

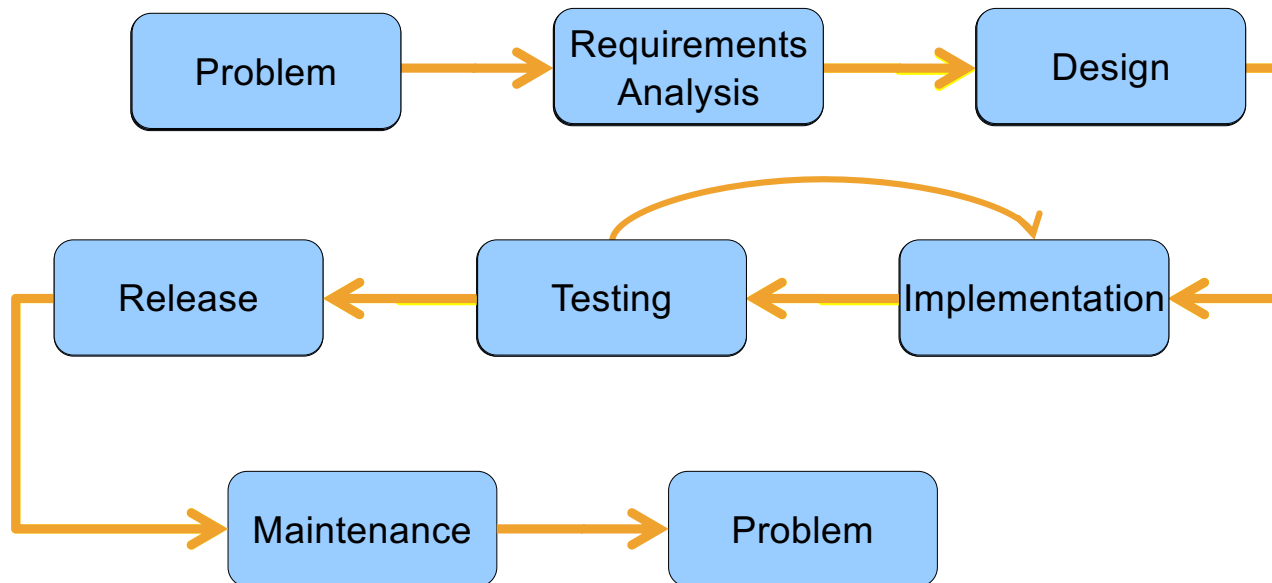
# CS 4320 / 7320

## Software Engineering

Module 3 - Requirements

# What is the SDLC?

## Where does Requirements Analysis fit?



# What is a software requirement?

A **requirement** is a property that **must be exhibited** by the software system in question in order to **solve some problem** in the real world.

Each requirement must be **verifiable** within **available resource constraints**.

# A requirement is either a ...

## Product Requirement

A need or constraint on the **software** being developed

Ex: The software shall verify a student meets all prerequisites before registering for a course.

## Process Requirement

A constraint on the **development** of the software

Ex: The software shall be developed using a FDD process.

# Product requirements may be...

## Functional

- Capabilities, features, or functions the software will execute
- Finite test steps can be written for it
- A description of a **behavior** that a system will exhibit under specific conditions. [Wiegers]

## Non-functional

- Constraints on the solution
- Quality attributes
  - Performance
  - Maintainability
  - Reliability
  - Security
  - Interoperability
  - Etc...
- A description of a **property or characteristic** that a system must exhibit or a **constraint** that it must respect. [Wiegers]

# System Requirements

A system is an interacting combination of elements to accomplish a defined objective. These include **hardware**, software, firmware, people, information, techniques, facilities, services, and other **support elements**.

**System requirements** are requirements for the **system as a whole**.

***Software requirements are derived from*** system requirements. [SWEBOK 1.6]

A system requirements is a **top-level requirement** for a product that contains multiple subsystems, which could be all software or software **and hardware**. [Wiegers]

# System Requirements

- Pre-requisites that often define the operating environment
- Architecture
- Hardware
  - *Storage*
  - *Memory*
  - *CPU*
  - *Connectivity etc.*

# User Requirements

- Identify the different classes of user in your system
- What are the goals of that user? What do they want to be able to do?

*A **prospective customer** shall be able to **add items to a shopping cart**.*

*A **customer** shall be able to **check out**.*



# Requirements are refined

1. **Business Requirements** – **WHY** is *the business* starting this project?  
Vision and Scope
2. **User Requirements** – **WHAT** does *the user* need to do?  
Most often represented with Use Cases
3. **Functional Requirements** – **WHAT** does *the developer* implement? (Not HOW)  
Derived from User Requirements, influenced by System Requirements and Non-Functional Requirements  
Gives the developer **enough detail** to know what **to implement**

# Example: User Requirements to Functional Requirements

## User Requirement for a Web Browser:

- The user shall be able to edit bookmarks.

## Functional Requirements for a Web Browser:

- The system shall display bookmarks as a collapsible and expandable hierarchical tree.
- The user shall be able to resequence bookmarks.
- The system shall display bookmark properties.
- The user shall be able to modify a bookmark's name, URL, and description. [Wiegers]

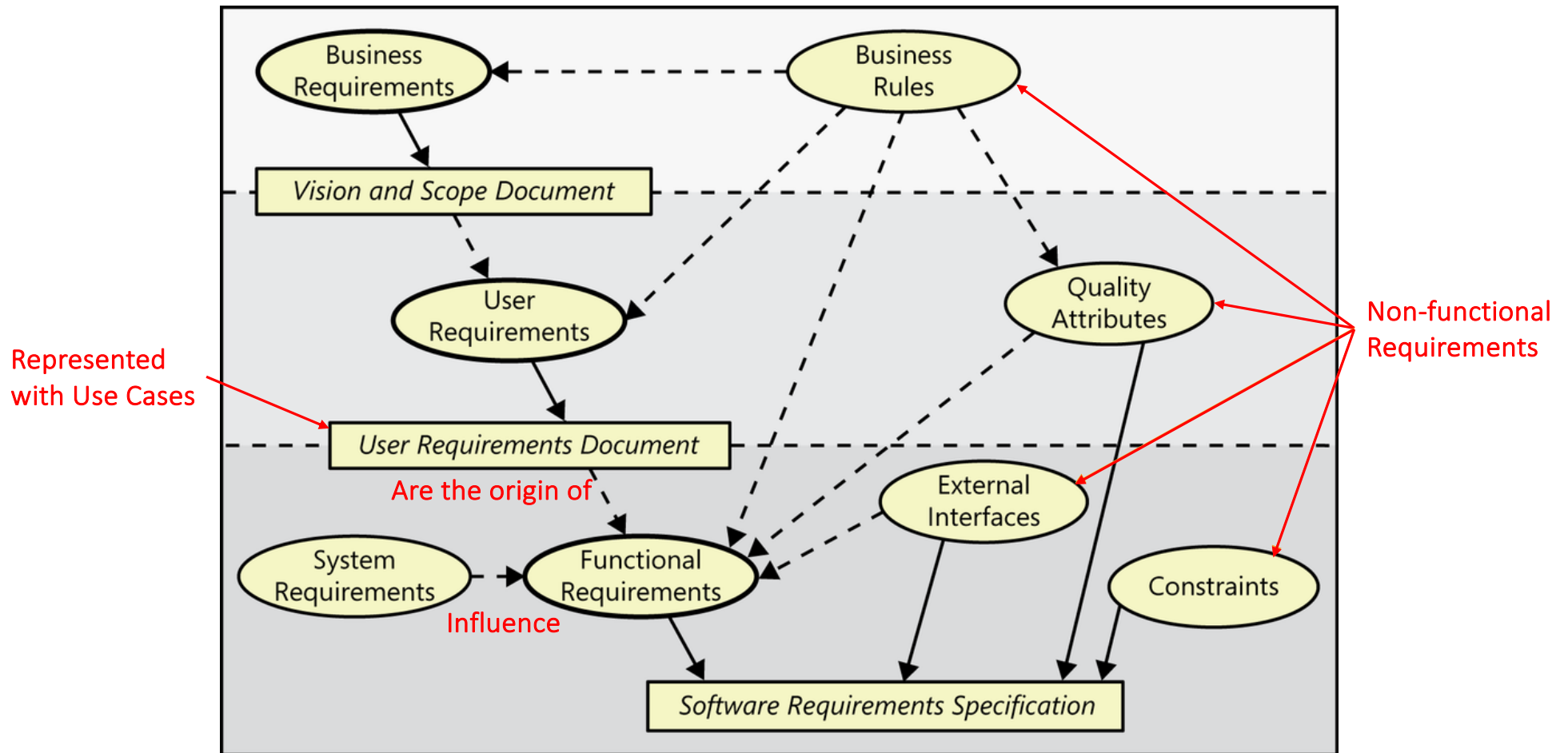


Figure 1-1. Relationships among several types of requirements information.

Solid arrows mean "are stored in"; dotted arrows mean "are the origin of" or "influence."

Source: K. E. Wiegers and J. Beatty, *Software Requirements*, 3rd ed.,  
Redmond, Washington: Microsoft Press, 2013.

# Excellent requirement specs are...

1. Complete – nothing missing (hard to tell!)
2. Consistent – does not conflict with other requirements
3. Correct – accurately states a user or external need (customer is the judge)
4. Feasible – can be implemented within known constraints
5. Modifiable – can be changed, with history, when necessary (unique identifiers, changelog or source control history)

Source: Video 24 - Characteristics of Excellent Requirements by Enfocus Solutions, <https://www.youtube.com/watch?v=6RSkUhZkPJM>

# Excellent requirement specs are...

6. Prioritized – ranked as to importance of inclusion in product
7. Necessary – documents what users really need  
(need  $\neq$  want)
8. Traceable – can be linked to system or user requirements, designs, code, and tests
9. Unambiguous – one possible meaning to all readers
10. Verifiable – testing, inspection, analysis, or demonstration can determine correct implementation.

Source: Video 24 - Characteristics of Excellent Requirements by Enfocus Solutions, <https://www.youtube.com/watch?v=6RSkUhZkPJM>

# Verifiable Requirements are (usually) Quantifiable

## Some Requirements ...

1. The software shall be reliable.
2. The call center will have better throughput.



*What does that mean? How do I know if I've achieved that?*

## Verifiable Requirements...

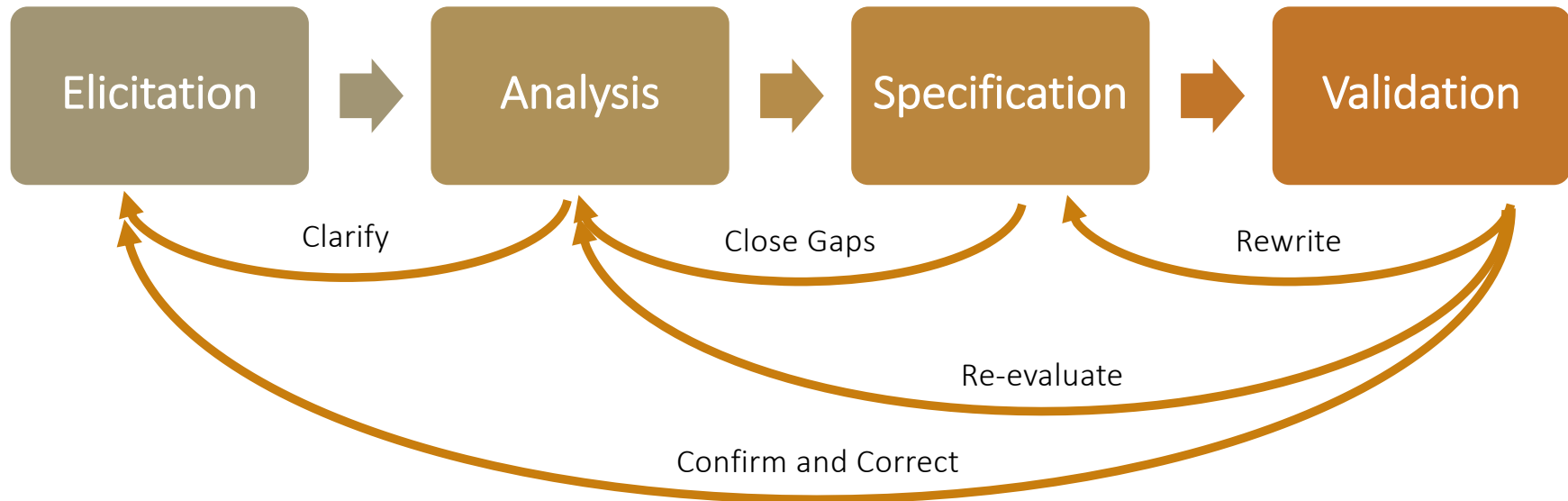
1. The software system shall have a probability of generating a fatal error during any hour of operation of less than  $1 \times 10^{-8}$ .
2. The call center software must increase the center's throughput by 20%.

[SWEBOK 1.5]

## A few practical properties...

- Priority rating   
*to enable trade-offs when faced with finite resources*
- Status value   
*to enable project progress tracking*
- Unique identifier **1.2a**  
*to enable tracking and configuration management*

# Requirements Process





# Requirements Elicitation Sources

- Goals (Business Requirements)
- Domain Knowledge
- Stakeholders
- Business Rules
- Operational Environment
- Organizational Environment

# Requirements Elicitation Techniques

- Interviews
- Scenarios (use cases)
- Prototypes
- Facilitated meetings
- Observation
- User Stories
- Analyzing competitor products
- Analyzing existing system being replaced

# Requirements Analysis – Why?

- Detect and resolve conflicts
- Discover the bounds of the software and how it must interact with its organizational and operational environment
- Elaborate system requirements to derive software requirements

# Requirements Analysis – How?

Classify requirements

to help think about them in an organized way

- Functional vs non-functional
- Derived from stakeholder or emergent
- Product or process
- Priority
- Scope
- Volatility/stability

# Requirements Analysis – How?

## Organize requirements

to help track them and look for interactions

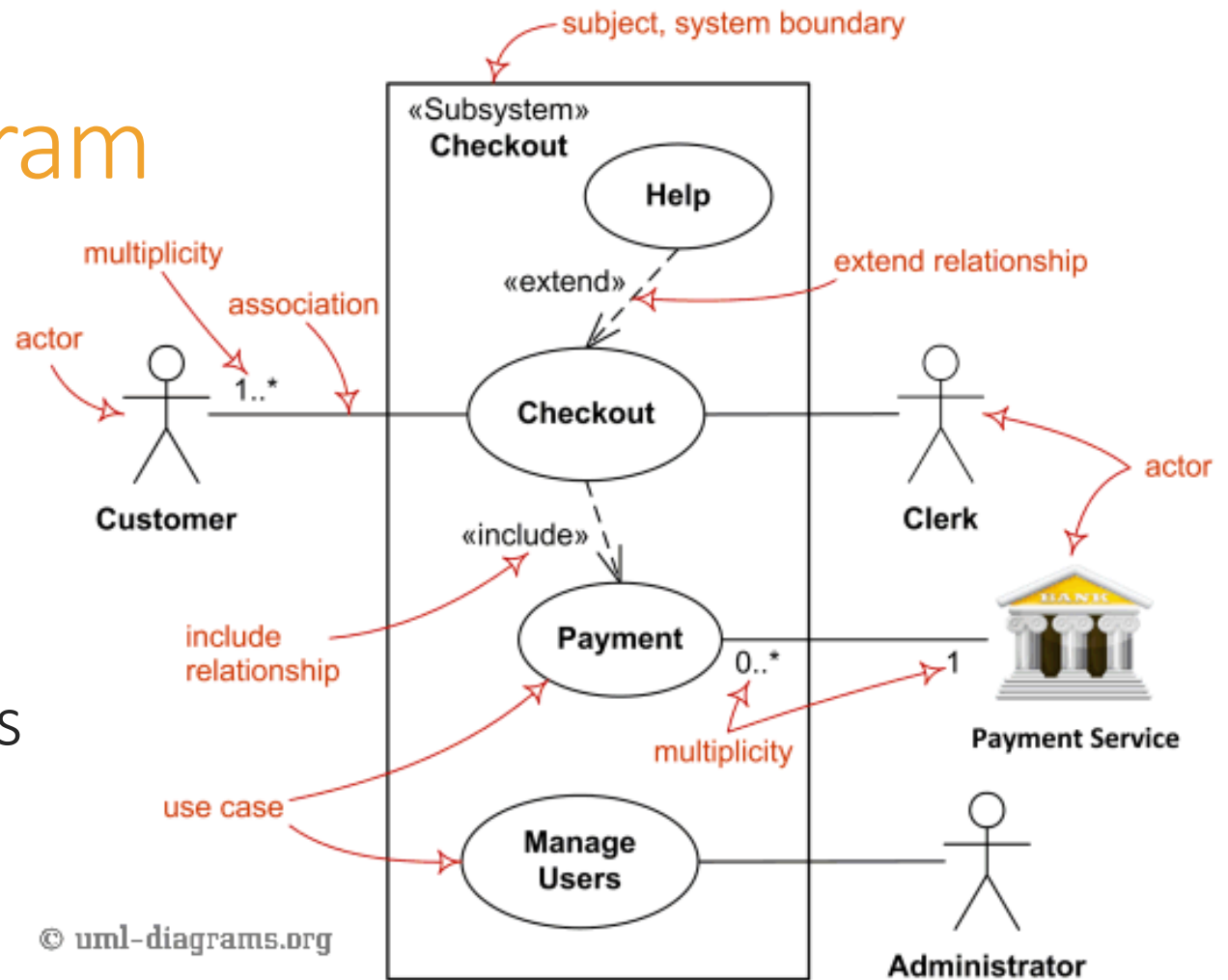
- By Business Process? Features? Subsystems?
- Leave room in your identifiers for additions and changes

## Conceptual modeling to understand the problem

- Use Case diagrams, data models, others as deemed useful

# Use Case Diagram

- Actors
- Use cases (functions)
- Included functions (required)
- Extended functions (optional)



# ERD

At Conceptual Level

Use to understand the data

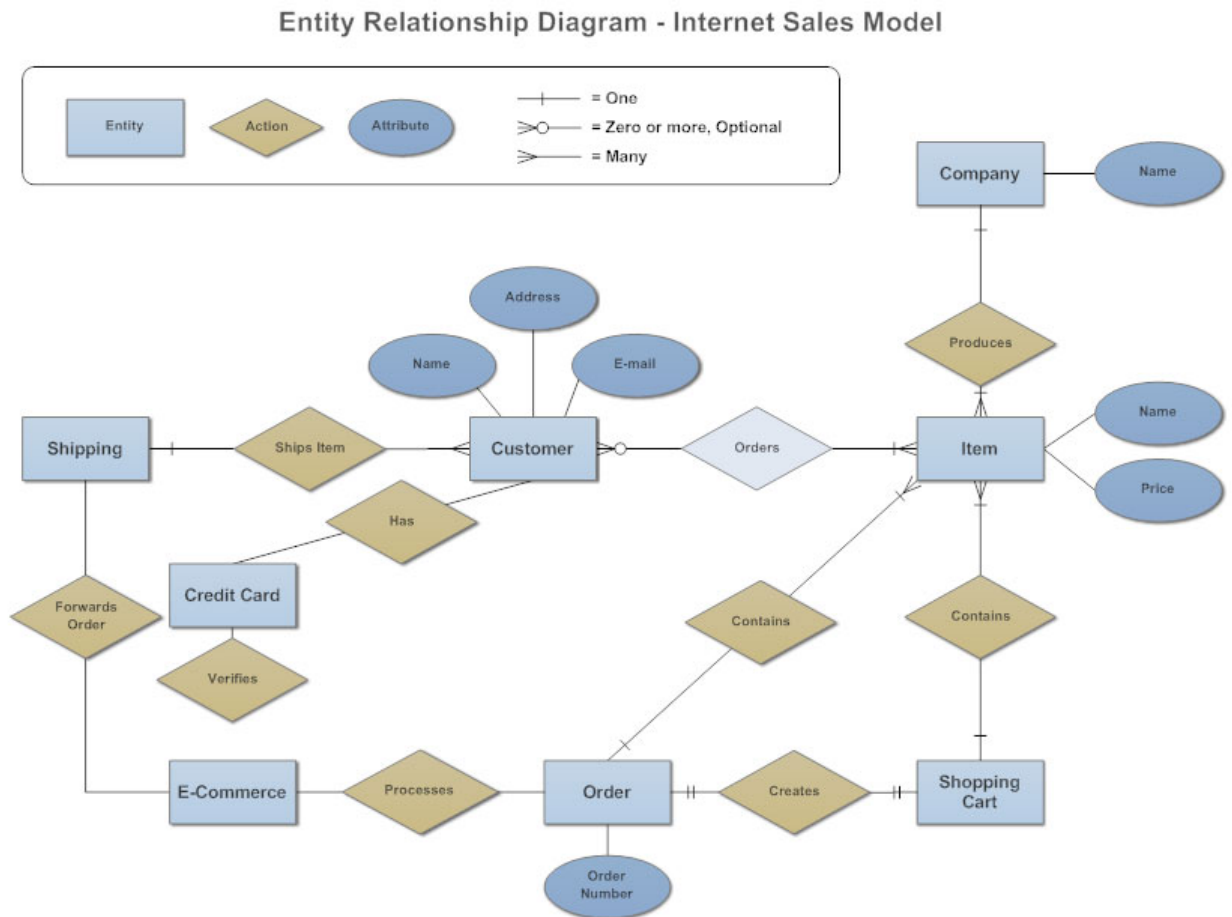
Even if you're using NoSQL!

Entities

Relationships

Cardinality

Attributes



Source: <https://www.smartdraw.com/entity-relationship-diagram/>

# Requirements Analysis – How?

## Architectural Design and Requirements Allocation

- Overlaps with design
- Which components will satisfy which requirements?
- Look for interactions

## Requirements Negotiation (Conflict resolution)

- Prioritization and customer involvement is key



# Requirements Specification

## Requirements documents

- Can be reviewed, evaluated, and approved.
- Has published versions, with changelogs noting changes made.

## Complex Projects

System Definition Document

System Requirements Specification

Software Requirements Specification

## Simple Projects

Software Requirements Specification

# Requirements Validation

- **Clear?** –unambiguous; one possible meaning to all readers
- **Correct?** – accurately states a user or external need
- **Consistent?** – no conflicts or contradictions
- **Complete?** – nothing missing
- **Feasible?** – can be implemented within resource constraints
- **Verifiable?** – we can tell if the requirement has been achieved

# Requirements Validation

- Requirements reviews
  - Include a customer representative
- Prototyping
  - Espec. for dynamic behaviors, user interfaces, critical features
  - Use low-quality prototypes to keep focus on topic
- Model Validation
- Acceptance Tests
  - Identifying and designing acceptance tests

# Requirements Management

## What to manage:

- Documentation
- Tracing - connecting up requirements, design, code, testing
- Change Management

## How to manage:

- Specialized tools
- Simpler, cheaper, but less satisfactory: spreadsheets

# References

P. Bourque and R. E. Fairley, Eds., SWEBOK v3.0: Guide to the Software Engineering Body of Knowledge, IEEE, 2014.

K. E. Wieggers and J. Beatty, Software Requirements, 3rd ed., Redmond, Washington: Microsoft Press, 2013.