SOFTWARE DESIGN - I

Content in part from Chapter 5 of "Head First Software Development", Pilone et al.

Miami University Software Technology & Analysis Group (MUSTANG)
Computer Science & Software Engineering
Miami University, Oxford, Ohio, USA

AGENDA

- Review
  - User stories & tasks
- Software Design Introduction
  - Arch. Vs. Detailed
- Use Cases & Use Case Diagrams
- Case Study

AGENDA

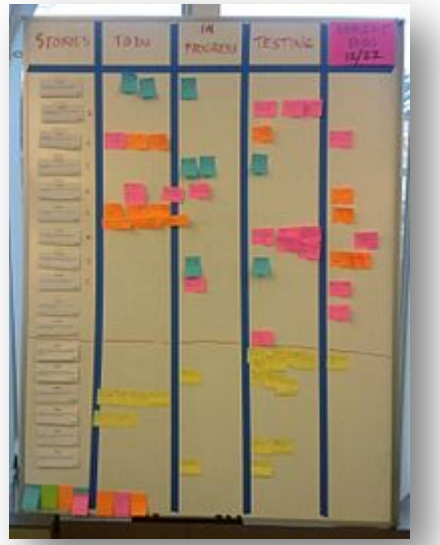
- Review
  - User stories & tasks
- Software Design Introduction
  - Arch. Vs. Detailed
- Use Cases & Use Case Diagrams
- Case Study

Retrospective Questions

What is the relationship between stories and tasks?

What should we do with our stories before estimating/planning poker?

Why does velocity help us not plan for the unexpected?



User Stories to Tasks

Breaking up the work and keeping track

User Stories are hard to take to code

- In customer language
- Features
- "Big", multi-skilled

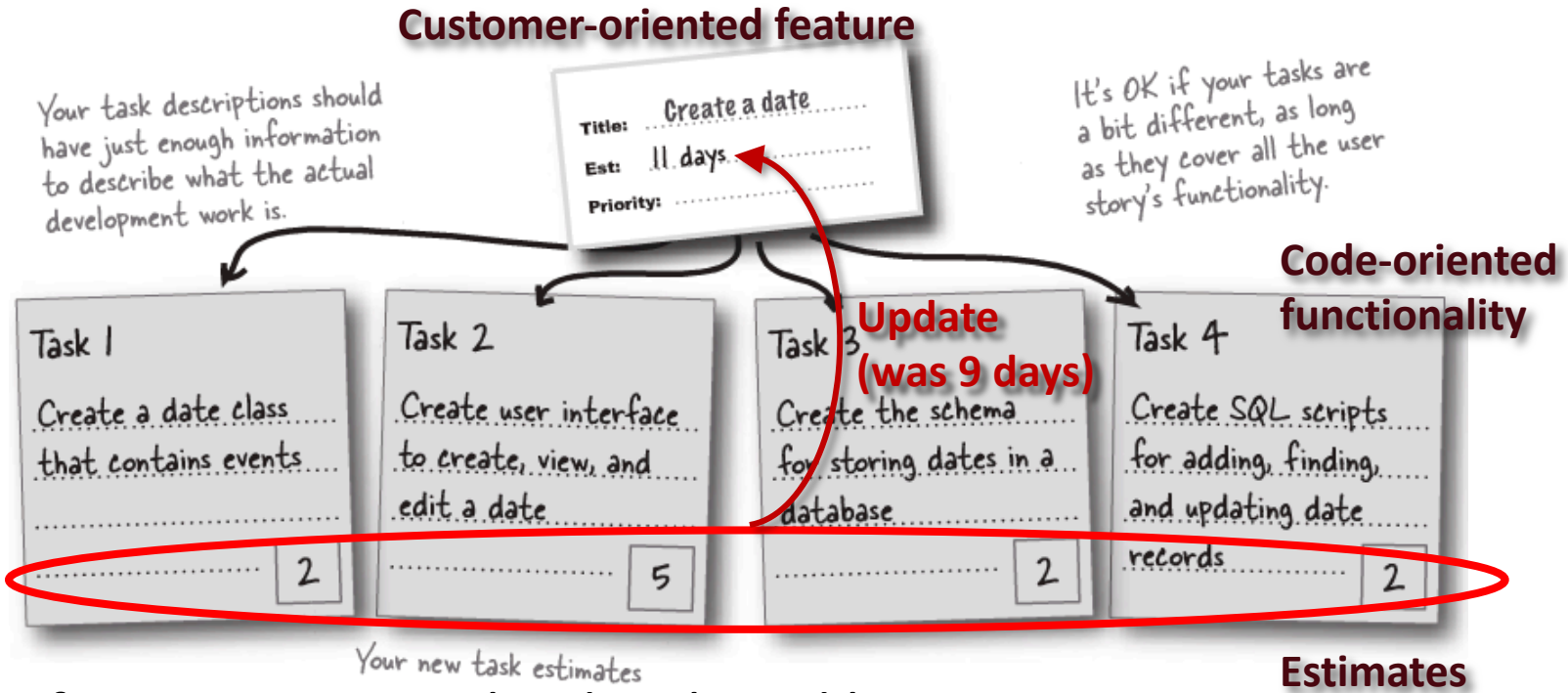
Answer is to subdivide Stories into *Tasks.* Task is:

- Written in technical language
- Small unit of developer work
  - Divided by skill, subsystem, etc.
- Often a module of executable functionality (class or two)

Goals of Creating and Executing Tasks

1. Break iteration's user stories into units of work to be performed (**tasks**)
2. **Estimate** work-days of the tasks
   - More accurate than user story estimates
3. **Prioritize tasks** by user story priorities and task dependence
   - Divvy up work according to skills (or, to *learn* skills!)
   - Try to order assignment of tasks so that system always runs
4. Daily start-of-day **stand-up**s to assess progress and identify problems
5. Track work on **big board**, with **burn-down chart**

NEW CONCEPT: TASK



Tasks are units of computation to be developed by one person

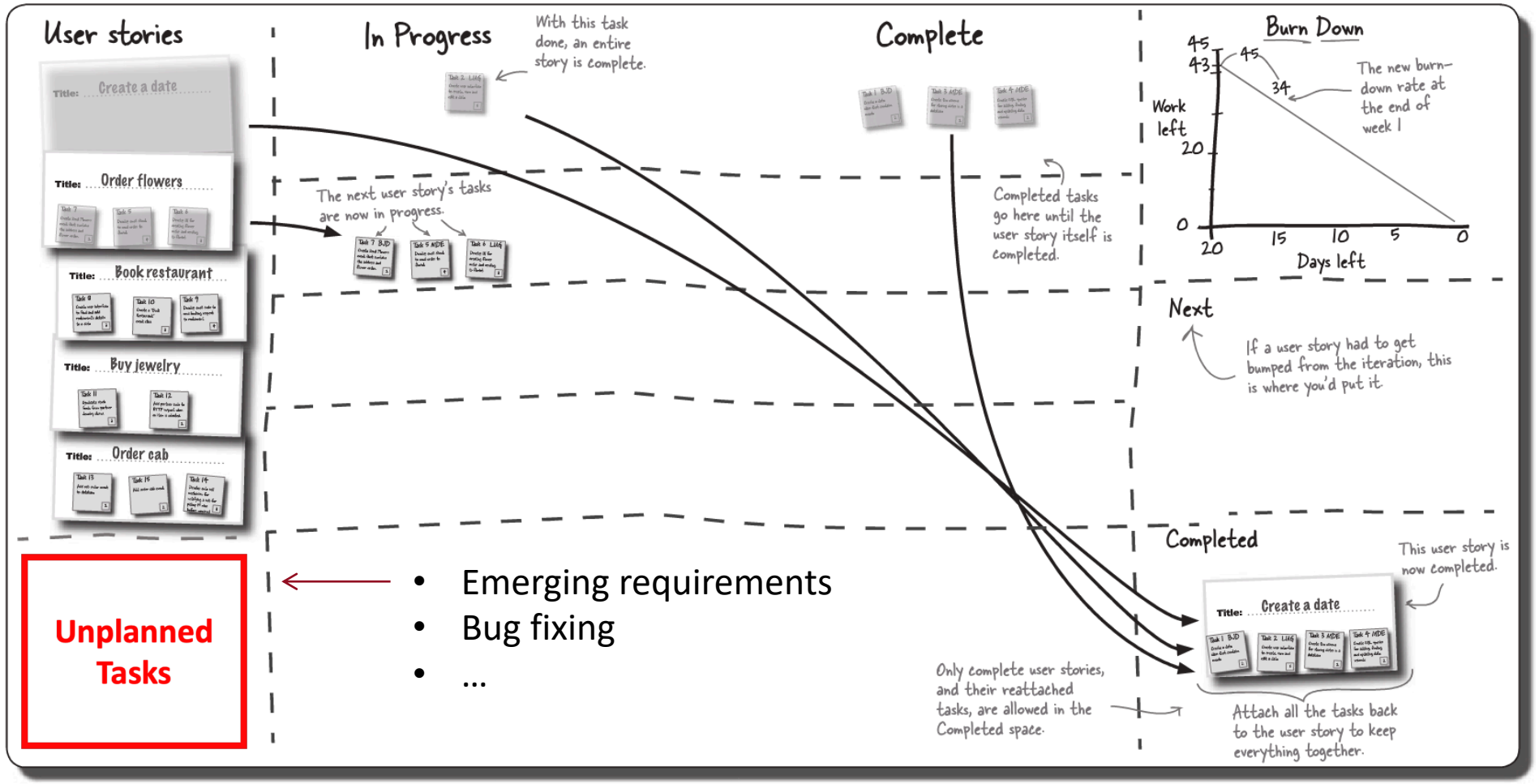
- Technically oriented; not end-user features

Often distinguished by skill set or subsystem

Estimates likely more accurate

Keeps track of what's doing, who, and how fast

EXPANDED CONCEPT: BIG BOARD

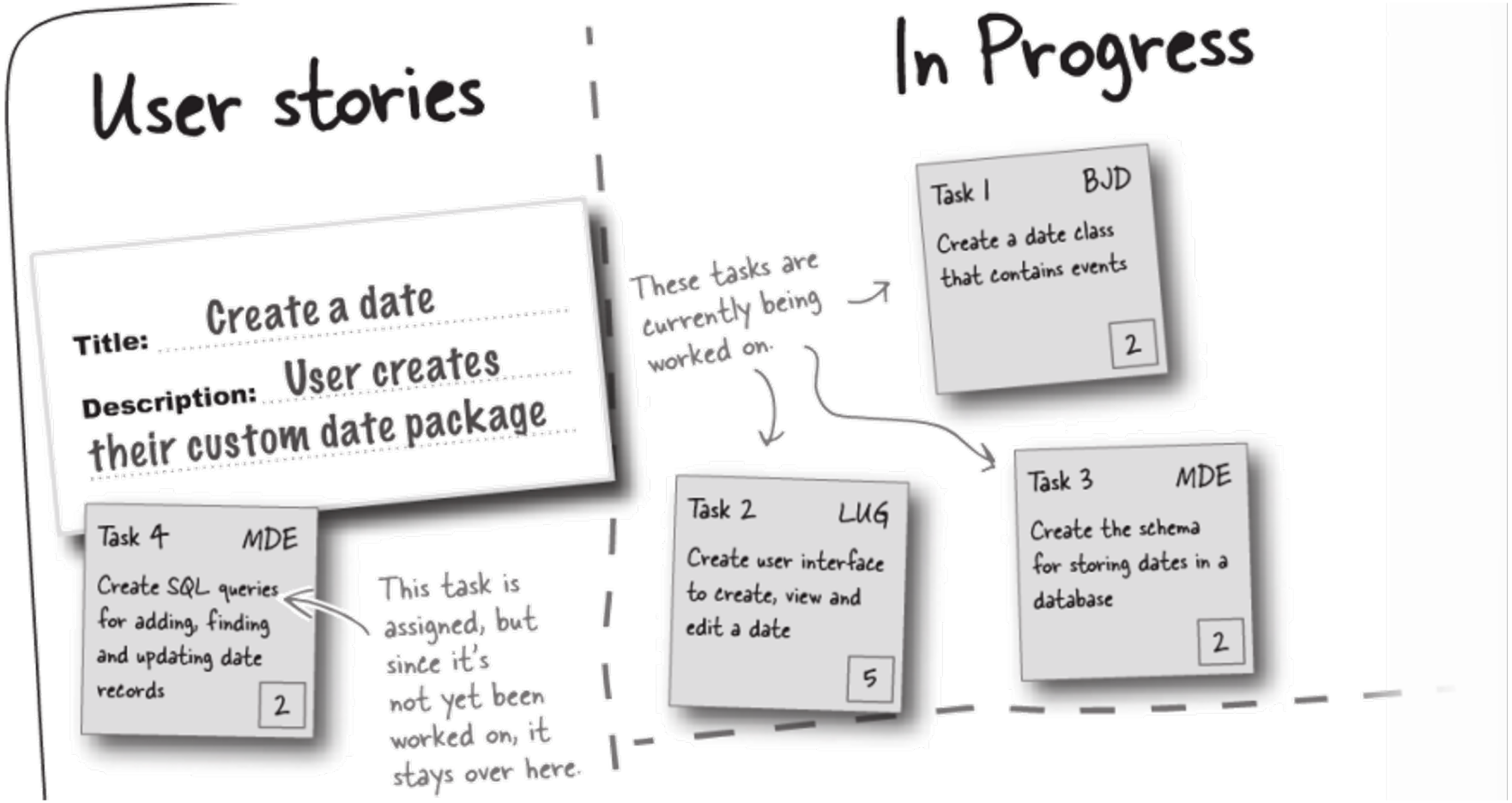


Scheduling Tasks on the Big Board

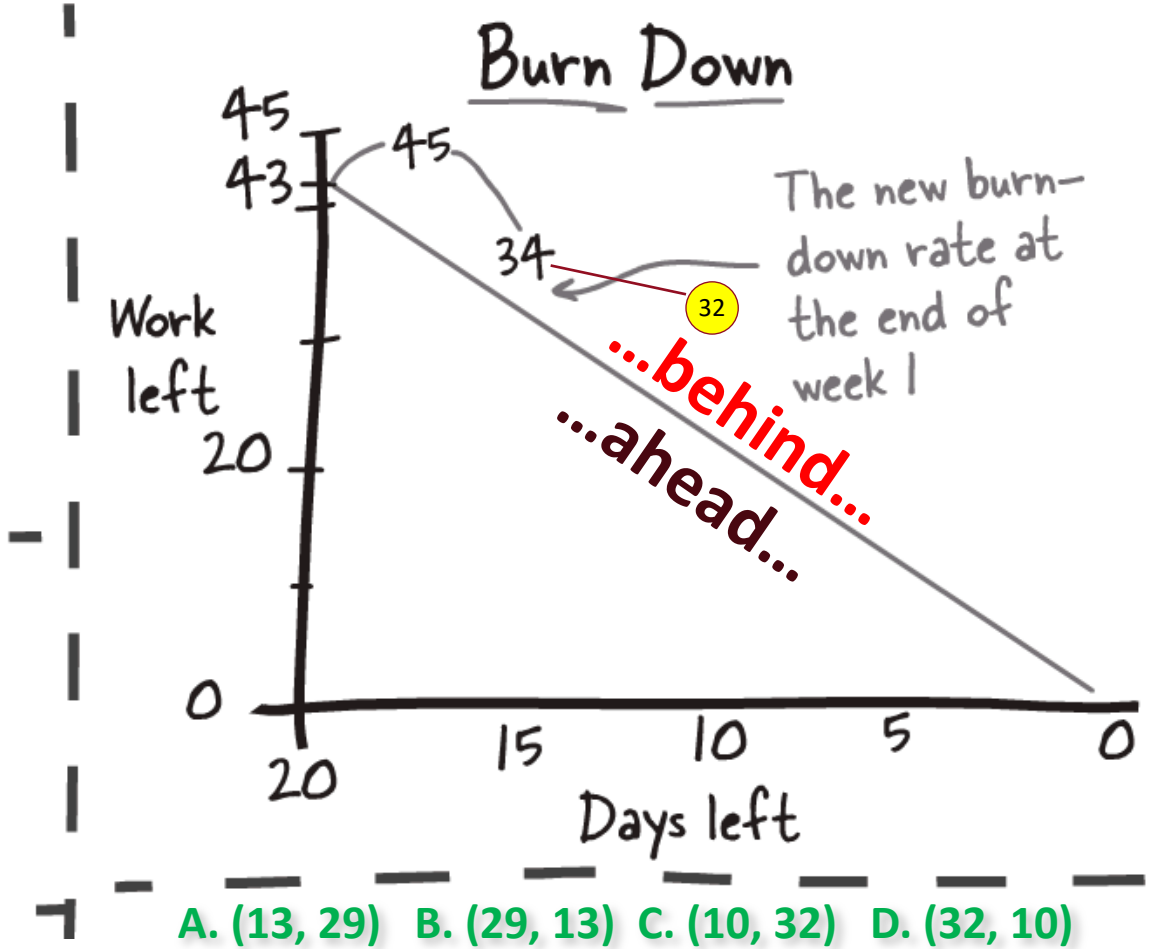
Schedule tasks in priority order (from user story priority)

Also task dependence – *if A calls B, do B first!*

Ideally system is always runnable



EXPANDED CONCEPT: BURN DOWN CHART



A. (13, 29) B. (29, 13) C. (10, 32) D. (32, 10)

With every task completion, plot a point at (days left, work left)

This task takes 5 days to finish, where put next point?

Starting at (15,34)

5 days past, 2 days of work

C. (10,32)

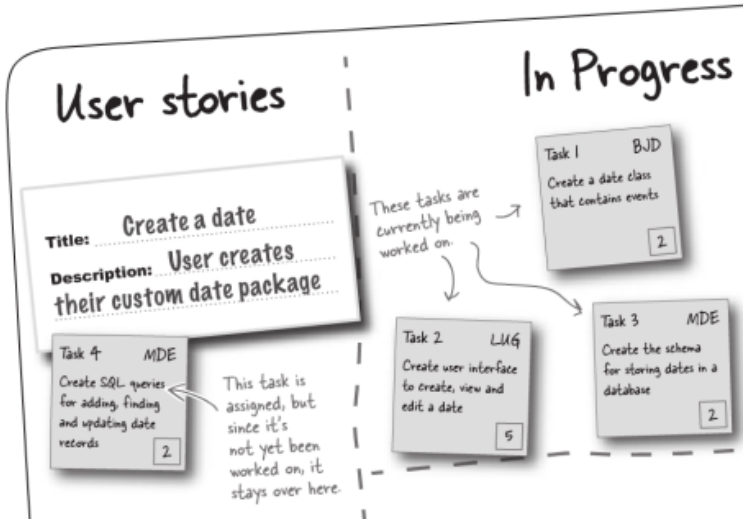
NEW CONCEPT: STAND-UP MEETING

- Key to Agile is knowing where you are, so you can make frequent, "agile" adjustments
- Big Board + Standups
- 5-15 minute daily AM meeting

1. Progress, status
2. Problems that are slowing things down

1. Update big board

Tasks and Risks



What **risks** do **tasks** address?

A. Scale – user story can take too long for one person to code up

B. Invisibility – team/self cannot "see" progress. – always "I'm ½ done."

C. Lack skill – user story may require diverse skills that a given programmer may not have all of

D. Misestimation – tasks expose tech. side (also smaller scale pieces easier to estimate)

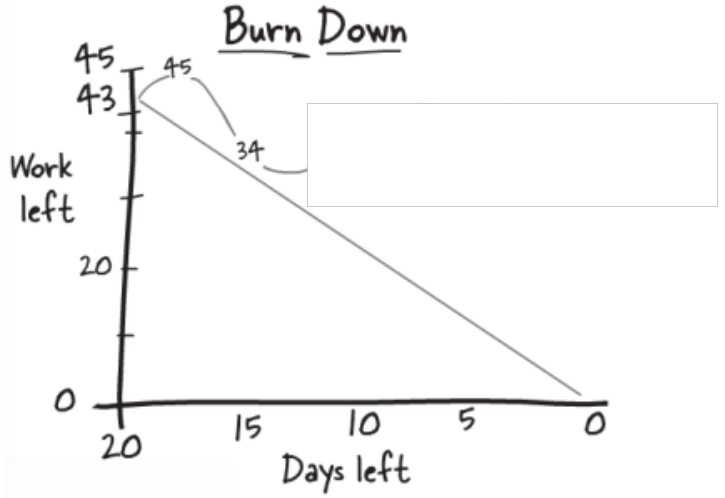
STAND UPS AND RISKS

What **risks** do **stand up meetings** address?

A. Big Board doesn't get updated
B. Miscommunication – people aren't on "same page" (e.g., component dependences)
C. Someone is stuck or going down wrong path and doesn't ask for or get help (late correction)
D. People make wrong decisions about what to work on next (kind of a same-page thing, though)

ARRIVAL TIME UNKNOWN

Your project has **fallen behind** on its iteration.
What should your team do?

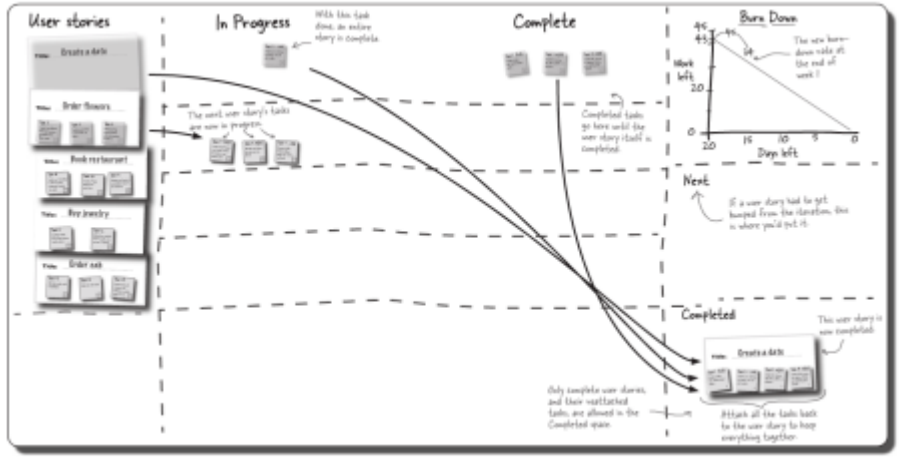


A. Look for causes – skill mismatches, bad software design, wrong velocity, underestimation, missed tasks, missed of risks

B. Assess possible consequences – late vs. drop features (e.g., hard to drop features from baseline)

C. Let the customer know, include in decision process

Big Board and Risks



What **risks** does the **big board address**?

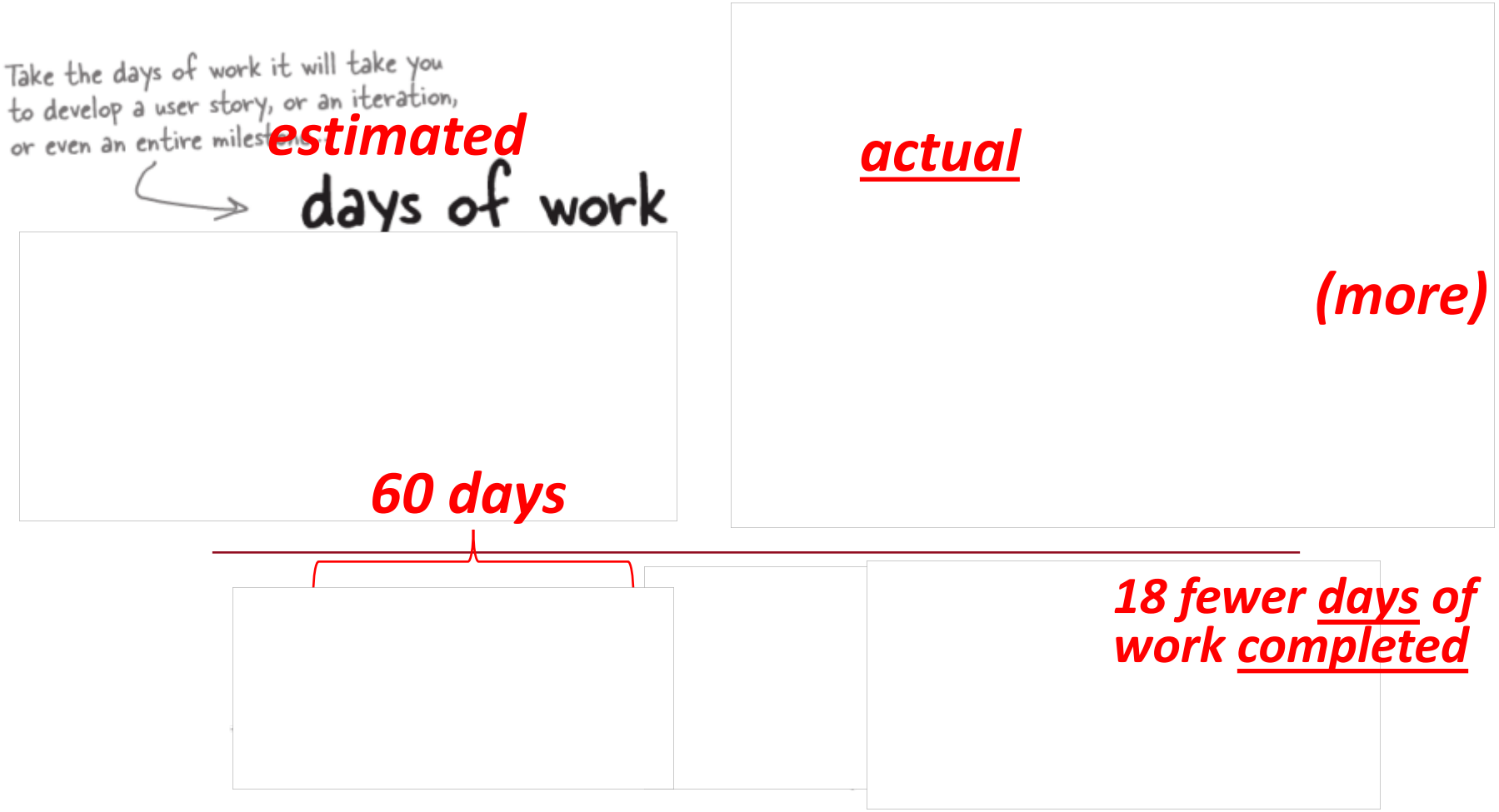
(w/o burn down chart)

A. Redundant and missed work due to miscommunication
B. Falling behind and not noticing (burn down chart)
C. Missing uneven progress (priorities and technical dependencies)

VELOCITY

Velocity is a fancy word for *efficiency*

Any sort of *likely*, recurring reason to slow work

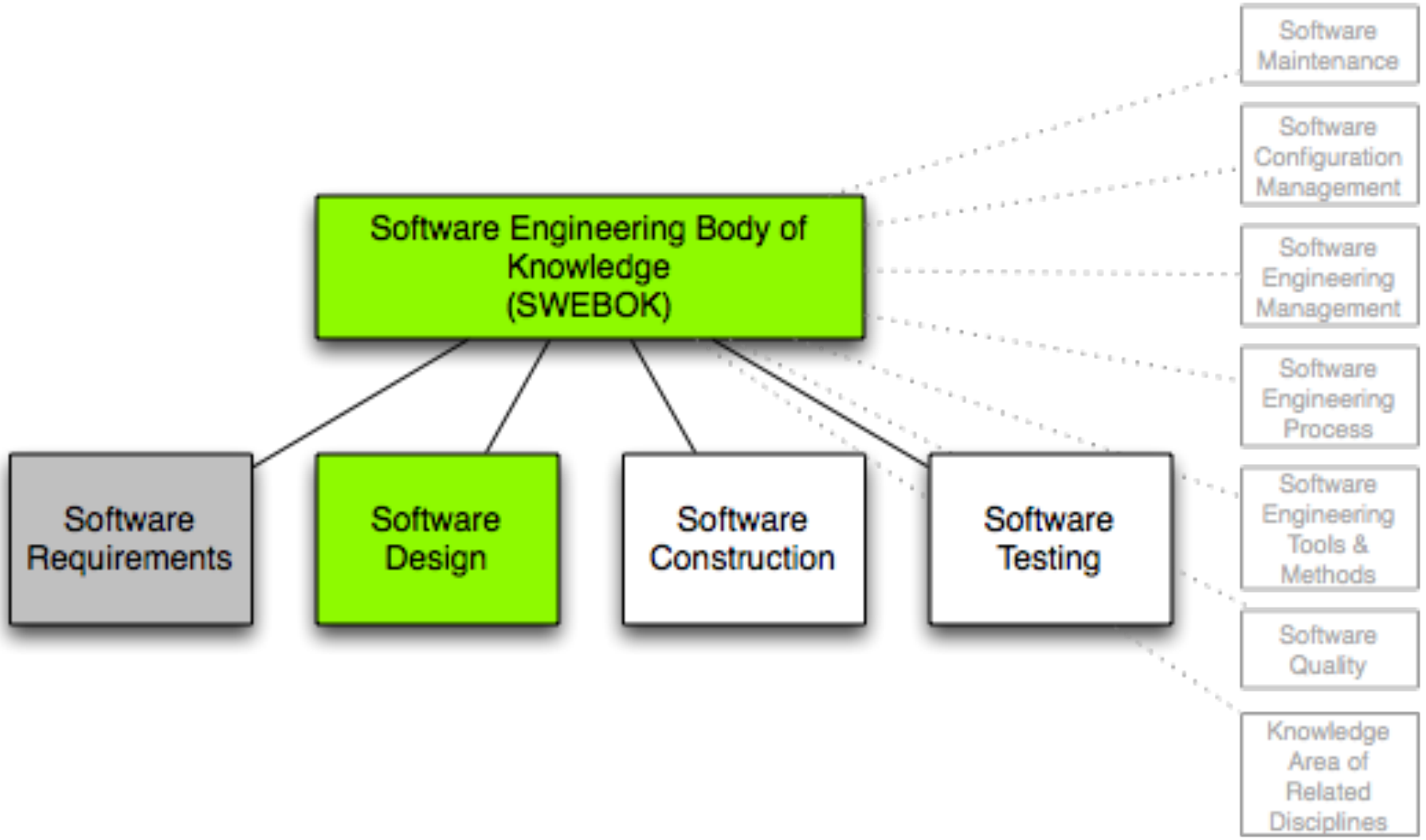


TAKE-AWAYS

- **User stories are too big and diverse** for one person to do
- **Tasks** are coherent **units of work**
  - Order them so that the **system is always runnable**
- **Big board** tells where you are at all times
  - Keeps team from getting in each other's way
- **Burn down** chart lets you know when you're falling behind
  - Talk to customer
  - Lets you make adjustments early
- It's **all about identifying problems & risks *early***

AGENDA

- Review
  - User stories & tasks
- Software Design Introduction
  - Arch. Vs. Detailed
- Use Cases & Use Case Diagrams
- Case Study


Software Engineering Body of Knowledge (SWEBOK)
Software Requirements
Software Design
Software Construction
Software Testing
Software Maintenance
Software Configuration Management
Software Engineering Management
Software Engineering Process
Software Engineering Tools & Methods
Software Quality
Knowledge Area of Related Disciplines

Software Design

- "The use of scientific principles, technical information, and imagination in the definition of a software system to perform **pre-specified functions** with maximum economy and efficiency" [GLOSSARY]

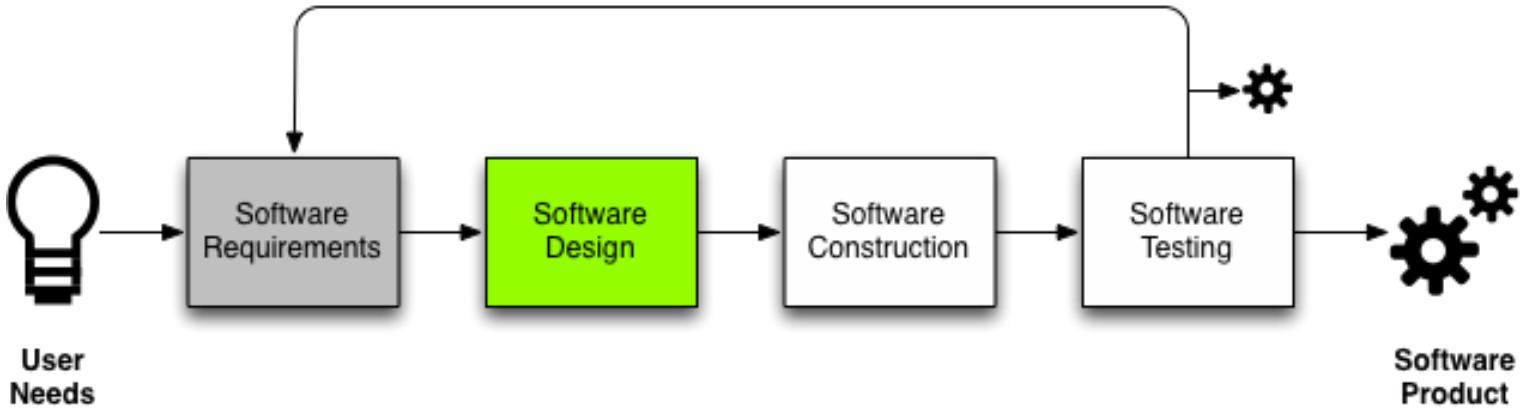
WHAT'S THE DIFFERENCE?

- Requirements:
  - "What is wanted?"

- Design:
  - "How will it be accomplished?"

Software Engineering Process

- Input
  - Requirements
- Output
  - Models and artifacts that record major design decisions



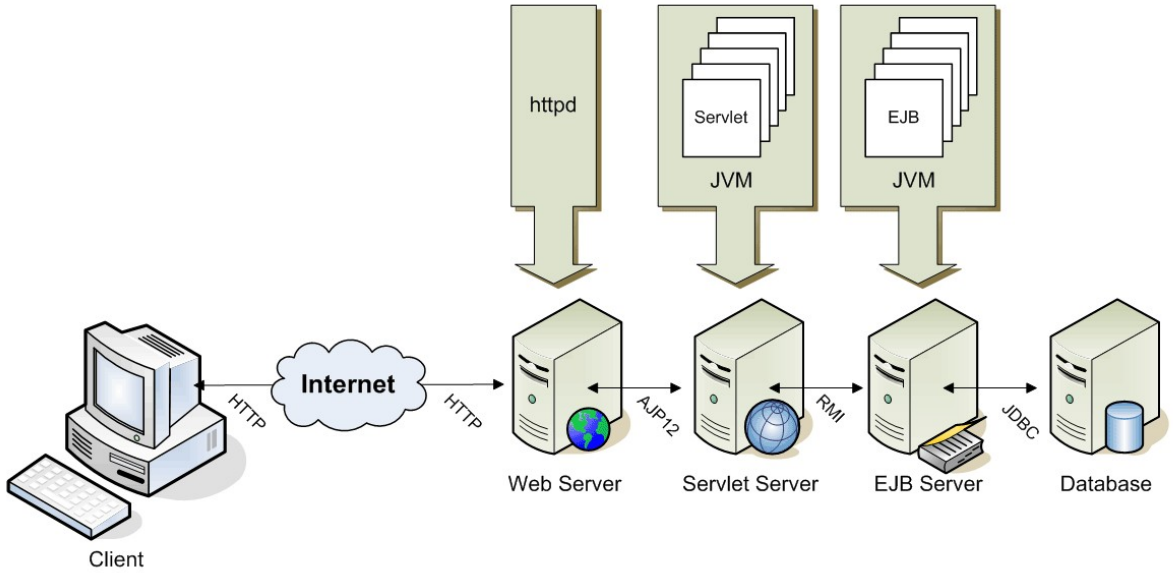
Design Two-Step

- Software Design has two steps:
  - Architectural design
  - Detailed design

- Why are there two steps to the design process?
  - Refining the design of our solution from general to specific

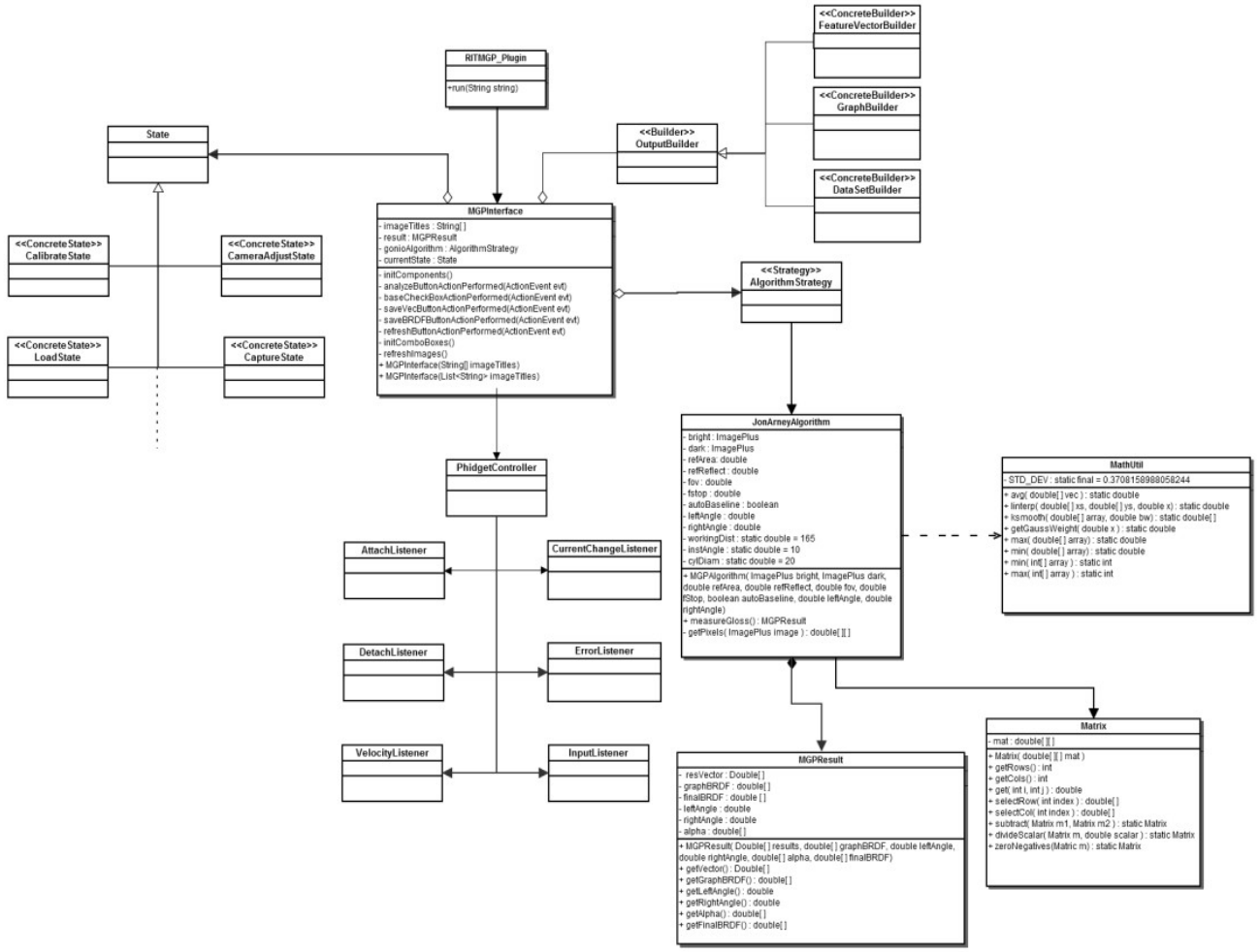
ARCHITECTURAL DESIGN

- Architectural Design
  - "the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system" [GLOSSARY]



Detailed Design

- Detailed Design
  - "the process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented" [GLOSSARY]



Architectural Vs. Detailed Design

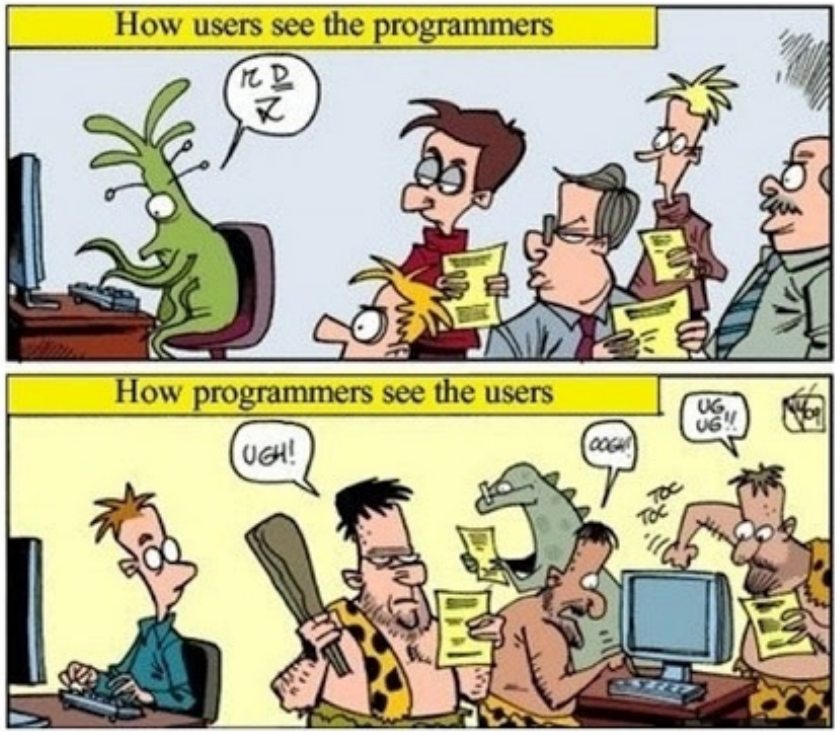
- What is more "abstract?"
  - **The end user will click a button to submit the order**
  - When the button is clicked, the button object will call the "order()" method
- What is more "concrete?"
  - **Users will be represented by the StoreUser Java class**
  - Users will all be unique

AGENDA

- Review
  - User stories & tasks
- Software Design Introduction
  - Arch. Vs. Detailed
- Use Cases & Use Case Diagrams
- Case Study

USE CASES

- The book uses the term "user stories" to describe requirements, features, and tasks
- Use Case
  - Similar to "user story"
    - Describe ONE THING that software needs to do
  - More detailed
  - **Not written by customer**



USE CASES

- Use Cases are generated through requirements elicitation
  - Used during requirements elicitation to represent external behavior

- A use case model is the set of all use cases. It is a complete description of the functionality of the system and its environment

- UML(?)

USE CASES – SOME CHARACTERISTICS

- Behavior of the system from an external point of view
- Display the high-level requirements of the system
- Actor
  - Any entity that interacts with the system (user, another system, physical environment)
- Define system boundaries–what is in the system vs. outside environment

GOALS VS. INTERACTION

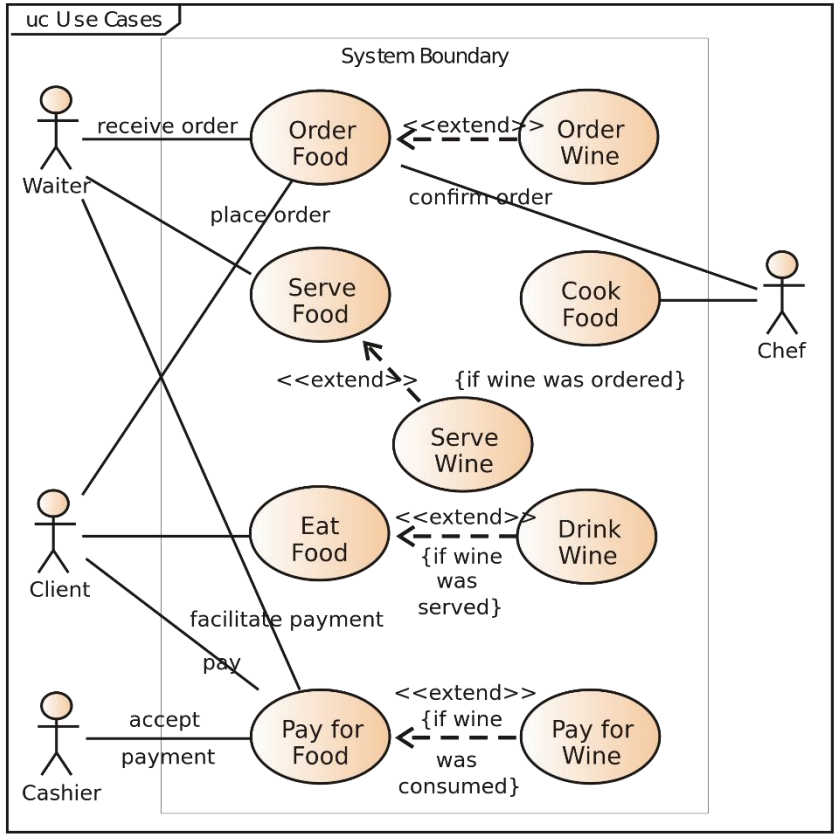
- Goals – something the user wants to achieve
  - Format a document
  - Ensure consistent formatting of two documents
- Interaction – things the user does to achieve the goal
  - Define a style
  - Change a style
  - Copy a style from one doc to the next

DEVELOPING USE CASES

- Understand what the system must do – capture the goals
- Understand how the user must interact to achieve the goals – capture user interactions
- Identify sequences of user interactions
- Start with goals and refine into interactions

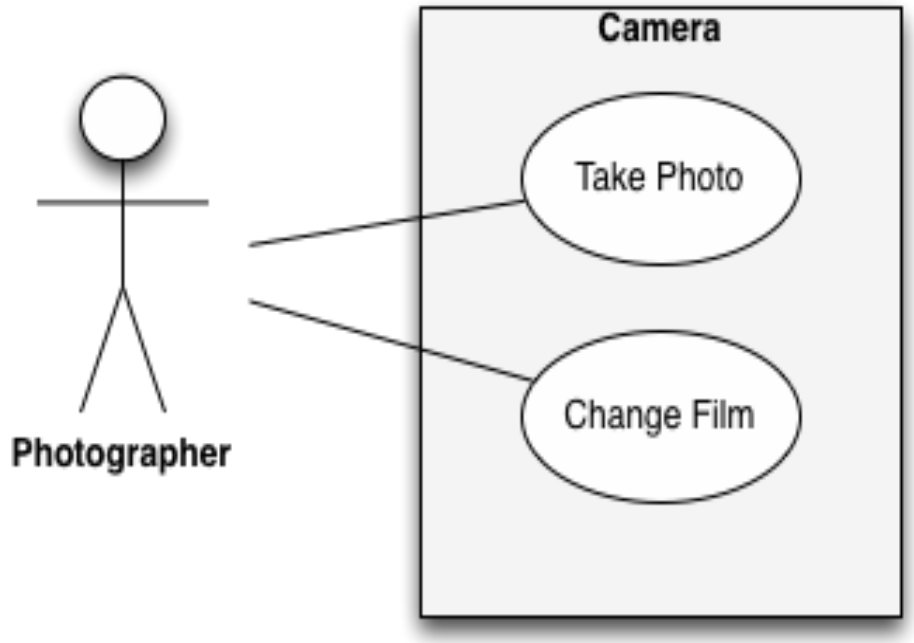
USE CASE DIAGRAM

- Use case diagrams are very high-level
  - Very abstract
- Describes a set of sequences.
- Each sequence represents the interactions of things outside the system (*actors*) with the system itself
- Use cases represent the functional requirements of the system
  - non-functional requirements must be given elsewhere
- Example use: describe all of the things that can be done with a system by all of the people who will be using the system



USE CASE DIAGRAM

- **Box**: system
- **Stick person**: actor
- **Oval**: use case
  - ONE THING that software needs to do

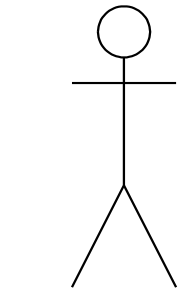


- **Arrows/Lines**: connections (interactions)
  - <uses> or <Include>
  - <extends>

CONSTRUCTING USE CASE DIAGRAMS

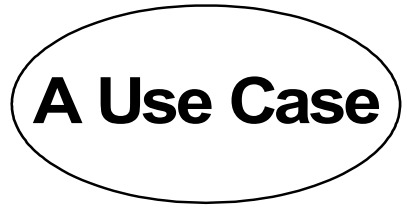
- **Actors** represent roles, that is, a type of user of the system
- **Use cases** represent a sequence of interaction for a type of functionality

ACTORS



- An actor models an *external entity* that communicates with the system:
  - User
  - External system
  - Physical environment
- An actor has a unique name and an optional description
- An actor is a set of roles that users of use cases play when interacting with the system
- Examples: Customer, Loan officer

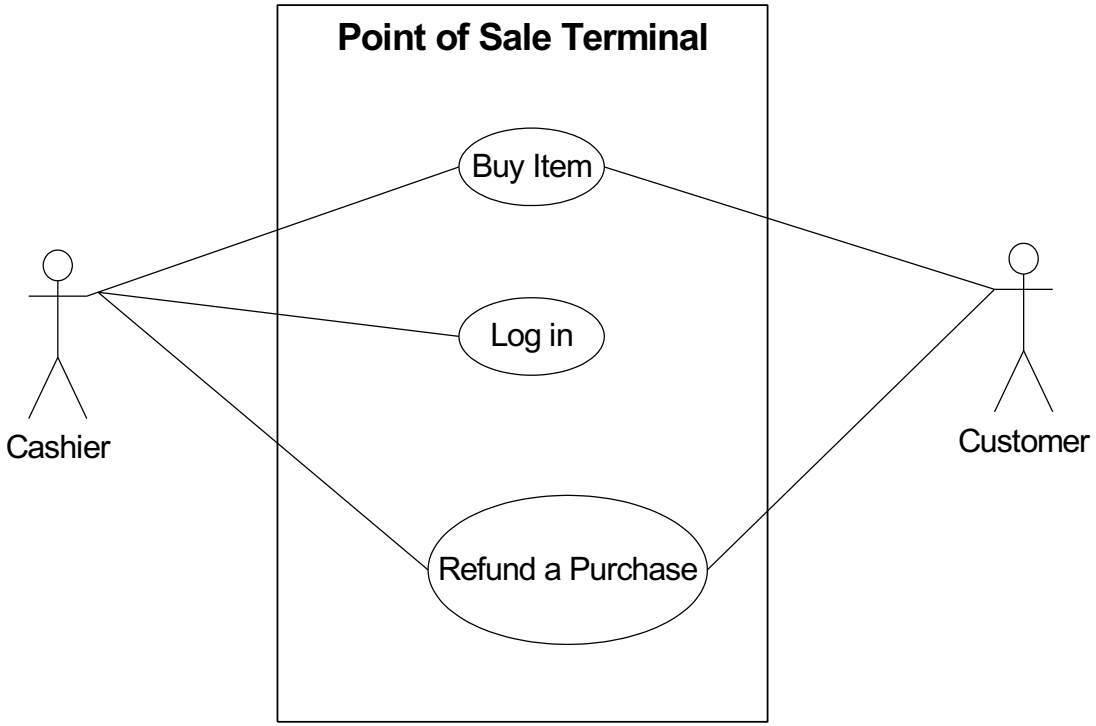
What is a Use Case?



- A use case represents a class of functionality provided by the system as an event flow.
- A use case consists of: Unique name, Participating actors, Entry conditions, Flow of events, Exit conditions, and Special requirements
- Each use case achieves a discrete *goal* for the user
- Describes *what a system does* **but not** how it does it.
- Examples: withdraw money, process loan

EXAMPLE

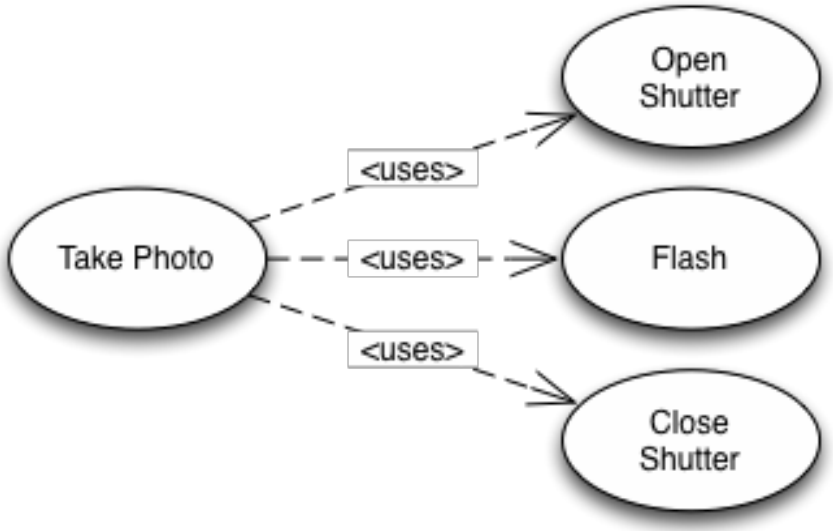
- **Actors**: Customer (initiator), Cashier
- **Type**: Primary
- **Description**: The costumer arrives at the checkout with items to purchase. Cashier records purchases and collects payment. Customer leaves with items



Refining Use Cases

- Separate internal and external issues
- Describe flow of events in text, clearly enough for customer to understand
  - Main flow of events
  - Exceptional flow of events
- Show common behaviors with *includes*
- Describe extensions and exceptions with *extends*

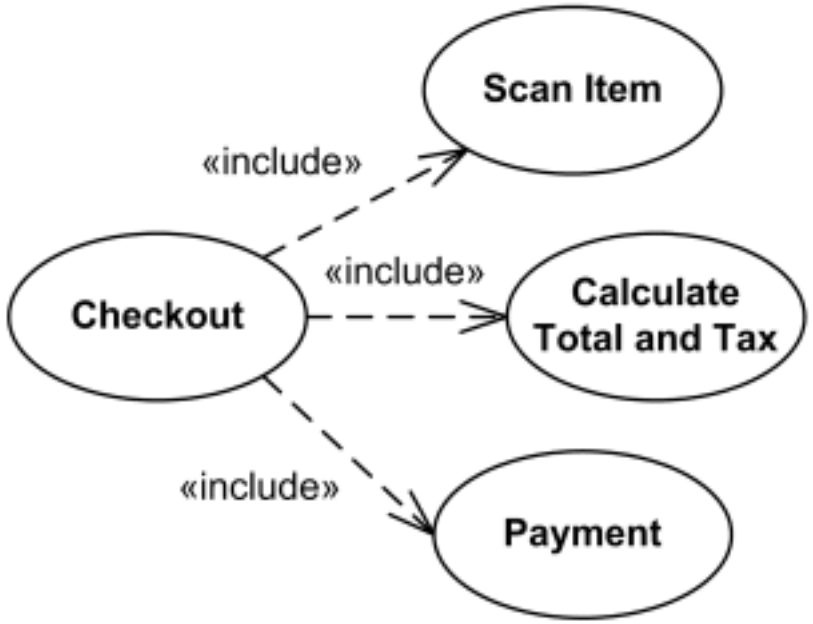
"USES" CONNECTION

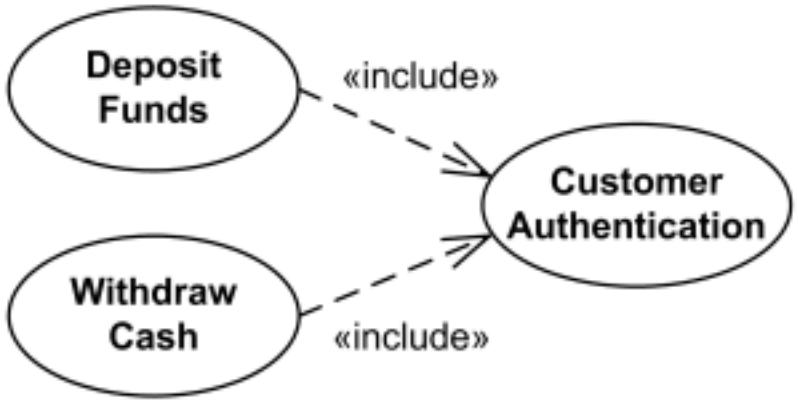


- <uses> arrow
  - The process of doing X involves Y

- **<<Uses>> == <<include>>**
- An <include> relationship represents behavior that is factored out of the use case.
- An <include> relationship represents behavior that is factored out for reuse, not because it is an exception.

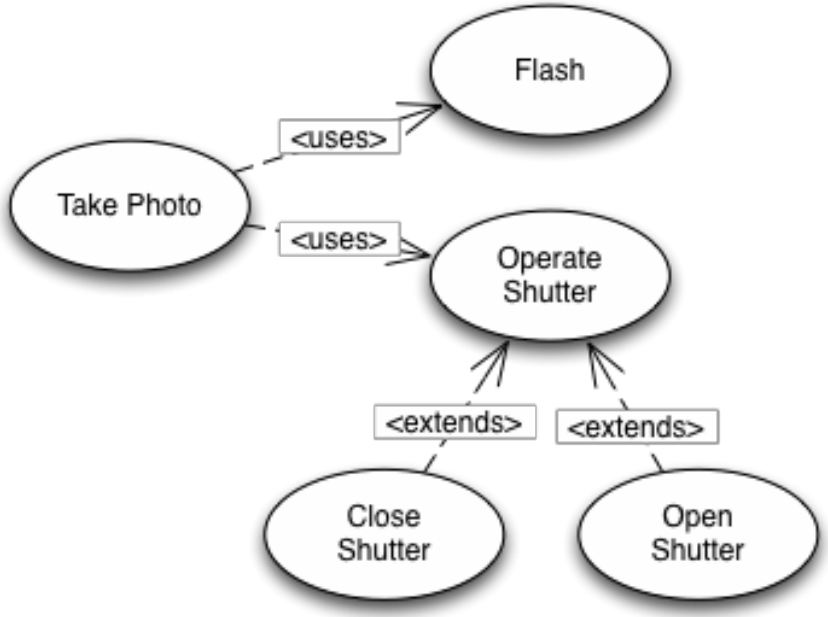
EXAMPLES


Scan Item
«include»
Checkout
«include»
Calculate Total and Tax
«include»
Payment


Deposit Funds
«include»
Customer Authentication
Withdraw Cash
«include»

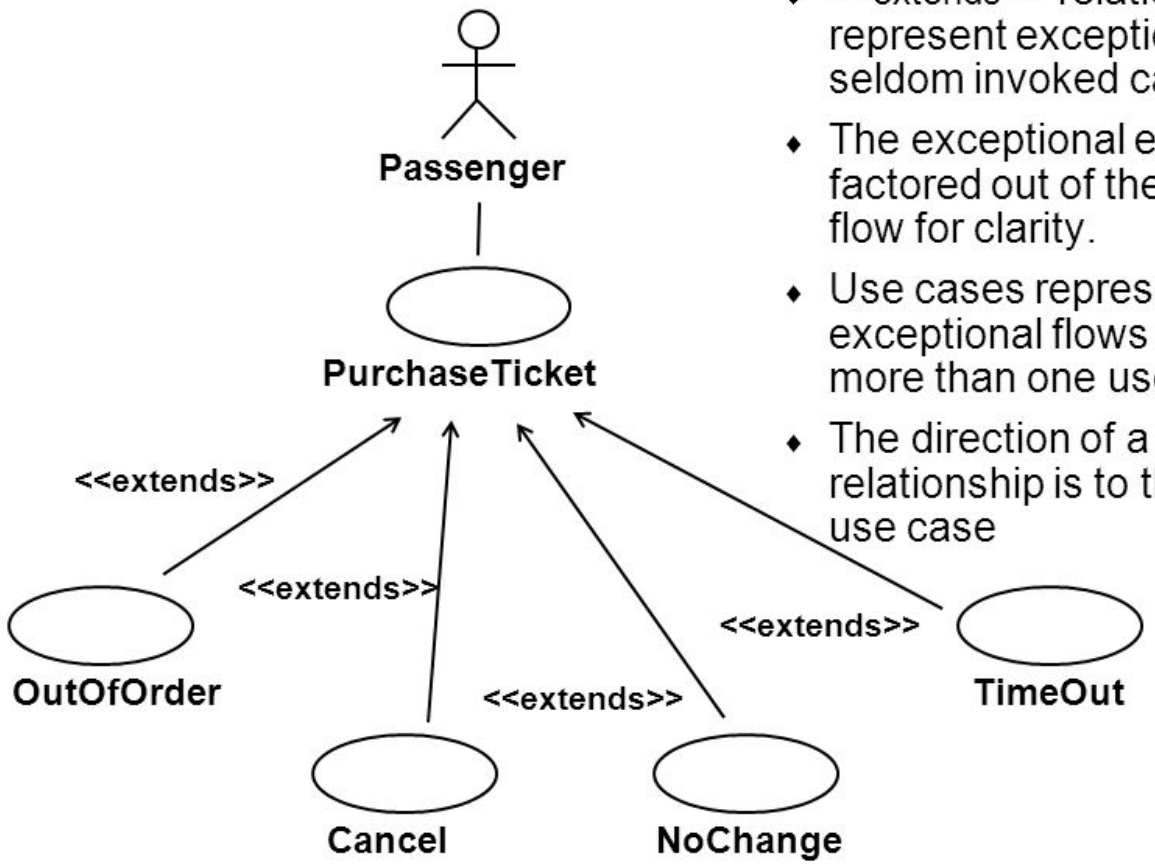
"EXTENDS" CONNECTION

- <extends> arrow
  - x is a special case of y



- <extend> relationships represent exceptional or seldom invoked cases
  - The exceptional event flows are factored out of the main event flow for clarity.
  - Use cases representing exceptional flows can extend more than one use case.

The <<extends>> *Relationship*



- <<extends>> relationships represent exceptional or seldom invoked cases.
- The exceptional event flows are factored out of the main event flow for clarity.
- Use cases representing exceptional flows can extend more than one use case.
- The direction of a <<extends>> relationship is to the extended use case

USES VS. EXTENDS

- X uses Y
  - X is a subtask of Y
  - An <include> use case is always executed by its parent –the functionality is just factored out so it can be re-used by multiple use cases.
  - The direction of a <include> relationship is to the using use case.
- X extends Y
  - X is a special case of Y
  - An <extend> use case is executed under special circumstances only (like when an error occurs)
  - The direction of a <extend> relationship is to the extended use case.

Types of Diagrams

- ***Structural Diagrams*** – focus on static aspects of the software system
  - Class, Object, Component, Deployment

- ***Behavioral Diagrams*** – focus on dynamic aspects of the software system
  - Use-case, Interaction, State Chart, Activity

STRUCTURAL DIAGRAMS

- **Class Diagram** – set of classes and their relationships.
  - Describes interface to the class (set of operations describing services)
- **Object Diagram** – set of objects (class instances) and their relationships
- **Component Diagram** – logical groupings of elements and their relationships
- **Deployment Diagram** - set of computational resources (nodes) that host each component.

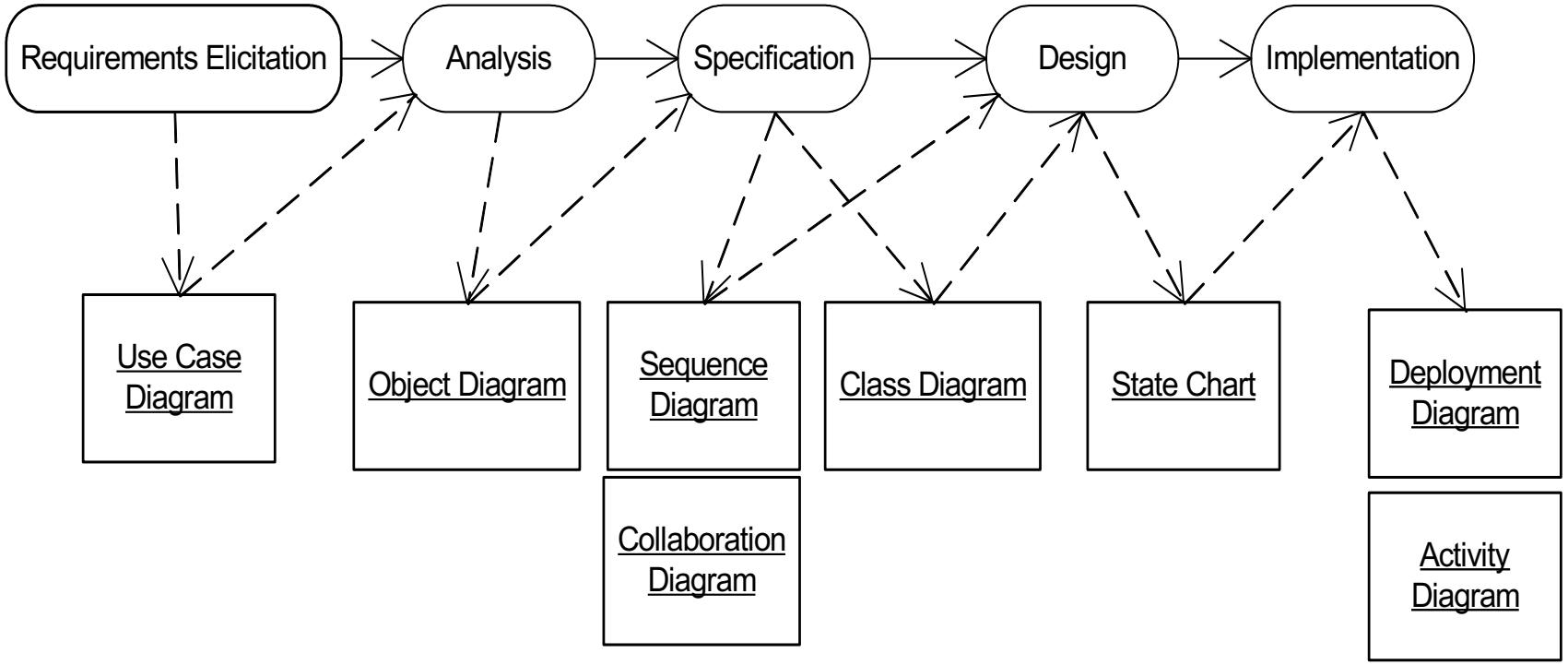
BEHAVIORAL DIAGRAM

- **Use Case Diagram** – high-level behaviors of the system, user goals, external entities: actors
- **Sequence Diagram** – focus on time ordering of messages
- **Collaboration Diagram** – focus on structural organization of objects and messages
- **State Chart Diagram** – event driven state changes of system
- **Activity Diagram** – flow of control between activities

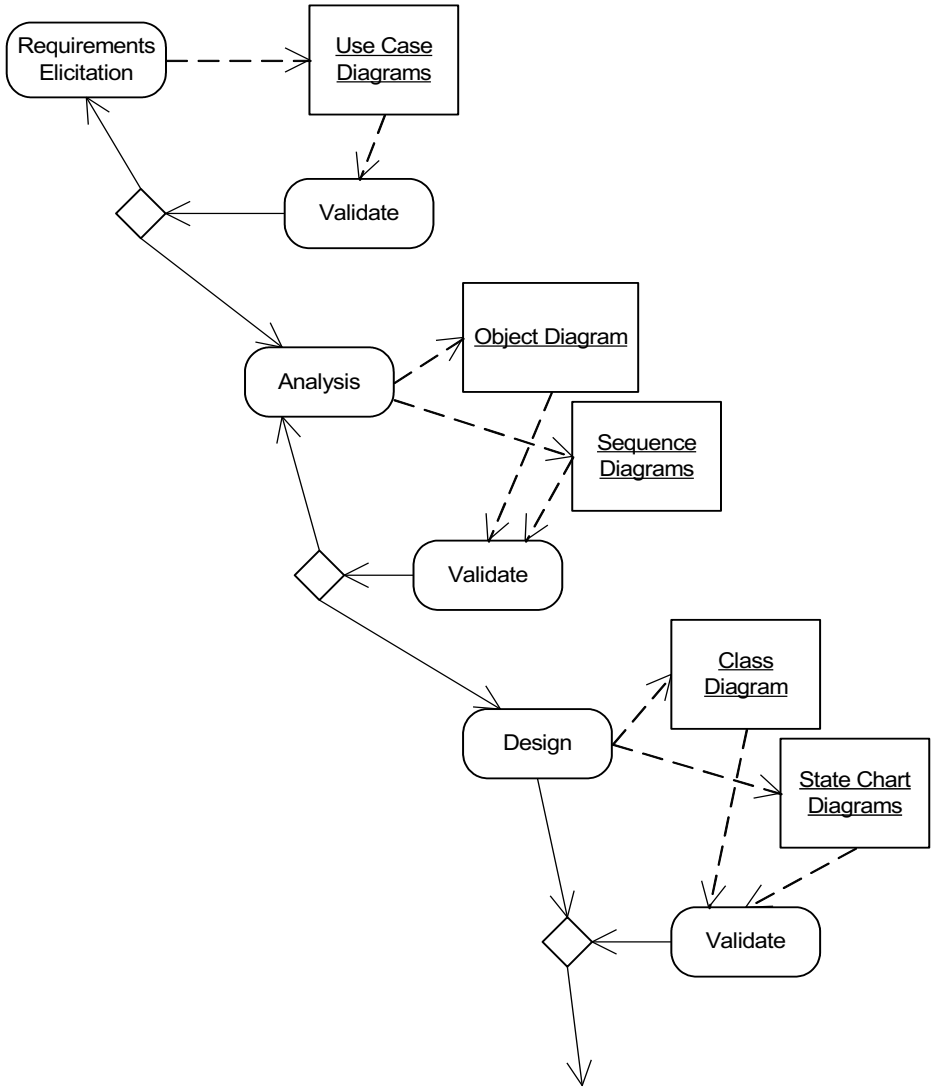
Analysis & Design Process

- Requirements elicitation – High level capture of user/system requirements
  - Use Case Diagram
- Identify major objects and relationships
  - Object and class diagrams
- Create scenarios of usage
  - Class, Sequence and Collaboration diagrams
- Generalize scenarios to describe behavior
  - Class, State and Activity Diagrams
- Refine and add implementation details
  - Component and Deployment Diagrams

UML Driven Process



UML Driven Process



AGENDA

- Review
  - User stories & tasks
- Software Design Introduction
  - Arch. Vs. Detailed
- Use Cases & Use Case Diagrams
- Case Study

CASE STUDY

- There is a need to develop software for tracking library loan records. Library patrons may borrow books, magazines, compact discs, and audio tapes. A library must manage each copy of a library item. For example, a library may have five copies of the book "The Grapes of Wrath". Each borrower has a library card.
- Each type of library item has a standard checkout period and maximum number of renewals. For example, children's books may be checked out for a month, while adult books may be checked out for only two weeks. Ordinary books can be renewed once; books with a pending request may not be renewed. The system must record the actual return date and any fine that was paid.

Retrospective Questions

- What is Software Design (input and output)?
- Arch. Vs. Detailed Design?
- What is a use case diagram?
- What are the 4 elements of a use case?

I POSTED A USE CASE "EXERCISES" ON CANVAS

- Canvas > Modules > Week-05 > **Use Case Diagram Exercises**

Next Class

- UML Class Diagrams