CSC207 Week 5

Larry Zhang

Logistics

Next week: Reading week

- no lecture
- no lab
- A1 Part 2 will be out soon, you should be working on that.

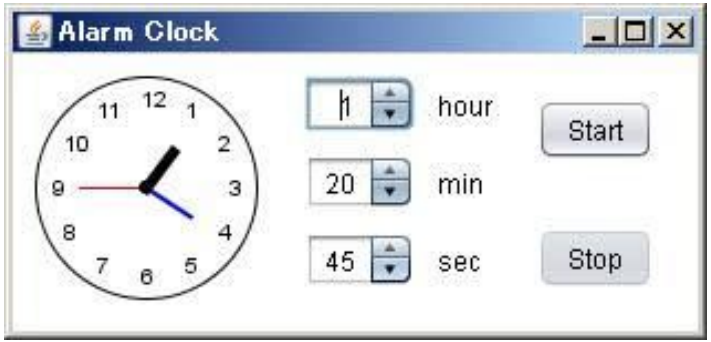
Design Patterns

- So far we have learned how to get things done in Java.
- Now we start talking about **design patterns**, which is about how get things done in the **best possible way**, so that
  - your code has great extensibility
  - you code is very easy to maintain
  - different modules in your code can be reused easily
  - different developers can work on different parts of your software simultaneously without interfering with each other
  - errors and misuses of the software can be prevented from emerging

Model-View-Controller (MVC)

a major architectural pattern
for software with graphical interface

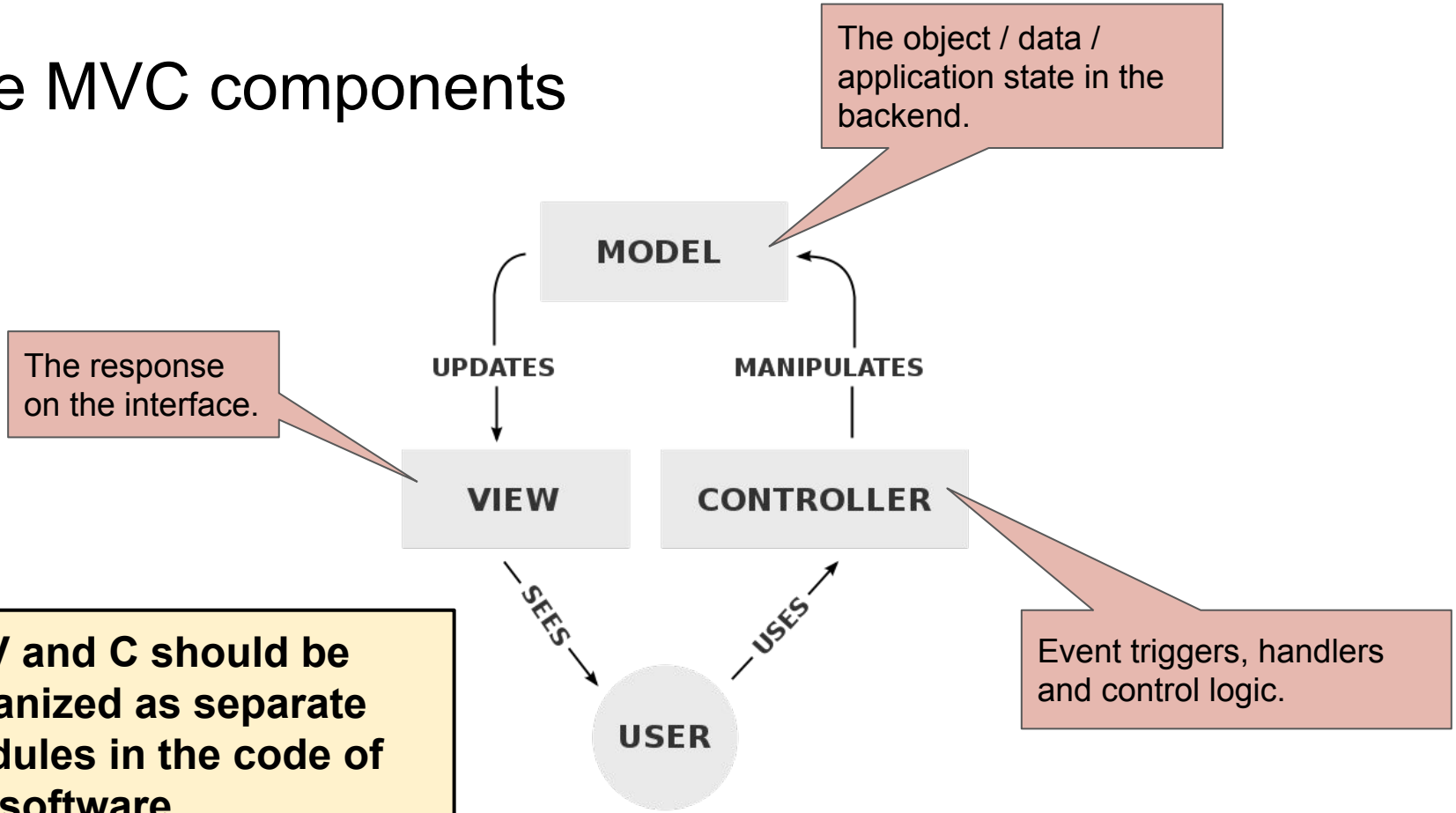
Motivating example



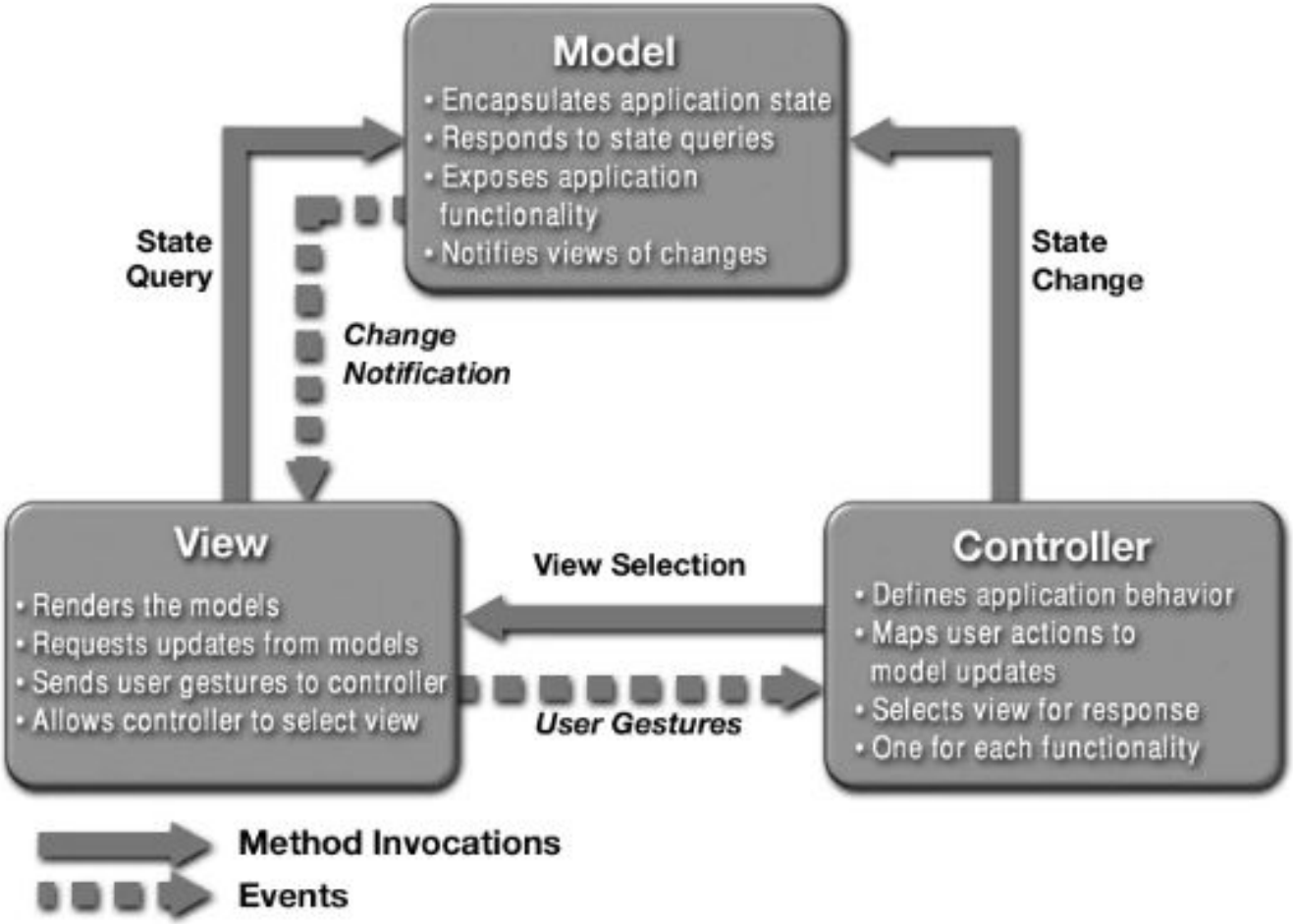
Technically, you can implement everything of this game in **one giant file** AlarmClock.java, but can you do better than this?

- Suppose two people want to collaborate on this project, how do you divide up the work? What are the different components of this software?
- You want to change the front-end interface from one look to another, what do need to change? Everything or just some part of your code?

The MVC components

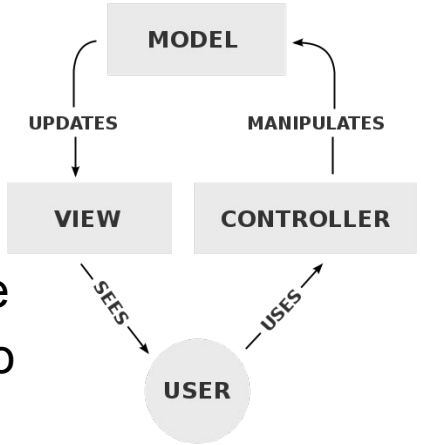


M, V and C should be organized as separate modules in the code of the software.


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
State Change
Change Notification
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Model, View, Controller in separate files

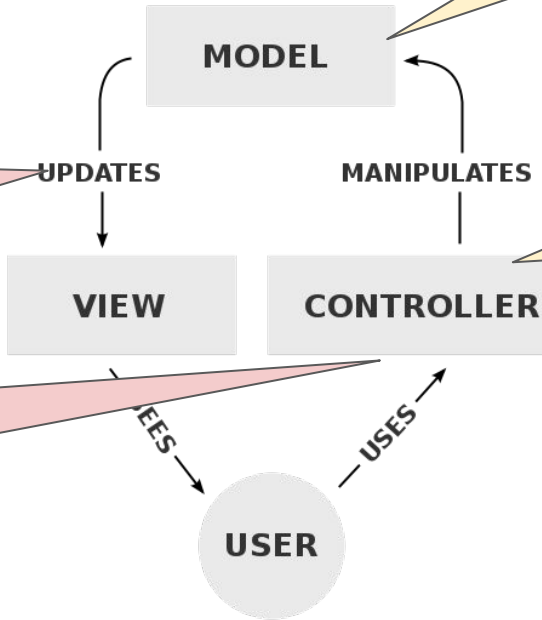
- **Model**: the internal object, data, application state
- **View**: the user interface, reflecting the changes in **model**.
- **Controller**:
  - receive an event triggered from the **view**.
  - can manipulate the **model** and change the application state.
  - connects the **model** and the **view**, so that when change to the **model** happens, the **model** and notify the **view** to make the corresponding change.



What to do in terms of Java

Define the classes like what we have been doing, like Jug and JugPuzzle

How to do this? Whenever there is a change in the **model**, **update** the **view** correspondingly.



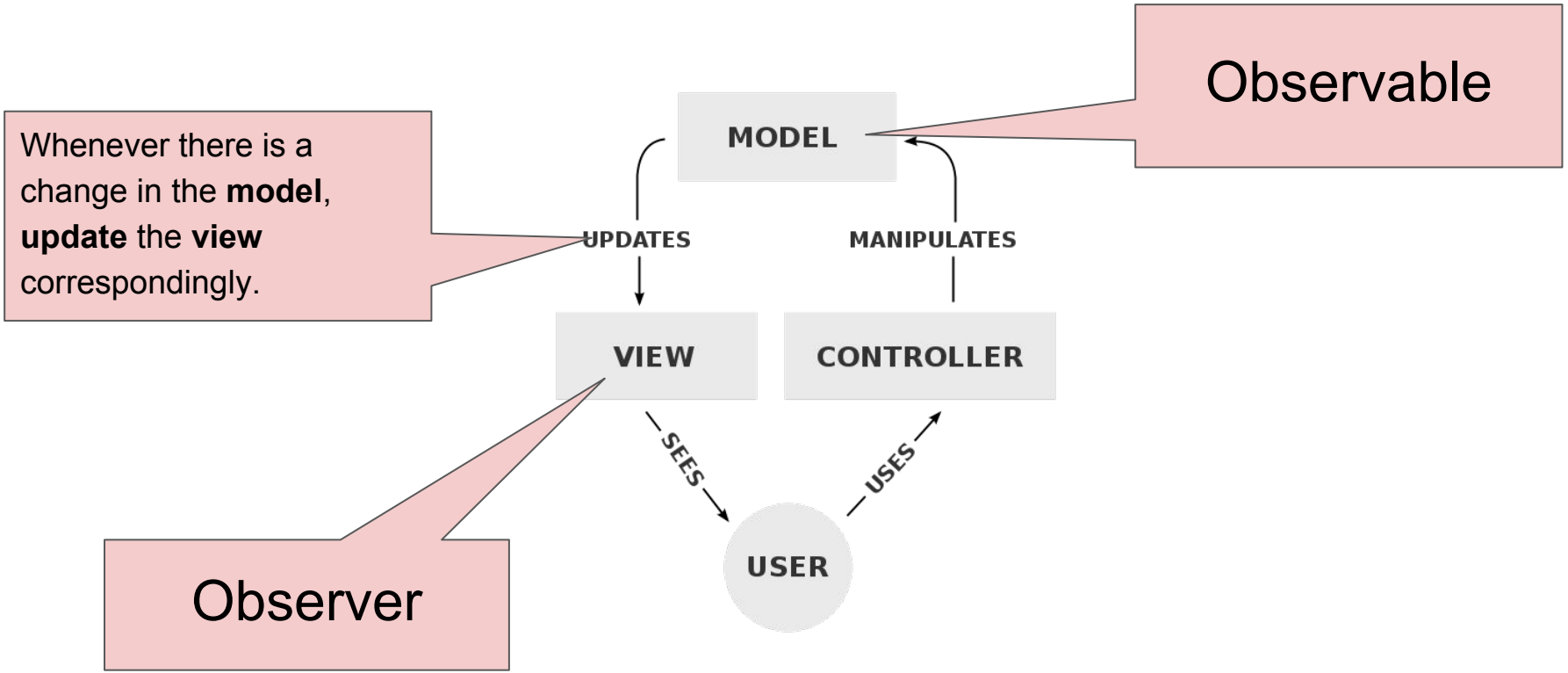
Catch interface events and trigger handlers that manipulate the classes in model, like the JugPuzzleController.

The controller hooks and the **model** and the **view** in a certain way so that the **update** can happen.

Design Pattern: Observer and Observable

- What it achieves: When one object (observable) changes state, all its dependents (observers) are notified and updated automatically.
- How to make a class **Observable**?
  - make the class **extend** the class called **Observable**.
  - You may override / use the following methods
    - addObserver(), add an observer
    - setChanged(), set the “changed” flag to be true
    - notifyObservers(), tell all observers about the change
- How to make a class **Observer**?
  - **implement** the interface called **Observer**
  - need to implement an **update**() method, which is called when the observer is notified about a change by the observable.

Who’s the Observer; who’s the Observable?



MVC DEMO #1

simple implementation

Model: Balloon.java

View: TextView.java

Controller: AutomaticController.java

MVC DEMO #2

replace auto-controller with keyboard-controller

`Model: Balloon.java`

`View: TextView.java`

`Controller: KeyboardController.java`

MVC DEMO #3

GUI controller and TextView

Model: Balloon.java

View: TextView.java

Controller: GUIApp1.java

MVC DEMO #4

GUI controller and GUI View

Model: Balloon.java

View: GUIView.java

Controller: GUIApp2.java

Summary of MVC

- Better extensibility and reusability
  - once separately properly, you can use these model view controller modules as building blocks to build any kind of application you need without needing to write much code, rather than rewriting all the code everything you need a new application.
- Supports better collaboration
  - The model developer, the view developer, and the controller developer can work on their own module simultaneously, without having to communicate with each other very frequently.
- Your code is one step closer to being elegant.

MVC is not the only architectural pattern

- Model-View-Presenter
  - https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter
- Presentation-Abstraction-Control
  - https://en.wikipedia.org/wiki/Presentation-abstraction-control
- Model-View-Adaptor
  - https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93adapter
- Model-View-Viewmodel
  - https://en.wikipedia.org/wiki/Model_View_ViewModel

NEXT TOPIC

The next topic is Scrum.

Software Engineering

- Historically, software used to often be written by a single developer
- As software grew more complex, teams of people started working on single projects
- Increased budgets and risk caused companies to start finding ways to try and minimize potential pitfalls

Credit: This slide and several following slides are stolen from Sadia Sharmin.

Waterfall vs Agile

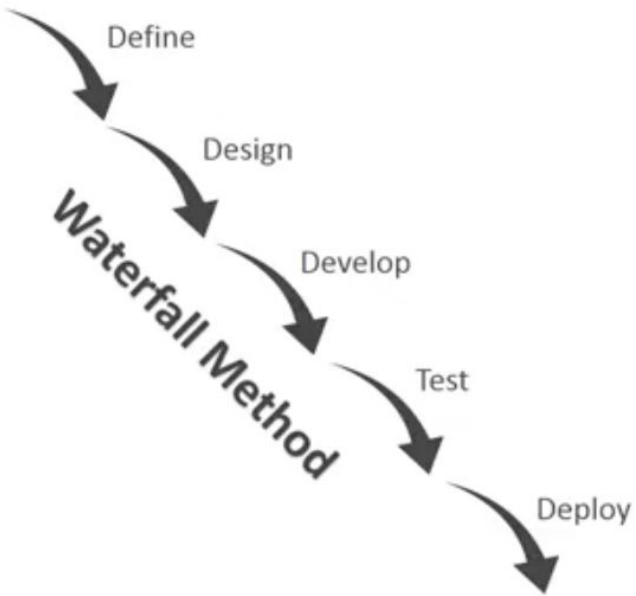
These are the two major contrasting development methodologies.

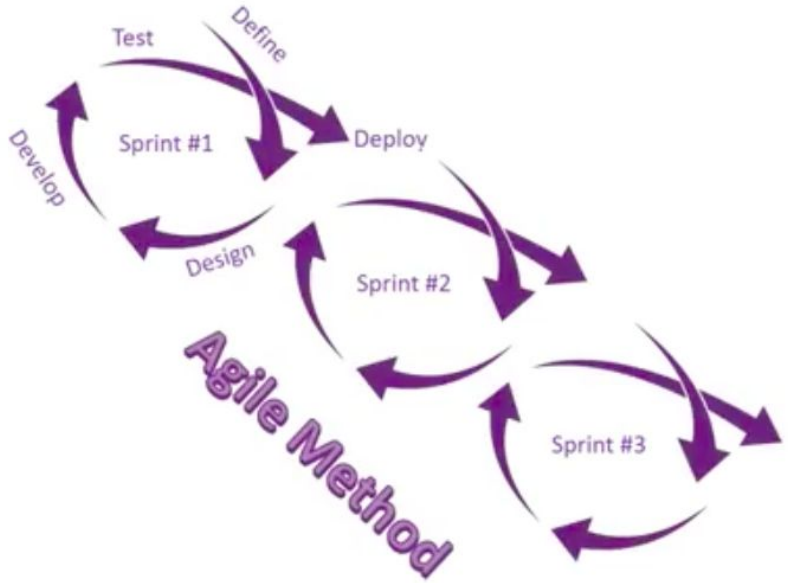
Waterfall

1. Gather and document requirements
2. Design
3. Code and unit test
4. Perform system testing
5. Perform user acceptance testing (UAT)
6. Fix any issues
7. Deliver the finished product

Agile

- Iterative approach
- A little bit of work done on each phase everyday
- Functional product (deliverables) ready after every increment
- Embracing change
- Continuous revisions; frequent feedback


Define
Design
Develop
Test
Deploy
Waterfall Method

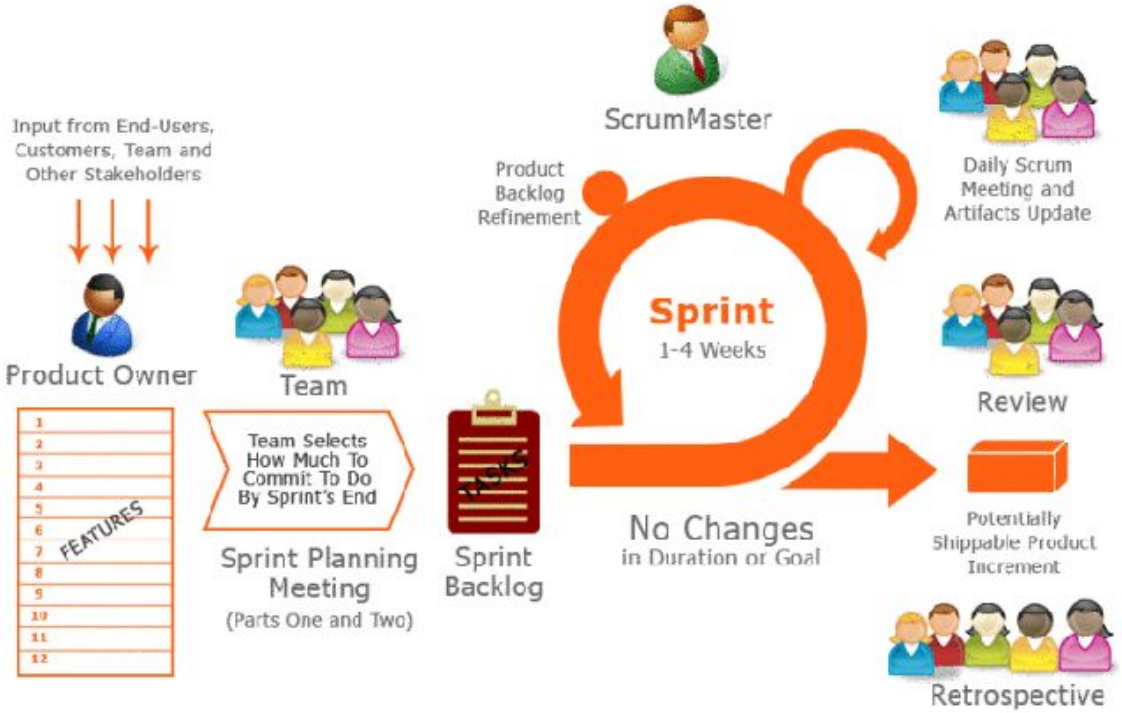

Test
Define
Develop
Sprint #1
Deploy
Design
Sprint #2
Sprint #3
Agile Method

Video: Waterfall vs Agile

Scrum

- An agile methodology
- Work is done in short (2-4 week) iterations called "sprints"
- Should have a visible, functional product at end of each sprint

A Scrum Sprint



http://axiom.utm.utoronto.ca/~207/17f/lectures/scrum/

The iterations for Scrum

```
while(!isProjectFinished){
        Plan
        Build
        Test
        Review
}
```

Important Principles

- Iterative
- Involve the customer in what is delivered, when.
- Continuously respond to customers changing needs, changing requirements.
- Continuously deliver working partial product. Confirms customer understanding as well as developer understanding.
- Continual reporting on effectiveness of team, progress on development, cost.
- Agile in general is good for managing customers expectations (Arnold's experience).

Roles in Scrum

- Product Owner
  - Knows what the product is supposed to do.
  - Maintains a **Product Backlog** (prioritized list of **user stories,** i.e., features) and communicates with the team about vision.
- Scrum Master
  - Facilitators of the Scrum events; services both the Product Owner and the Team
- Team
  - Designs and builds what the Product Owner wants.
  - Analysts, Designers, Developers, Testers, ...

The Process

- **Sprint planning meeting**
  - A meeting with PO, SM and Team. Take highest priority user stories and create the **Sprint Backlog** (the feature list for the current sprint).
- **Sprint (1 to 3 weeks)**
  - Team works on delivering the Sprint Backlog. They architect, code, test, document etc.
  - **Daily Scrum meeting**:
    - A meeting with SM and Team. 15 Minute standup meeting, each team member says:
      - What I did yesterday.
      - What I am doing today.
      - Obstacles I face.
- **Sprint Review:**
  - A meeting with PO, SM and Team. Show off product to PO. Acceptance test as well as discussion of Scrum Process improvement.
- **Sprint Retrospective**:
  - Discuss how to improve the Scrum process

References

Arnold’s notes on Scrum:

- http://axiom.utm.utoronto.ca/~207/17f/lectures/scrum/

Scrum Tutorial:

- https://www.tutorialspoint.com/scrum/scrum_tutorial.pdf

Video introduction to Scrum

- https://www.youtube.com/watch?v=9TycLR0TqFA