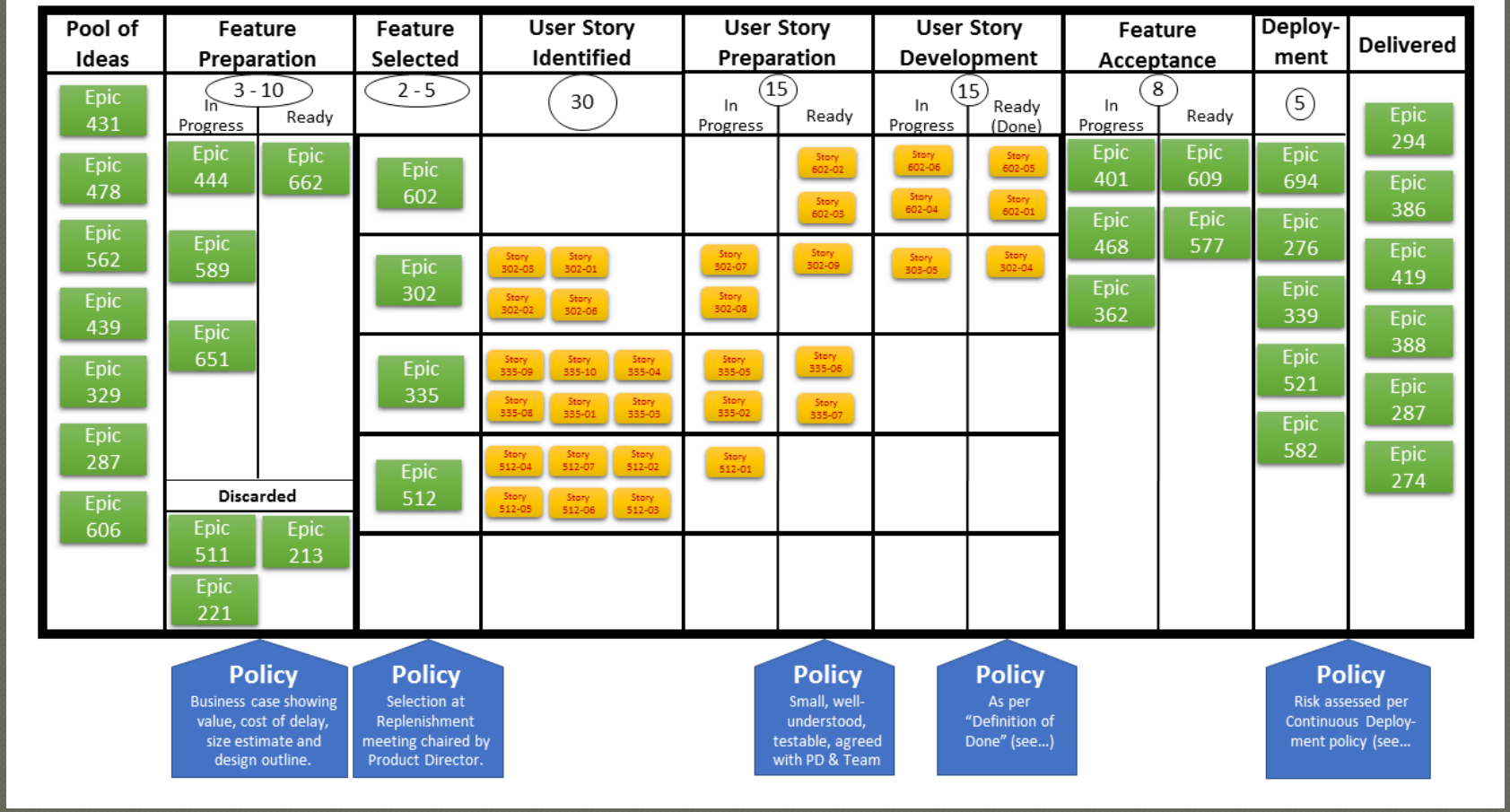
The Kanban Board

- The Kanban process begins with a prioritized list of features, called a backlog
- Agile team members work on the highest priority feature, based on their team role
- Getting features to completion in priority order is the focus
- There is no set timeframe to complete features

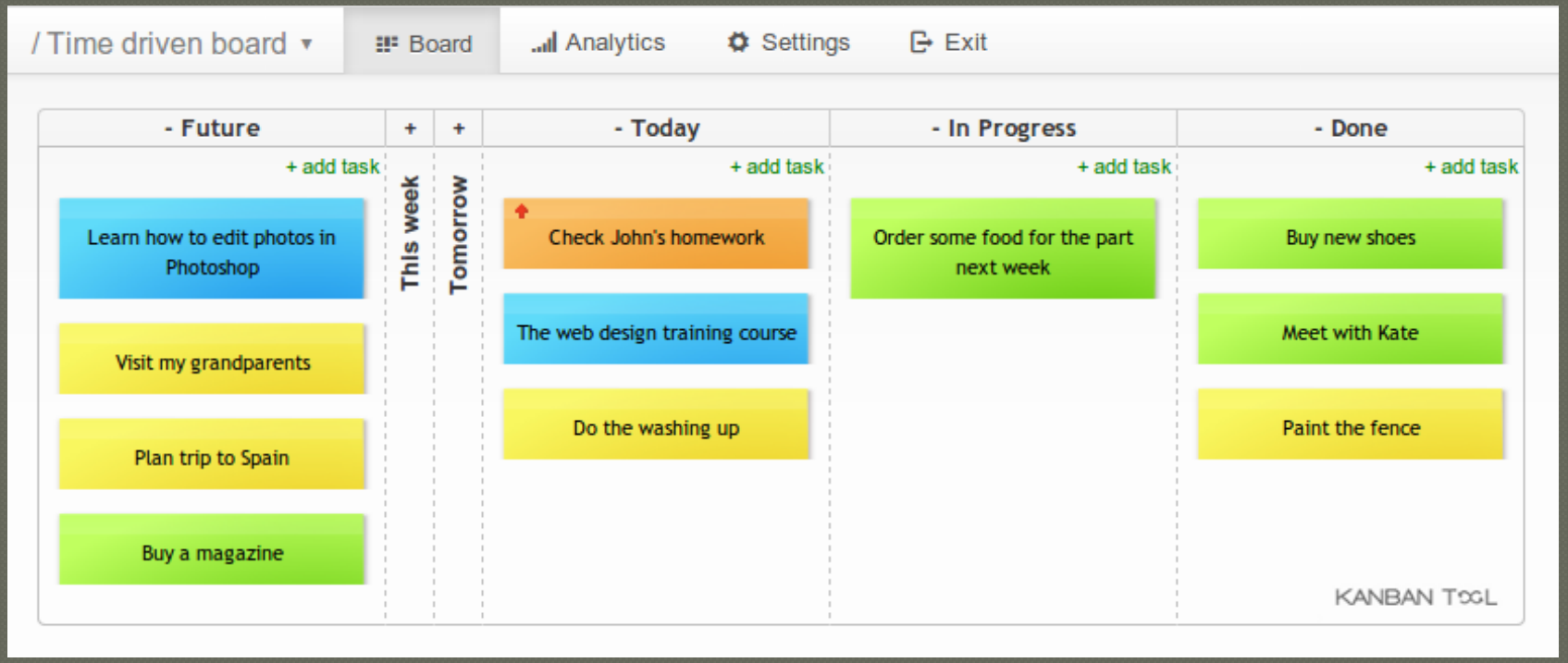


* https://www.inflectra.com/methodologies/kanban.aspx

Kanban


Pool of Ideas
Epic 431
Epic 478
Epic 562
Epic 439
Epic 329
Epic 287
Epic 606
Feature Preparation
3 - 10
In Progress
Ready
Epic 444
Epic 589
Epic 651
Epic 662
Discarded
Epic 511
Epic 213
Epic 221
Feature Selected
2 - 5
Epic 602
Epic 302
Epic 335
Epic 512
User Story Identified
30
Story 302-03
Story 302-01
Story 302-02
Story 302-06
Story 335-09
Story 335-10
Story 335-04
Story 335-08
Story 335-01
Story 335-03
Story 512-04
Story 512-07
Story 512-02
Story 512-05
Story 512-06
Story 512-03
User Story Preparation
15
In Progress
Ready
Story 602-02
Story 602-03
Story 302-07
Story 302-08
Story 302-09
Story 335-05
Story 335-02
Story 335-06
Story 335-07
Story 512-01
User Story Development
15
In Progress
Ready (Done)
Story 602-06
Story 602-04
Story 602-05
Story 602-01
Story 303-05
Story 302-04
Feature Acceptance
8
In Progress
Ready
Epic 401
Epic 468
Epic 362
Epic 609
Epic 577
Deploy-ment
5
Epic 694
Epic 276
Epic 339
Epic 521
Epic 582
Delivered
Epic 294
Epic 386
Epic 419
Epic 388
Epic 287
Epic 274
Policy
Business case showing value, cost of delay, size estimate and design outline.
Policy
Selection at Replenishment meeting chaired by Product Director.
Policy
Small, well-understood, testable, agreed with PD & Team
Policy
As per "Definition of Done" (see...)
Policy
Risk assessed per Continuous Deploy-ment policy (see...

Online Kanban tool:



https://kanbantool.com/online-kanban-board

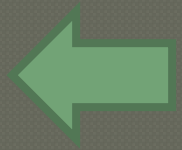
Agile for your Project?

- You don't have a design yet
- You will probably have to change it
- You don't know all the features

- Agile methods are good for you
- Kanban is probably the most productive approach for a class project

The Software Development Process

- Intro to the software development process
- Software Lifecycle Models
- Agile Software Development
- Software Requirements



Now what?

You have a project idea

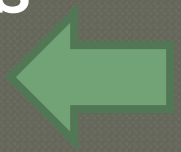
What's the first step?

Requirements

- Every software project starts with requirements
- Requirements state what the software should do or be
- Gathering requirements is called *elicitation*

The Software Development Process

- Intro to the software development process
- Software Lifecycle Models
- Agile Software Development
- Software Requirements
  - Elicitation
  - Stories (functional & non-functional)
  - Acceptance Criteria
  - Agile Terminology



Requirement Elicitation

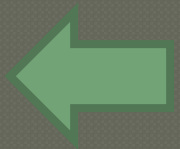
- Numerous techniques: brainstorming, interviews, focus groups, prototyping, questionnaire
- Requirements engineering is a separate field
- For you guys: talk it out. Stories should be collaborative.

Kinds of requirements

- Stories: requirements from perspective of user: "as a chef, I want to be able to download recipes to my phone so I can access them in the kitchen"
- System requirements: from perspective of software system "when the user selects 'download recipe', the browser will initiate an HTTP GET operation for the corresponding recipe URL"
- Formal requirements
  - Detailed specific requirements that are traceable and testable
- We will use stories in this class

The Software Development Process

- Intro to the software development process
- Software Lifecycle Models
- Agile Software Development
- Software Requirements
  - Elicitation
  - Stories (functional & non-functional)
  - Acceptance Criteria
  - Agile Terminology



Why stories?

Stories train developers to think from customer's perspective

Stories are lightweight and easy to modify

Stories are small, and fast to implement

Story Cards

- Short statements about what a user wants.
- Written on a notecard
- Follow this pattern:
  - As a _____, I want to _____, so I can _____(optional).

Why this format?

1

Writing the user type makes it easier to imagine the software from their point of view.

2

Writing what they want is obviously important.

3

Writing *why* lets you prioritize stories and aids brainstorming.

Examples

- As a **student**, I want to **search for courses in my major** so I can **enroll in a relevant class.**
- As a **shopper**, I want to **leave a review** so I can **express my opinion about the product I purchased**.
- As a **modder**, I want to **upload a mod** so I can **share my work with the community**.
- As an **artist**, I want to **change the color balance on a photo** so I can **correct for poor lighting conditions**.

User Story


EXTREME PROGRAMMING
I CAN'T GIVE YOU ALL OF THESE FEATURES IN THE FIRST VERSION.
www.dilbert.com scottadams@aol.com
AND EACH FEATURE NEEDS TO HAVE WHAT WE CALL A "USER STORY."
© 2002 United Feature Syndicate, Inc.
OKAY, HERE'S A STORY: YOU GIVE ME ALL OF MY FEATURES OR I'LL RUIN YOUR LIFE.

Functional vs Non-functional Requirements

Some stories describe what software *does.*

Others describe what software *is*.

Examples

- Functional: As a publisher, I want to monetize my video so I can make money.
- Non-functional: As a publisher, I want monetization to be as easy as possible so that I can focus on other tasks.

- ?: As a racer, I want the simulation to account for horizontal g-forces when I turn the car so that chassis and weight choices are meaningful.
- ?: As a racer, I want the simulation to be as accurate as possible when turning.

More Examples

- As a user, I want pages to load in under 500ms to make using the site as enjoyable as possible.
- As an artisan, I only want to do business with suppliers who have been shown to be trustworthy so that I don't waste time and money on scams.
- As a chef, I want all ingredients to comply with federal health regulations so that my food is safe to eat.
- As a lawyer, I want case law to include recent rulings so that I don't overlook legal changes.
- As a simulator, I want gigantic simulations to run without crashing the system.

Why does this matter?

Non-functional stories shape how we interpret functional stories.

Two different teams can implement the same story radically differently

Consolation

- Functional vs. non-functional requirements are a difficult subject
- For more practice, try generating non-functional stories related to these attributes:
  - Security
  - Usability
  - Performance
  - Safety
  - Aesthetics (look and feel)

How Do We Determine the Extent of a Story?

Place a limit on the amount of time a story can take to implement (e.g., 1 week)

Split or merge stories until they meet this time.

Ideally, each story corresponds to one person

Examples of Splitting Stories

- "As an administrator, I want to see a log of all site activity so that I can respond to cyber attacks."

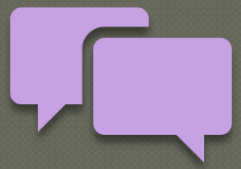
- Splits →
  - "As an administrator, I want to see a log of all downloads so I can respond to DDoS attacks."
  - "As an administrator, I want to see a log of all password changes so I can respond to phishing attacks."

Story Guidelines

Stories must be specific enough to suggest validation criteria:

Bad: "As a user, I want the system to be good so I can be happy."

Good: "As a student, I want to view my grades so I can know whether I should drop the course."

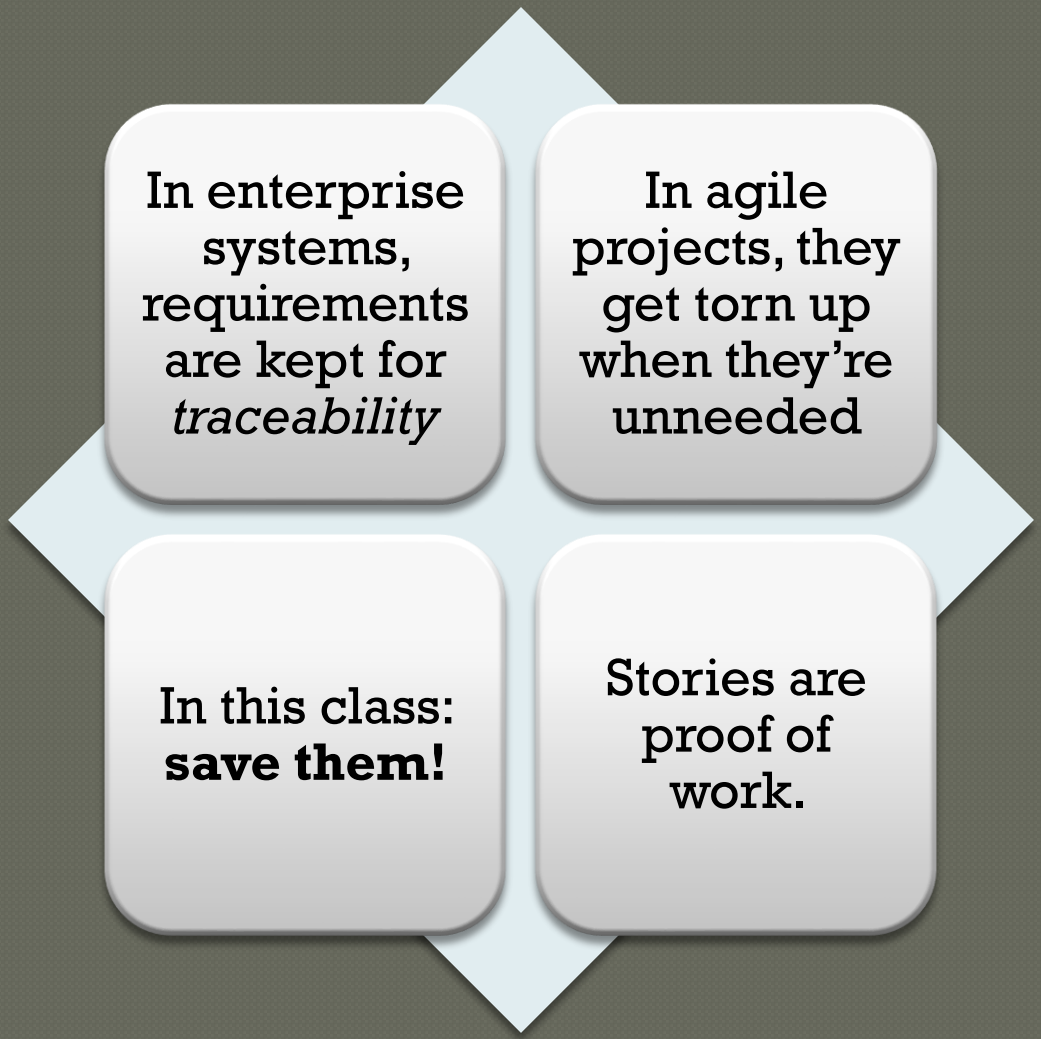


Stories should (in this class) refer to specific users or personas:

Bad: "As a user, I want to login using Facebook so I don't have to remember a password."

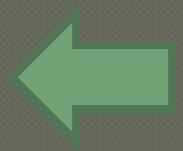
Good: "As a **customer**, I want to login using … "

What Happens to Stories?



The Software Development Process

- Intro to the software development process
- Software Lifecycle Models
- Agile Software Development
- Software Requirements
  - Elicitation
  - Stories (functional & non-functional)
  - Acceptance Criteria
  - Agile Terminology



Acceptance Criteria (Confirmations)

- Listed on the back of the user cards
- Specific statements that can be evaluated to determine if we have implemented this story
- Follow this format:

Given _____, **when** _____, then _____

Examples of Acceptance Criteria

- **Given** a valid image file, **when** the user uploads an image, **then** the image will be added to the database.
- **Given** that a user is located in the vicinity, **when** the user walks past a participating restaurant, **then** they should be sent a notification of daily specials.
- **Given** the user is logged in, **when** the user clicks "log-out", **then** the user should no longer be logged in and should be redirected to the homepage.
- **When** the user clicks "refresh", **then** the most recent weather is retrieved from the API.

“Given” vs “When”

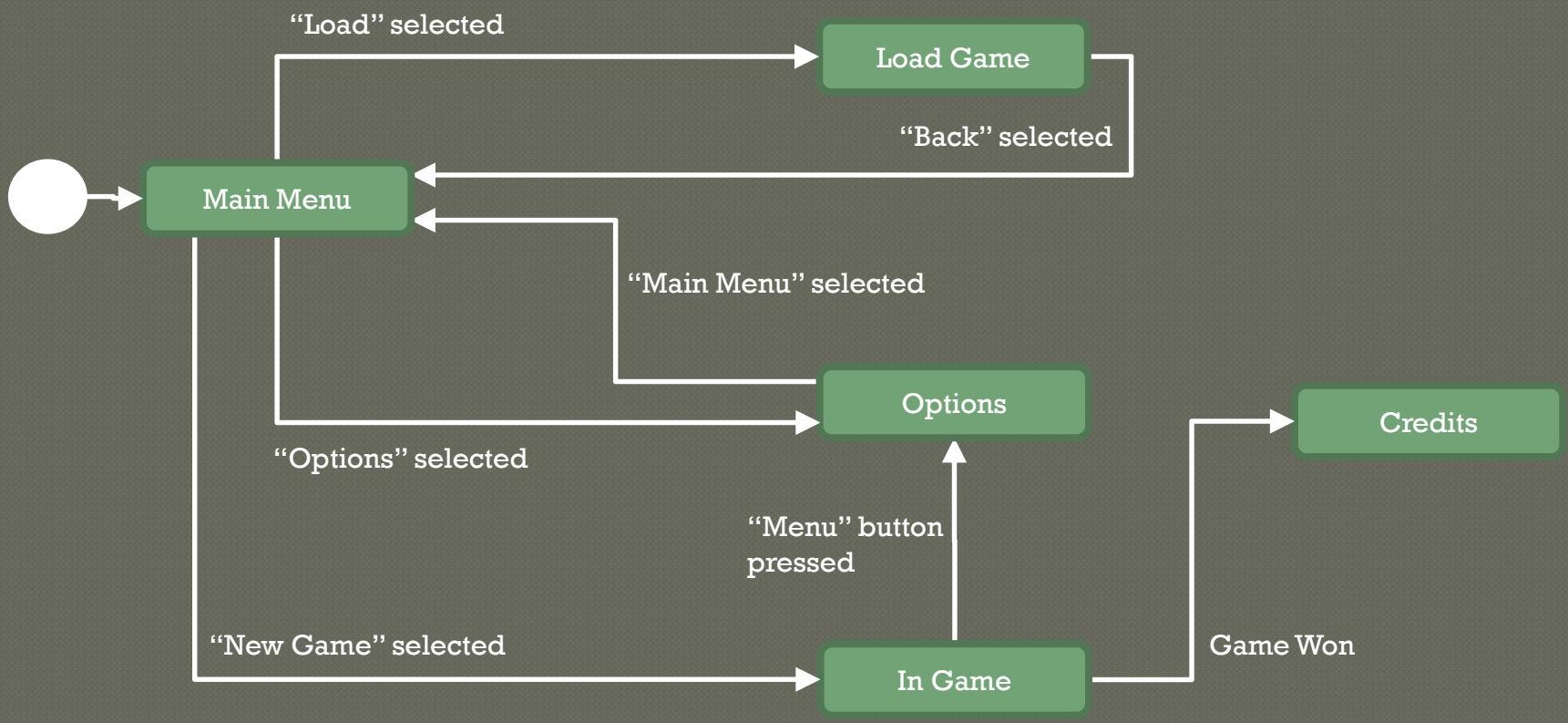
- Given [state of software]:
  - Here, we specify the context in which the action can occur
- When [condition]:
  - Here, we specify the condition that will cause the software to change state.
- Then [action]:
  - Here, we specify the action that will take place (and optionally the new state the software will enter).

Detailed Example

- Given [the user is on the main menu],
  when [the user selects “load game”],
  then [the current state will change to the load game menu]

- Given [the user is in the ‘view product’ activity],
  when [the user clicks ‘edit’],
  then [the product description becomes editable and gains focus]

State Machines



Non-functional Stories Can Have Validation Criteria, Too

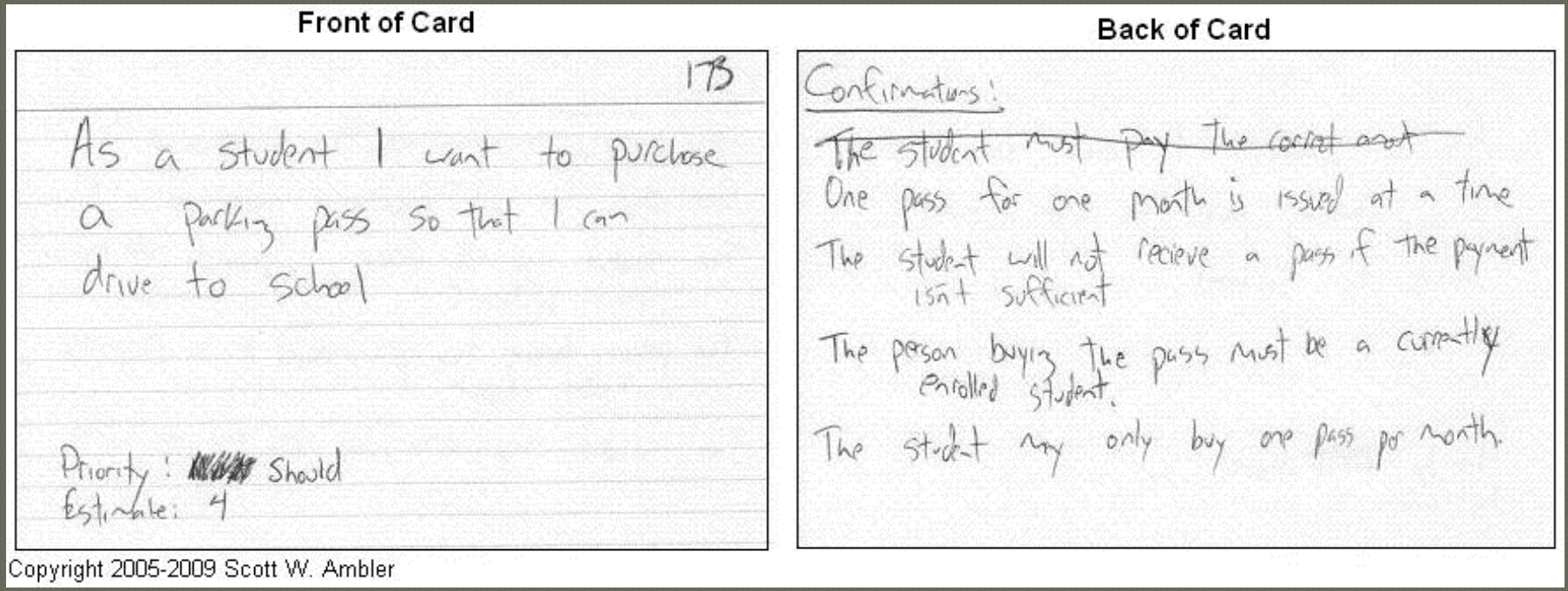
- As a musician, I want flipping through a manuscript to be as smooth as possible with no buffering. [performance]
- Validation criterion:
  - Given that I am viewing a manuscript, when I click 'next page', then the next page should appear within 500 ms.
- As a user, I don't want my friend requests to contain spam. [controlling experience]
- Validation criterion:
  - When I view my friend requests, none should include spam accounts.

Acceptance Criteria Guidelines

- Must be **testable**, not vague:
  - Bad: Given that the user is logged out, when the user favorites a post, then the correct action will be taken
  - Good: Given that the user is logged out, when the user favorites a post, then the user will be redirected to the sign-in page

  - Bad: When the developer has selected some code, the user must see a popup
  - Good: When the developer has selected some code, the code snippet suggestion popup must appear

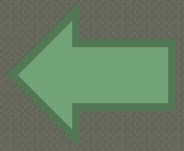
Index Card Approach



* A confirmation is essentially the same thing as an acceptance criterion.

The Software Development Process

- Intro to the software development process
- Software Lifecycle Models
- Agile Software Development
- Software Requirements
  - Elicitation
  - Stories (functional & non-functional)
  - Acceptance Criteria
  - Agile Terminology



Epics

- Stories which cannot be completed in a single sprint
- Will be split into regular stories when you start working on it
- Highly broad, general features or desires
- Don't need to be testable

Examples of Epics

- As a storage customer, I want to be able to upload a variety of files and have integrated viewers so that I don't have to download the files to see what's on them.
- As a game developer, I want to be able to browse a marketplace of assets so that I don't have to develop them myself.
- As a designer, I want to be able to manipulate objects in 3D so that I don't have to rely on inflexible axis-aligned views.

Themes

- Themes group stories together, and cut across different epics
- Themes can be used to decide who to assign stories to
- Related to *quality attributes*
- Often come from story's justification

Assigning Themes

- As a security auditor, I want to be able to filter logins by IP address so that I can determine if a user has shared their account. [**maintaining security**]
- As an administrator, I want to be able to reset a user's password so that I can rectify a situation where a password has been shared or forgotten. [**maintaining security**]
- As a user, I want to see a display showing how strong my password is so that I can choose one which is hard to guess. [**maintaining security**]

More Theme Examples

- As a photographer, I want to retrieve the raw source image I've uploaded along with the resized version so that I don't have to maintain my sources separately. [**ease of use**]
- As a customer, I want to be able to preview photographs on the search page so that I don't have to view the preview on a different screen. [**ease of use**]
- As a customer, I want the final licensing price to be displayed by a image thumbnail so that I don't have to compute how much it will cost after my discount. [**ease of use**]

Issues With Themes

- Not always clear which theme to assign:
  - As a chat moderator, I want to be able to change chat to "slow mode" so that I am not overwhelmed by spam. [**discouraging bad behavior**] or **[maintaining ease-of-use]?**
- In non-agile, multiple themes (quality concerns) are okay. In agile, only use one.
- Themes can simply be groups of story cards.