Informatics 43

LECTURE 4

“HOW DO WE KNOW WHAT TO BUILD (PART 2)?”

Today’s Lecture – **How do we know what to build?**

- Requirements: Analysis and Specification
  - defines “what” the system should do without saying “how” it should do it
  - serves as the basis for all future development
  - should be done right to prevent costly changes later
  - has a specific structure
- Use Cases
  - What and why
  - How
  - Use case diagrams
- Quiz 2 Study Guide
- Homework 1

Definition

Requirements =
what the software should do (without saying **how** it should do it)

Note: requirements address what a customer <u>needs</u>, not what a customer wants

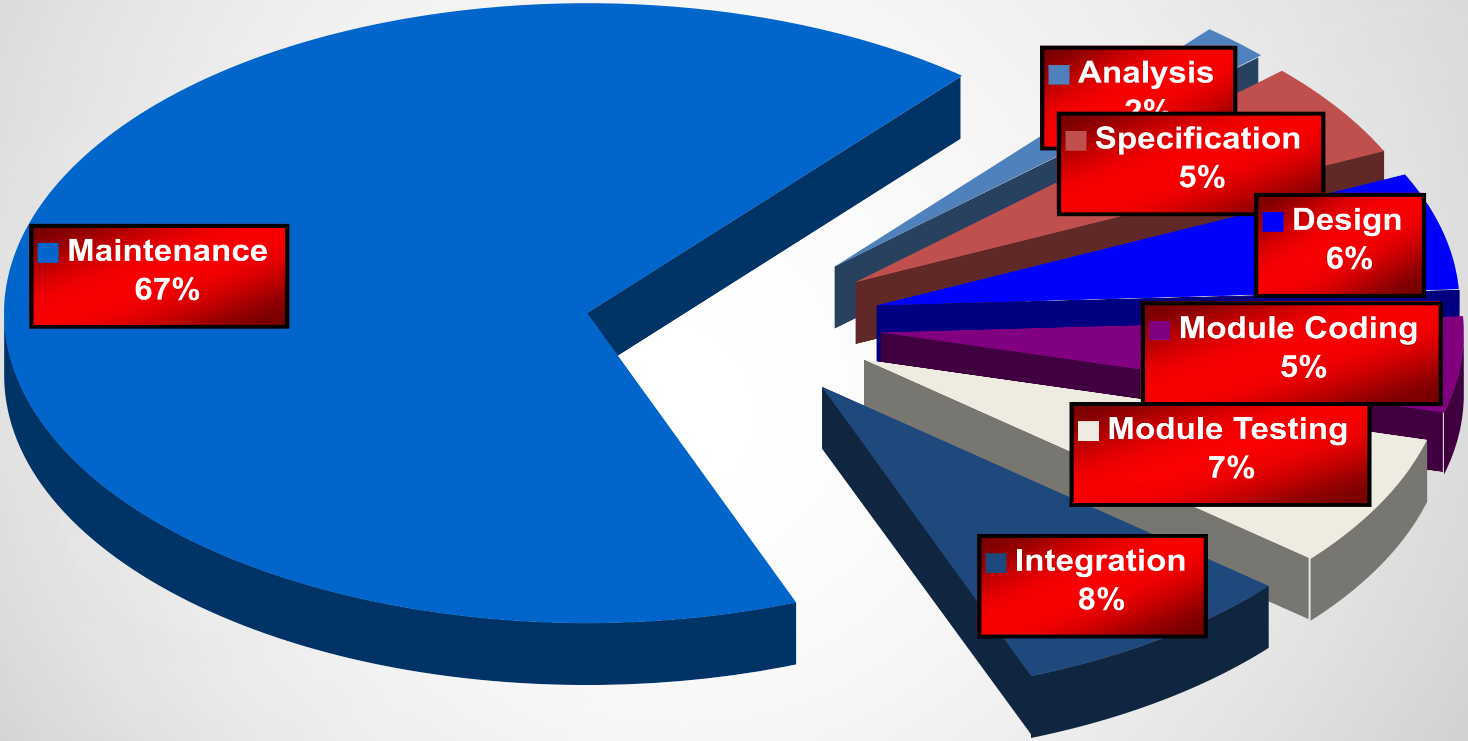
Why Requirements?

- “[We] have grown to care about requirements because we have seen more projects stumble or fail as a result of poor requirements than for any other reason”
  - (Kulak and Guiney, in “Use Cases: Requirements in Context”)
- Studies show that many of the key contributors to project failures originate or relate to requirements
  - (The Standish Group CHAOS reports)

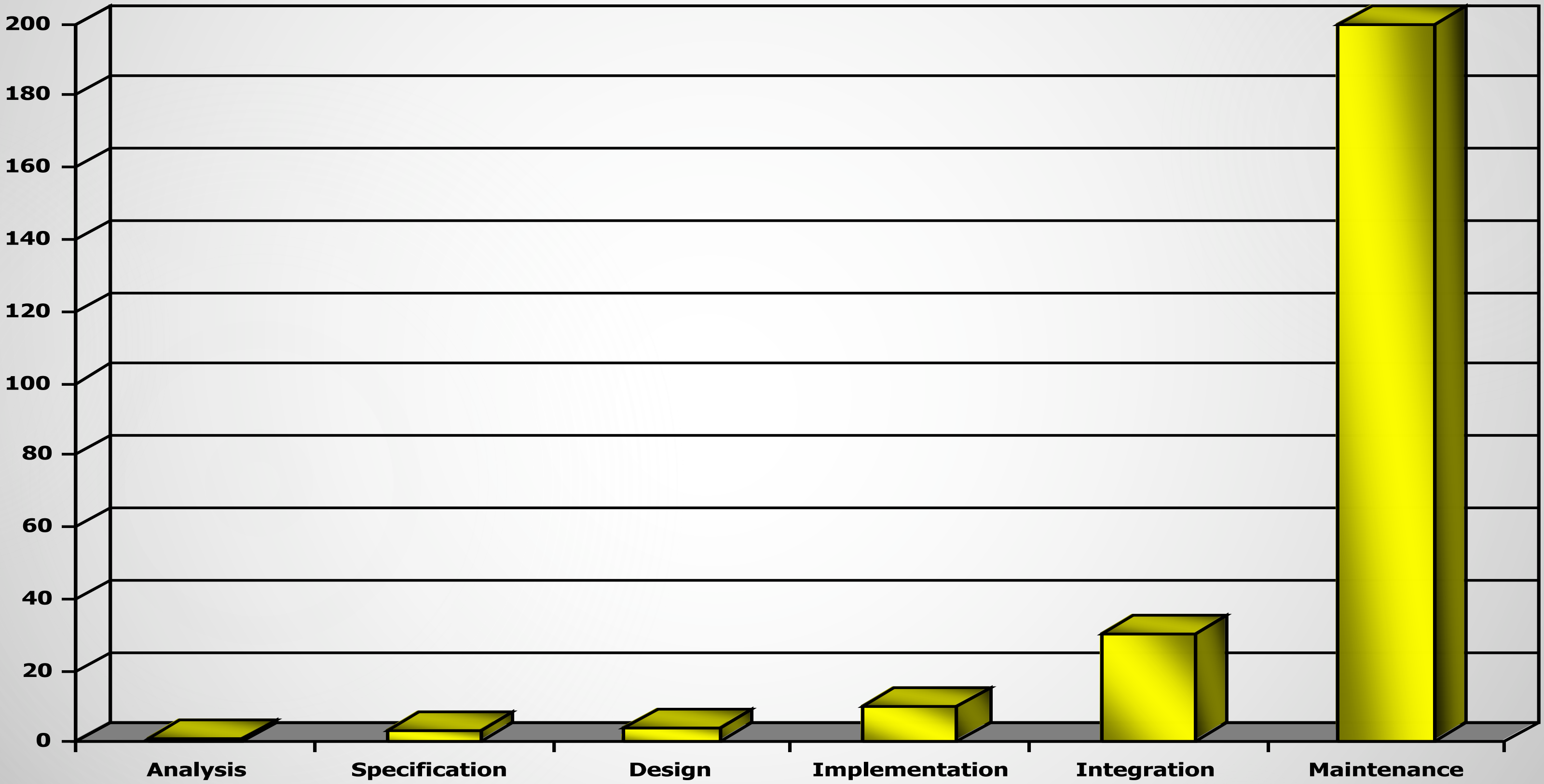
More stats…

- From CHAOS reports: Requirements defects are expensive
  - They represent more than 70% of rework costs
  - Rework consumes about 30-50% of total project budget
  - Lack of user input/user involvement listed as most frequent problem

More Stats: Software Life Cycle Costs



More Stats: Cost of Change Progressively Higher



Today’s Lecture – **How do we know what to build?**

- Software failures
- Why requirements?
- Requirements engineering
  - Requirements phase
  - Requirements analysis
  - Requirements specification (documentation)

Which category employs the most computer programmers in the U. S.?

A. In-house staff writing systems for internal use

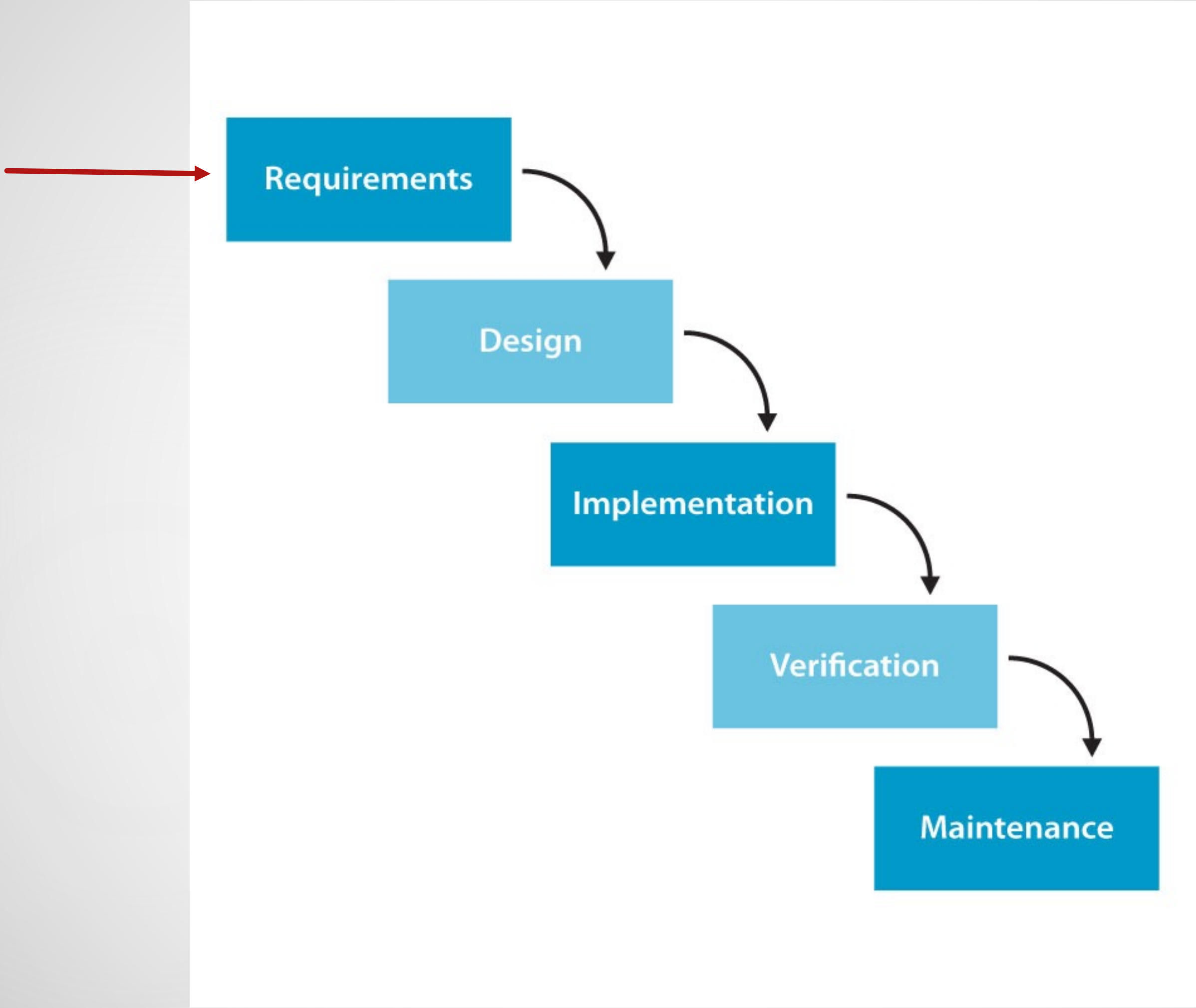
B. Games, apps, productivity software

C. Consulting companies/contract programming

D. Open source projects

E. Creators of viruses and other malware

Waterfall



Requirements

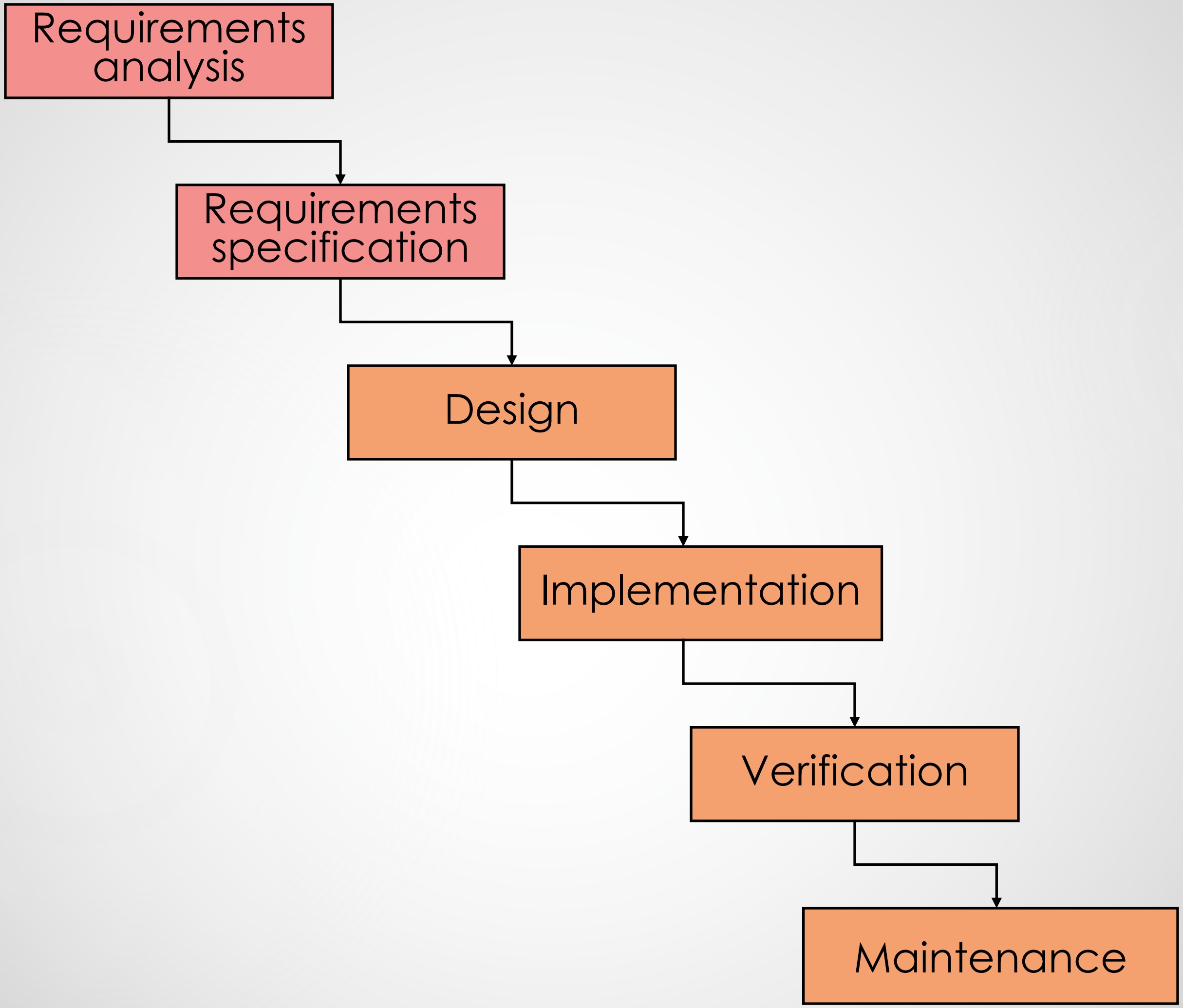
Design

Implementation

Verification

Maintenance

Waterfall


Requirements analysis
Requirements specification
Design
Implementation
Verification
Maintenance

Requirements Phase - Terminology

- Requirements analysis
  - <u>Activity</u> of discovering/observing/gathering customer’s needs
- Requirements specification
  - <u>Activity</u> of describing/documenting customer’s needs
  - Can also refer to the requirements document

Today’s Lecture – **How do we know what to build?**

- Software failures
- Why requirements?
- Requirements engineering
  - Requirements phase
  - Requirements analysis
  - Requirements specification (documentation)

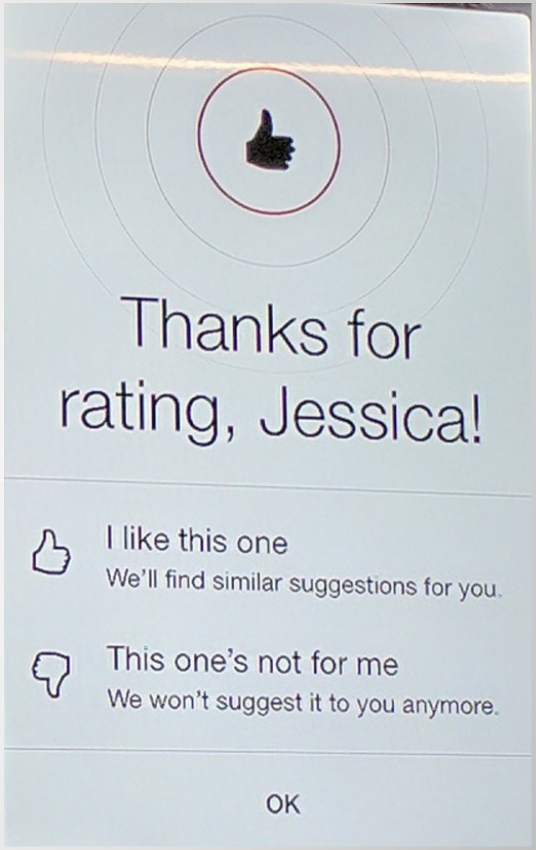
Techniques for Requirements Analysis

- Interview customer
- Observe customer (CNBC article)
- Create use cases/scenarios/user stories
- Prototype solutions
- Identify important objects/roles/functions/goals
- Perform research
- Construct models
- …
- Data (see next slide)

https://www.youtube.com/watch?v=I_GTTyE9i9Y

Data-driven requirements analysis

- Usage data is analyzed and business metrics are recorded to discover the value of new (potential) features
  - Techniques
    - Data analytics
    - A/B testing
    - Prototyping
    - Market research
  - This information is used to create/prioritize/analyze/evaluate requirements



Thanks for rating, Jessica!

I like this one
We'll find similar suggestions for you.

This one's not for me
We won't suggest it to you anymore.

Requirements Specification

- Serves as the fundamental reference point between customer and software producer
- Defines
  - the "what," not the "how"
  - environmental requirements on the software
  - constraints on the software
  - software qualities

Why Spend a Lot of Time?

- A requirements specification is *the* source for all future steps in the software life cycle
- Changes are cheaper now than later

Users of a Requirements Document

- Customers
- Developers
- Managers
- Testers
- Documenters
- Maintainers
- System engineers
- Software architects
- Marketing personnel
- …

Document Structure

- Introduction
- Executive summary
- Application context
- Environmental requirements
- Functional requirements
- Software qualities
- Other requirements
- Time schedule
- Potential risks
- Assumptions
- Future changes
- Glossary
- Reference documents

Introduction

- What is this document about?
- Who was it created for?
- Who created it?
- Outline

Executive Summary

- Short, succinct, concise, to-the-point, description of the product
  - No more than one page
- Identifies
  - main goals
  - key features
  - key risks/obstacles

Application Context

- Describes the situation in which the software will be used
    - Home, office, inside, outside, …
- Identifies all things that the system affects
    - Objects, processes, other software, hardware, and people

Environmental Requirements

- Platforms
  - Hardware
    - Operating systems, types of machines, memory size, hard disk space
  - Software
    - Is it a Web app? Mobile app? Desktop app?
    - Is it open source? Linux? Apache? PHP/MySQL?
    - Is it enterprise software? .Net? Enterprise Java, J2EE?
- Programming language(s)
- Standards

Functional Requirements

- Identifies all concepts, functions, features, and information that the system provides to its users
- Provides an abstraction for each of those, characterizing the properties and functions that are relevant to the user
  - What is the system supposed to do?
  - What information does the system need?
  - What is supposed to happen when something goes wrong?

Desired Software “ilities” (Qualities), or Non-functional Requirements

- Correctness
- Reliability
- Efficiency
- Usability
- Maintainability
- Portability
- Reusability
- Interoperability
- Robustness
- Security
- Scalability
- …

This section helps developers assess tradeoffs in the system’s implementation

Other Requirements

- What about cost?
- What about documentation?
- What about manuals?
- What about tutorials?
- What about on-the-job training?
- What about requirements that do not fit in any of the previous categories?

Time Schedule

- By when should all of this be done?
  - Initial delivery date
  - Acceptance period
  - Final delivery date
- What are some important milestones to be reached?
  - Prototype delivered
  - Architecture/design/implementation/testing completed
  - Sprint/increment 1/2/3 etc. completed

Potential Risks

- Risks: “future uncertain events with a probability of occurrence and a potential for loss” (softwaretestinghelp.com)
- Any project faces risks
  - new methodology
  - requirements new to the group
  - special skills and resource shortage
  - aggressive schedule
  - tight funding
  - security
  - ethical
- It is important to identify those risks *up-front* so you and the customer are aware of them

Assumptions

- Factors that are believed to be true during the life cycle of the project
- If changed, they may affect the project outcomes negatively
- Examples
  - end-user characteristics
  - known technology infrastructure
  - resource availability
  - funding availability

Future Changes

- Any project faces changes over time
  - It is important to identify those changes *up-front* so you and the customer are aware of them
- Structure the requirements document in such a way that it easily absorbs changes
  - Define concepts once
  - Partition separate concerns
  - Avoid redundancy

Glossary

- Precise definitions of terms used throughout the requirements document

Reference Documents

- Pointers to existing processes and tools used within an organization
- Pointers to other, existing software that provides similar functionality
- Pointers to literature

User Interface

- An approximate UI is normally part of a requirements specification

Observations (about the specifications document)

- Document is structured to address the fundamental principles
  - Rigor
  - Separation of concerns
    - Modularity
    - Abstraction
  - Anticipation of change
  - Generality
  - Incrementality
- Not every project requires every section of the document

These principles apply to all aspects of software engineering!

Specification Methods

- Natural language
  - SRS, user stories
- Diagrams
  - Data flow, finite state machines, Petri nets
- Formal language
  - Z, TLA+
- Models
  - Domain model (objects), usage model (use cases), goal model

Verification

- Is the requirements specification *complete?*
- Is each of the requirements *understandable?*
- Is each of the requirements *unambiguous?*
- Are any of the requirements *in conflict?*
- Can each of the requirements be *verified?*
- Are are all terms and concepts *defined?*
- Is the requirements specification *unbiased?*

Acceptance Test Plan

- Often accompanies a requirements specification
- Specifies how it will be determined that each requirement is met
- Binds a customer to accept the delivered system if it passes all the tests
- May include:
  - some specific test cases
  - the number of test cases that must pass

Real Requirements Documents

- https://www.joelonsoftware.com/whattimeisit/
- https://www.slideshare.net/indrisrozas/example-requirements-specification
- http://frost.ics.uci.edu/GameDesign/ClawDesignDoc.pdf

Ziv's Law

- Software development is unpredictable and the documented artifacts such as requirements will never be fully understood
- Uncertainty is inherent and inevitable in SE processes and products!

Today's Lecture – **How do we know what to build?**

- A requirements specification
  - defines "what" the system should do without saying "how" it should do it
  - serves as the basis for all future development
  - should be done right to prevent costly changes later
  - has a specific structure
- Use Cases
  - What and why
  - How
  - Use case diagrams
- Quiz 2 Study Guide
- Homework 1

What is a Use Case?

- A textual description of a set of actions defining interactions between an actor and the system to achieve a goal
- Includes
  - Basic functionality/goal
  - Any precondition
  - Flow of events
  - Any postcondition
  - Any error condition and/or alternative flow
- Use cases go hand-in-hand with requirements

Flows

- Flow = a sequence of steps describing an interaction between a “user” and a “system”
- A use case describes a set of flows that together accomplish a specific user “goal”

Types of Flows

- Basic Flow: "Happy day" scenario
- Alternative Flow: The goal is achieved, but in an alternate way
- Exception Flow: The goal is not achieved

A use case should capture all possible flows—successful and unsuccessful ones

Why Use Cases?

Other Requirements Engineering Methods	Use Cases
May not map well to design/code	Map well to design and implementation constructs
May not translate well to acceptance tests	Make it easy to verify/validate a design and implementation against user goals
Can be difficult for non-experts to understand	Framed in terms of user goals and flows of events, user requests and system responses

Why Not Use Cases?

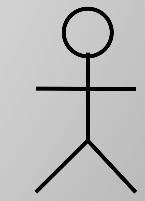
- Use cases are not good for specifying
  - User interfaces
  - Data formats
  - Business rules
  - Non-functional requirements

Produce use cases in conjunction with other requirements engineering methods

Actors

- Represent **external** entities that interact with the system
  - Human
  - Hardware
  - Another software system
- A use case is initiated by an actor (**primary actor**) to invoke a certain functionality in the system
- A use case is a dialogue between actors and the system

Symbolic representation of an actor within a use case diagram



Identifying Actors (I)

- Actors are discovered
  - by reading system documents
  - by talking with customers and domain experts
- Useful questions for identifying actors:
  - Who uses the system?
  - Who installs the system?
  - Who starts up the system?
  - Who shuts down the system?
  - What other devices and external systems work directly with the system?

Identifying Actors (II)

- Additional questions for identifying actors are:
  - Who gets information from this system?
  - Who provides information to the system?
  - Does anything happen automatically at a preset time?
  - Who is interested in a certain requirement?
  - Where in the organization is the system used?
  - Who will benefit from the use of the system?
  - Who will support and maintain the system?
  - Does the system use an external resource?
  - Does one person play several different roles?
  - Do several people play the same role?
  - Does the system interact with a legacy system?

Practice on your own

Brainstorm some actors for ToDo List App

Today's Lecture – **How do we know what to build?**

- Use Cases
  - What and why
  - How
  - Use case diagrams
- Quiz 2 Study Guide
- Homework 1

Identifying Use Cases – Useful Questions

- What functions will the actor want from the system?
- What are the tasks of each actor?
- Does the system store information? What actors will create, read, update, or delete (CRUD) that information?
- What use cases will support and maintain the system?
- Can all functional requirements be performed by the use cases?

Scope of a Use Case

- *What*, not *how*
- Use audience-friendly terminology
- Include
  - How the use case starts and ends
  - The interactions (in sequence) between use case and actors
  - What data is needed by/exchanged during the use case
  - Basic flow
  - Alternative/exception flows (if applicable)

Example Use Case: Buy a Product

- Basic Flow
  1. Customer browses catalog and selects items to buy
  2. Customer goes to check out
  3. Customer fills in shipping information
  4. System presents full pricing information, including shipping
  5. Customer fills in credit card information
  6. System authorizes purchase
  7. System confirms sale immediately
  8. System sends confirmation email to customer
- Alternative Flow

  3a. Customer is regular (repeat) customer

  1. System displays current shipping, pricing, and billing information
  2. Customer may accept or override defaults, returns to BF at step 6

Example Use Case: Add Item Deadline

- Basic Flow
  1. User selects list item to which they want to add a deadline
  2. System presents detailed view of list item (name, completed/not completed, description)
  3. User selects the deadline field
  4. System presents user with a way to enter a date and time
  5. User enters date and time for deadline
  6. System shows item with new deadline
- Alternative Flow

  5a. User wants to add deadline to calendar
  1. After adding deadline, user selects an option to post to calendar
  2. System posts deadline to calendar, displays a confirmation to user
  3. Return to BF at step 6
- Exception Flow

  5a. Date/time is invalid
  1. System notifies user that date/time is invalid, deadline not added to item

Example Use Case: Toggle Completed State

- Basic Flow
  1. User selects list item they wish to toggle
  2. System presents detailed view of list item (name, completed/not completed, description)
  3. User toggles completed state
  4. System shows item with new completed state
- Alternative Flow

  1a. User swipes left on list item

  1. System displays the option to toggle completed state
  2. User confirms selection
  3. Return to BF at step 4

How to Build a Use Case

- Name use case
- Describe Basic Flow
- Add variations, if applicable
  - Alternative Flows
  - Exception Flows

Use Case Template

- Name/title
- Description
- Revision History
- Actors
- System Scope
- Goal
- Level
- Assumptions
- Relationships
  - Includes
  - Extends
  - Extension Points
- Precondition
- Trigger Events

Use Case Template (cont’d)

- Basic Flow 1 – Title
  - Description (steps), etc.
- Post conditions
- Alternative Flow 1 – Title
  - Description (steps)
- Alternative Flow 2 – Title
  - Description (steps)
- Alternative Flow 3 – Title
  - Description (steps)
- Exception Flow 1 – Title
  - Description (steps)
- Activity Diagram
- User Interface
- Special Requirements
  - Performance Requirements
  - Reports
  - Data Requirements
- Outstanding Issues

Practice on your own

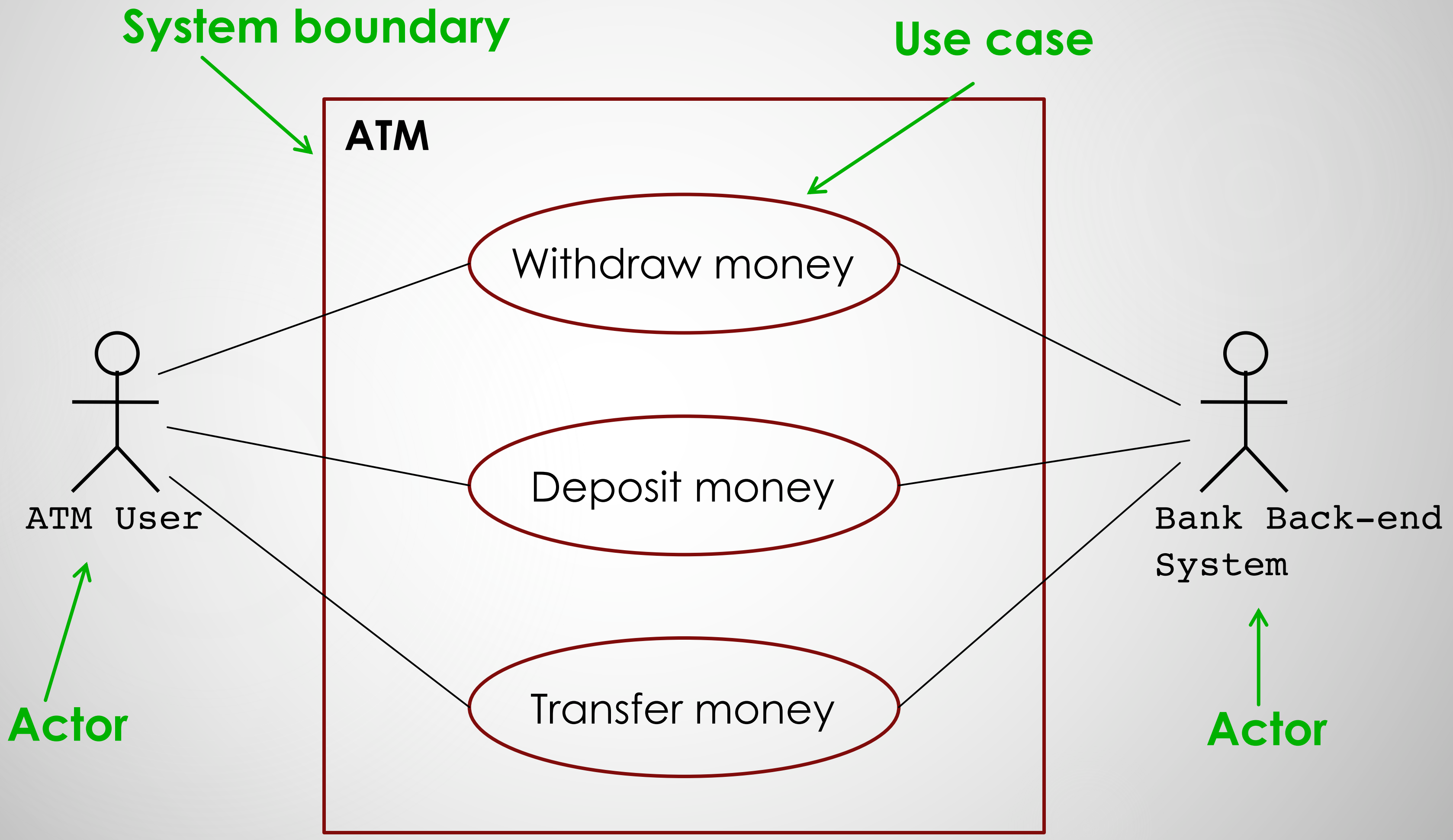
Brainstorm some more use cases for ToDo List App

Today’s Lecture – **How do we know what to build?**

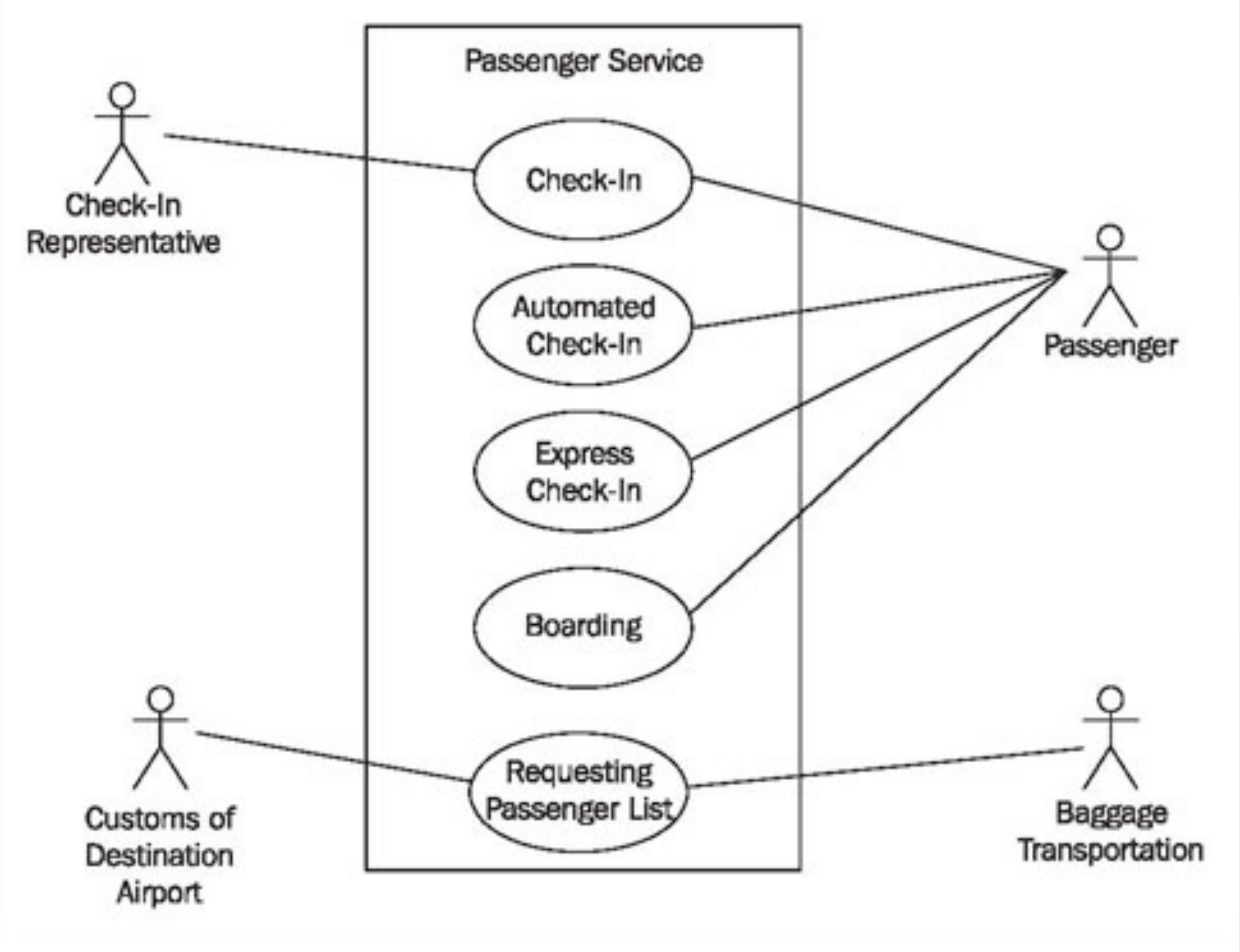
- Use Cases
  - What and why
  - How
  - Use case diagrams
- Quiz 2 Study Guide
- Homework 1

Use Case Diagrams

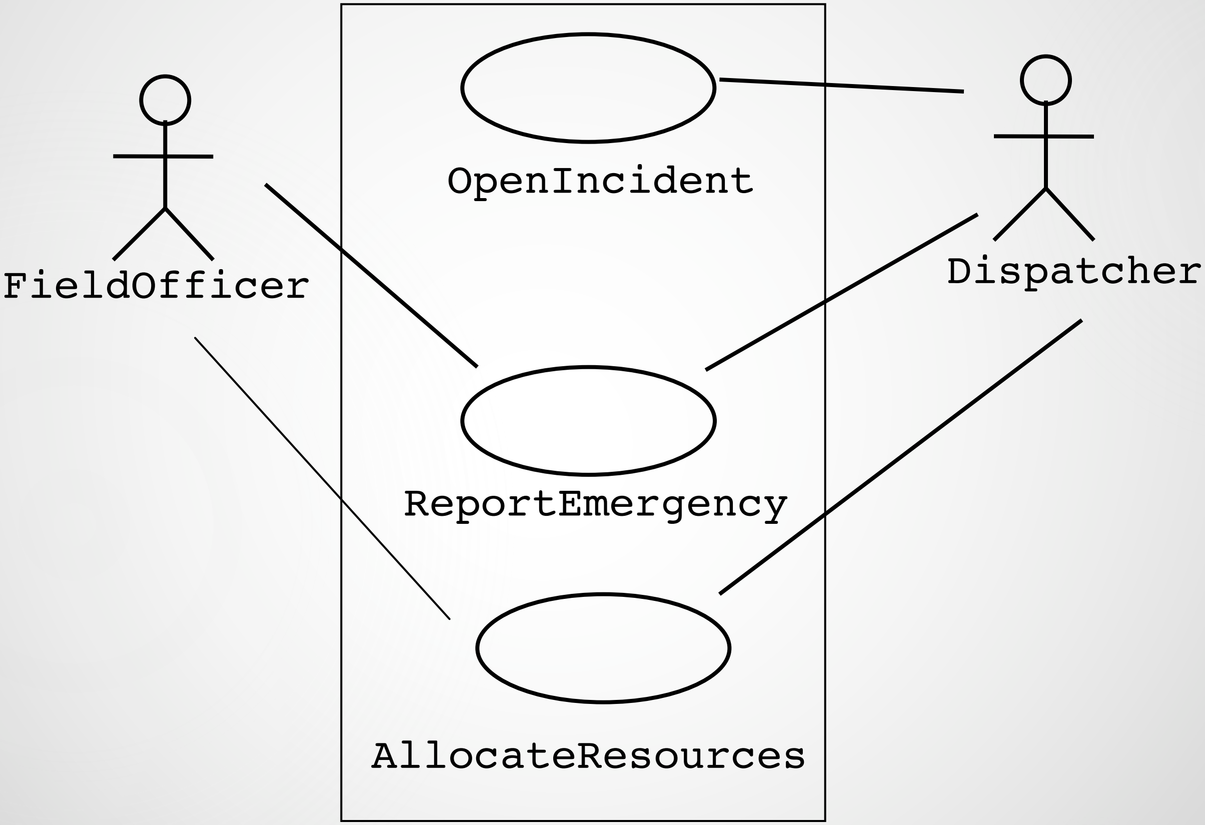
A graphical view of actors, use cases, and their interactions



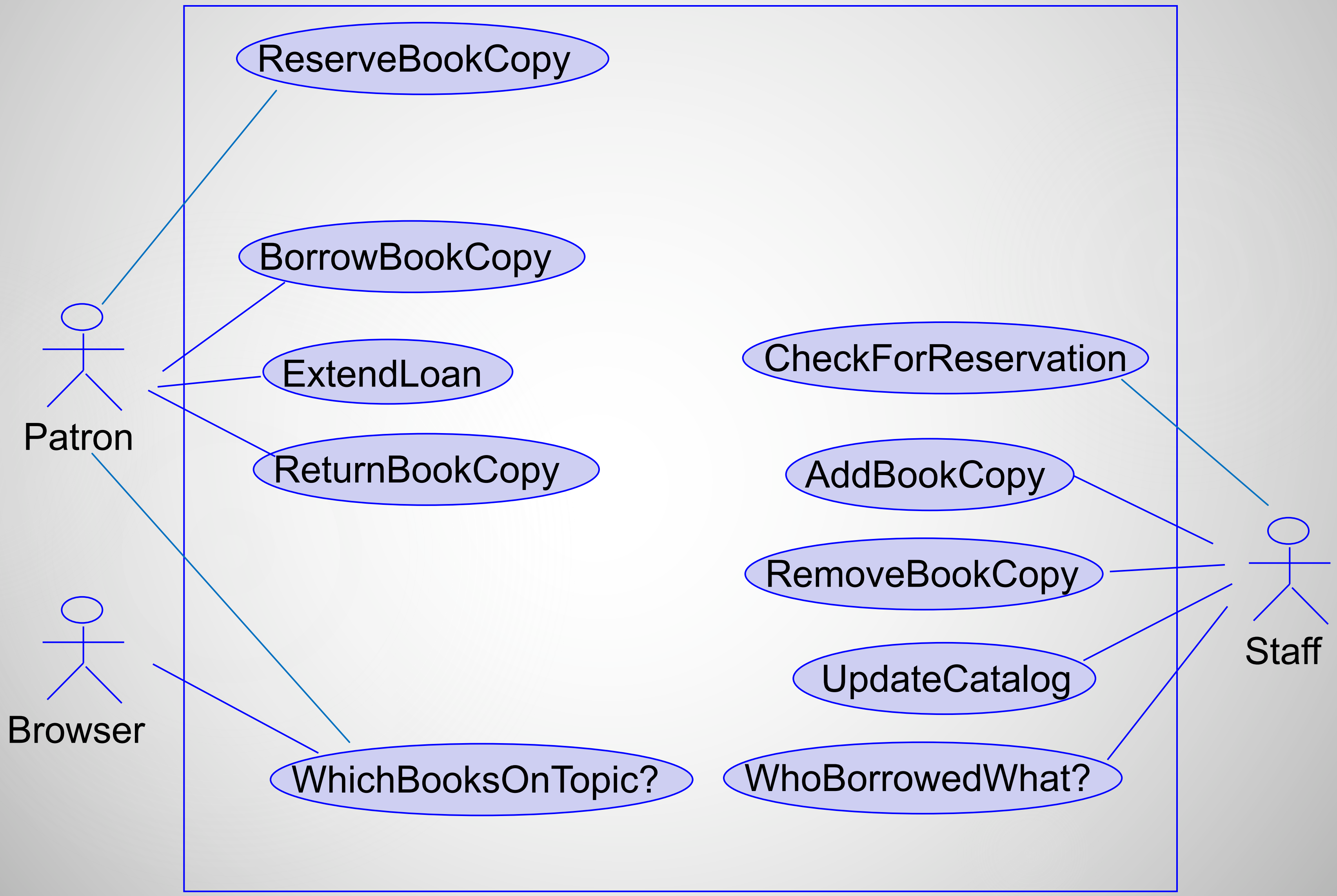
Air Travel Use Case Diagram



Incident Management Use Case Diagram



Library Software Use Case Diagram

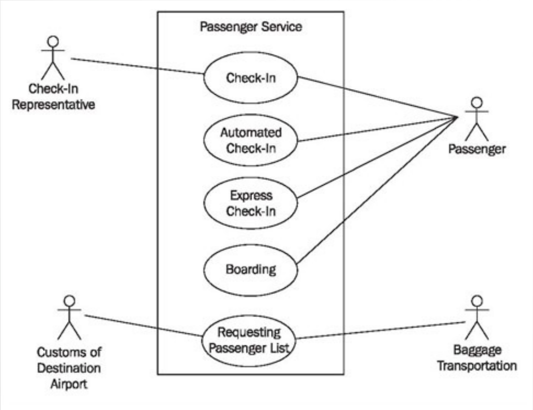


Attendance Quiz

Bringing it all together…

There is generally one **use case model** per system, consisting of:

1. Use case diagrams (one or more)
2. Textual descriptions of use cases



+

The connection was reset

The connection to the server was reset while the page was loading.

. The site could be temporarily unavailable or too busy. Try again in a few moments.

. If you are unable to load any pages, check your computer's network connection.

. If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

. The site could be temporarily unavailable or too busy. Try again in a few moments.

. If you are unable to load any pages, check your computer's network connection.

A Variety of Readers

- Marketing personnel, human factors engineers, specialty engineers
- System engineers
- Reviewers
- Software developers
- System/software testers
- Project managers
- Technical writers
- …

Use cases are used as a unifying thread throughout development

A Variety of Readers

- Marketing personnel, human factors engineers, specialty engineers
- System engineers
- Reviewers
- Software developers
- System/software testers
- Project managers
- Technical writers

Use cases are used as a communication/understanding tool among diverse stakeholders

Reminder: Fundamental Principles

- Rigor and formality
- Separation of concerns
  - Modularity
  - Abstraction
- Anticipation of change
- Generality
- Incrementality

These principles apply to all aspects of software engineering!

Summary

- Use case = textual description defining interactions between an actor and the system to achieve the primary actor's goal
  - Includes different flows (basic, alternative, exception)
- Use case model
  - Diagram(s)
  - Use case descriptions
- Use cases serve as a unifying thread throughout development
- Use cases serve as a communication/understanding tool among diverse stakeholders

Today’s Lecture – **How do we know what to build?**

- Use Cases
  - What and why
  - How
  - Use case diagrams
- Quiz 2 Study Guide
- Homework 1

Quiz 2 – Topics (I)

- Know and understand software failure causes and how they relate to requirements issues
- Know “which category employs the most programmers in the US”
- Know and understand the main ideas of the online failure readings
- Know and understand
  - The definition of requirements
  - What happens in the requirements phase
  - Techniques for requirements analysis
  - What goes in each section of a requirements document
  - Ziv’s Law

Today's Lecture – **How do we know what to build?**

- Use Cases
  - What and why
  - How
  - Use case diagrams
- Quiz 2 Study Guide
- Homework 1

Homework 1

- You will specify the requirements for an app
- Homework 1 will be posted before Tuesday’s class
  - Along with a template, rubric, and some example requirements documents
  - All of the instructions are contained in the prompt and in the template

Homework 1 - Client Interview

- Client interview will span two lectures
- Read the prompt and template
- Read the samples
- Come prepared with your questions
  - : http://www.bridging-the-gap.com/what-questions-do-i-ask-during-requirements-elicitation/
- Research existing similar systems
- For this exercise, I am a "naïve" client who
  - knows very little about software engineering
  - knows only about my "business"
  - will not answer any "customer" questions outside of lecture

Next Time (next Tuesday)

- Client interview, part 1 (3:30–4:20pm)
- Quiz 2 (4:20–4:50pm)