[CS3704] Software Engineering

Dr. Chris Brown
Virginia Tech
9/18/2023

Announcements

- **Project Milestone 1 due Friday (9/22) at 11:59pm**
  - Lightning talk presentation slides
  - Proposal document

Requirements Specification

Requirements Specification
Documentation
User Stories
Domain Models and Class Diagrams

Learning Outcomes

By the end of the course, students should be able to:

- **Understand software engineering processes, methods, and tools used in the software development life cycle (SDLC)**
- **Use techniques and processes to create and analyze requirements for an application**
- Use techniques and processes to design a software system
- Identify processes, methods, and tools related to phases of the SDLC
- Explain the differences between software engineering processes
- Discuss research questions and current topics related to software engineering
- Create and communicate about the requirements and design of a software application

Requirements

Goal: Understand customer requirements for the software system

- The *what* of the project

- Very difficult to "get right" the first time and evolve over the course of development
  - Remember the Top 3 reasons for project failure: **(2)** Incomplete and **(3)** Changing Requirements
- *Software Artifacts:* requirements documents, use cases, user stories,...

What are requirements?

Definition: Capabilities and conditions to which the system — and more broadly, the project — must conform. [Larman]

- Focusing on the **WHAT** *not* the **HOW**
- **Should always come *first* in the SDLC**

Analysis vs. Specification

- **Analysis:** process of *understanding* the problem and the requirements for a solution
- **Specification:** process of *describing* what a system will do

➔ *Analysis leads to Specification* – they are not the same!

Requirements Specification

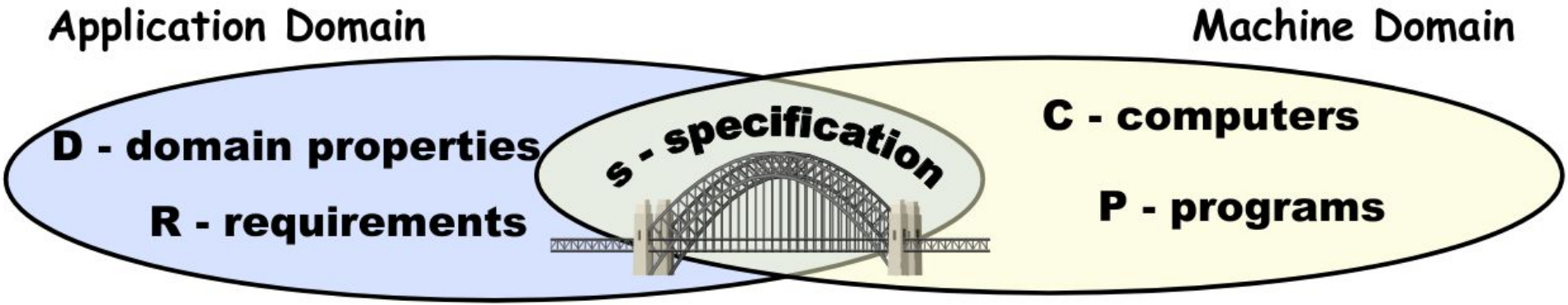
- Describing what the system will do

“Creating a business, product or a piece of software, is a complicated and long winded process…The software requirements specification, very often, is the developer’s bible, for guidance and direction.”

Requirements Specification (cont.)

- Why we need to write specifications
  - Purpose and audience
  - Choosing an appropriate size and formality
- Desires for Specifications
  - Properties of good specifications
  - Typical problems
  - What not to include
- Structure of a requirements document
  - IEEE standard

Requirements Specification (cont.)




Application Domain
Machine Domain
D - domain properties
R - requirements
S - specification
C - computers
P - programs

Software Requirements Specification

- How we communicate Requirements to others

Purpose

- Communication
  - explains the application domain and the system to be developed
- Contractual
  - May be legally binding!
  - Expresses agreement and a commitment
- Baseline for evaluating the software
  - supports testing, V&V
  - “enough information to verify whether delivered system meets requirements”
- Baseline for change control

Audience

- Customers & Users
  - interested in system requirements…
  - ...but not detailed software requirements
- Systems (Requirements) Analysts
  - Write other specifications that inter-relate
- Developers, Programmers
  - Have to implement the requirements
- Testers
  - Have to check that the requirements have been met
- Project Managers
  - Have to measure and control the project

SRS (cont.)

- SRS will differ based on your project, team, etc.

Consider two different projects:

A) Tiny project, 1 programmer, 2 months work

- programmer talks to customer, then writes up a 2-page memo

B) Large project, 50 programmers, 2 years work

- team of analysts model the requirements, then document them in a 500-page SRS

SRS Approaches

- **Natural language**
  - ***Documentation***
  - ***User stories***

    - "The system shall report to the operator all faults that originate in critical functions or that occur during execution of a critical sequence and for which there is no fault recovery response."

    (this is adapted from a real NASA spec for the international space station)

- **Visualization**
  - ***Decision Table***
  - ***Class Diagrams***

Originate in critical functions?	F	T	F	T	F	T	F	T
Occur during critical sequence?	F	F	T	T	F	F	T	T
No fault recovery response?	F	F	F	F	T	T	T	T
Report to operator?								

SRS (cont.)

	Project A	Project B
Purpose of spec?	Crystalizes programmer's understanding; feedback to customer	Build-to document; must contain enough detail for all the programmers
Management view?	Spec is irrelevant; have already allocated resources	Will use the spec to estimate resource needs and plan the development
Readers?	**Primary**: Spec author; **Secondary**: Customer	**Primary**: programmers, testers, managers; **Secondary**: customers

Qualities of SRS

Valid (or “correct”)

- Expresses the real needs of the stakeholders (customers, users,...)
- Does not contain anything that is *not* “required”

Unambiguous

- Every statement can be read in exactly one way

Complete

- All the things the system must do...
- ...and all the things it must not do!
- Conceptual Completeness
  - E.g. responses to all classes of input
- Structural Completeness
  - E.g. no TBDs!!!

Understandable (Clear)

- E.g. by non-computer specialists

Consistent

- Doesn’t contradict itself
- Uses all terms consistently

Ranked

- Indicates relative importance / stability of each requirement

Verifiable

- A process exists to test satisfaction of each requirement

Modifiable

- Can be changed without difficulty
  - Good structure and cross-referencing

Traceable

- Origin of each requirement is clear
- Labels each requirement for future referencing
- Doesn’t contradict itself
- Uses all terms consistently

[Adapted from IEEE-STD-830-1998]

Content of SRS

Software Requirements Specification should address:

- *Functionality*
  - What is the software supposed to do?
- *External interfaces*
  - How does the software interact with people, the system's hardware, other hardware, and other software?
  - What assumptions can be made about these external entities?
- *Required Performance*
  - What is the speed, availability, response time, recovery time of various software functions, and so on?
- *Quality Attributes*
  - What are the portability, correctness, maintainability, security, and other considerations?
- *Design constraints imposed on an implementation*
  - Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) and so on?

Content of SRS

Software Requirements Specification should *not* address:

- Project development plans
  - E.g. cost, staffing, schedules, methods, tools, etc
  - Lifetime of SRS is until the software is made obsolete
  - Lifetime of development plans is much shorter
- Product assurance plans
  - Configuration Management, Verification & Validation, test plans, Quality Assurance, etc
    - Different audiences
    - Different lifetimes
- Design
  - Requirements and designs have different audiences
  - Analysis and design are different areas of expertise
    - i.e. requirements analysts shouldn't do design!
  - Except where application domain constrains the design
    - i.e. limited communication between different subsystems for security reasons.

Organizing SRS

- **Need a logical organization for the document**
  - IEEE standard offers different templates
- **Example Structures to organize by:**

...External stimulus or external situation
e.g., for an aircraft landing system, each different type of landing situation: wind gusts, no fuel, short runway, etc

...System feature
e.g., for a telephone system: call forwarding, call blocking, conference call, etc

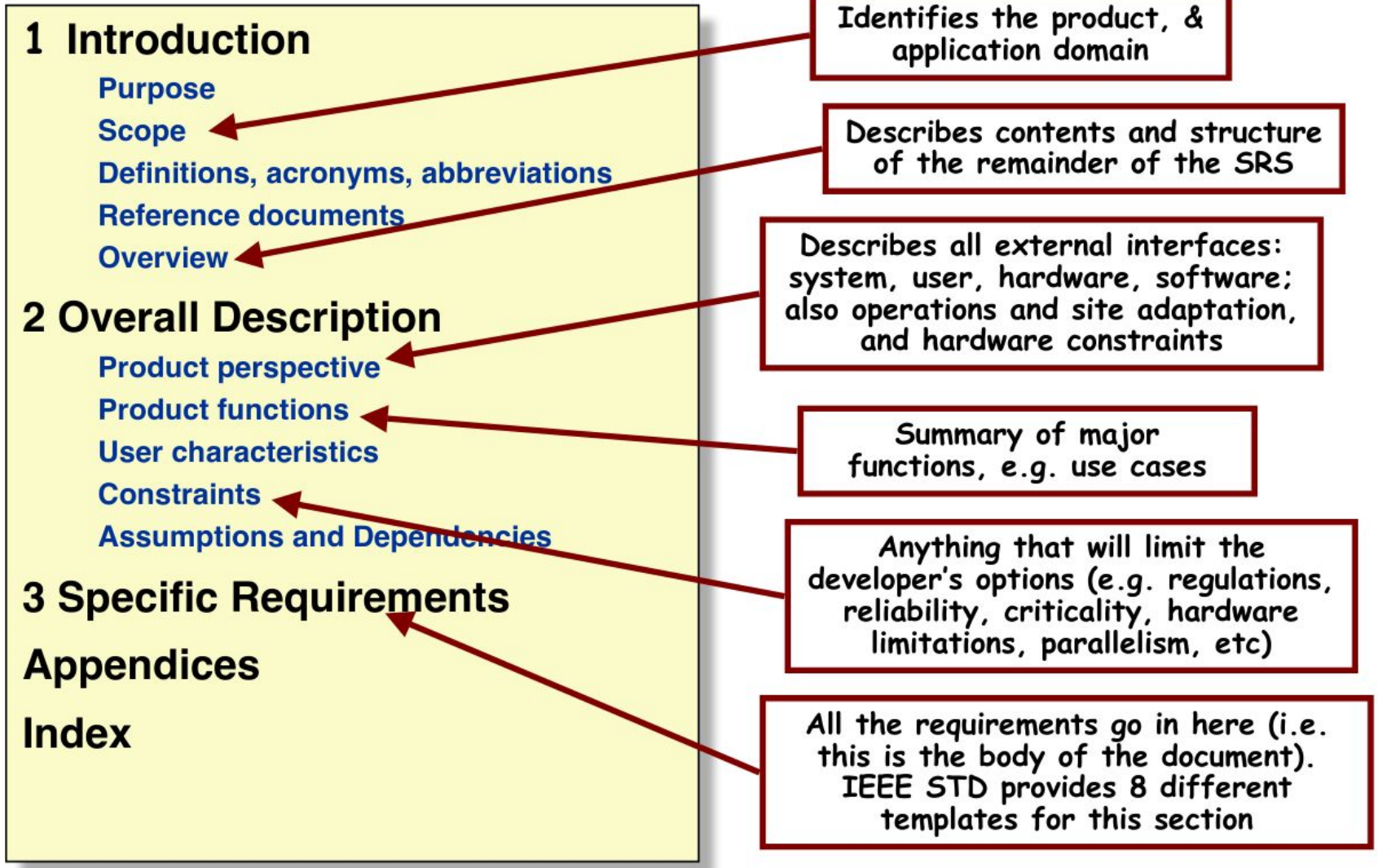
...System response
e.g., for a payroll system: generate pay-checks, report costs, print tax info;

...External object
e.g. for a library information system, organize by book type

...User type
e.g. for a project support system: manager, technical staff, administrator, etc.

...Mode
e.g. for word processor: page layout mode, outline mode, text editing mode, etc

...Subsystem
e.g. for spacecraft: command & control, data handling, comms, instruments, etc


1 Introduction
Purpose
Scope
Definitions, acronyms, abbreviations
Reference documents
Overview
2 Overall Description
Product perspective
Product functions
User characteristics
Constraints
Assumptions and Dependencies
3 Specific Requirements
Appendices
Index
Identifies the product, & application domain
Describes contents and structure of the remainder of the SRS
Describes all external interfaces: system, user, hardware, software; also operations and site adaptation, and hardware constraints
Summary of major functions, e.g. use cases
Anything that will limit the developer's options (e.g. regulations, reliability, criticality, hardware limitations, parallelism, etc)
All the requirements go in here (i.e. this is the body of the document). IEEE STD provides 8 different templates for this section

IEEE Standard: Section 3

3.1 External Interface Requirements
3.1.1 User Interfaces
3.1.2 Hardware Interfaces
3.1.3 Software Interfaces
3.1.4 Communication Interfaces

3.2 Functional Requirements
(this section organized by mode, user class, feature, etc…)
For example:
3.2.1 Mode 1
3.2.1.1 Functional Requirement 1.1
...
3.2.2 Mode 2
3.2.1.1 Functional Requirement 1.1
...
…
3.2.2 Mode n
...

3.3 Performance Requirements
Remember to state this in measurable terms!

3.4 Design Constraints
3.4.1 Standards compliance
3.4.2 Hardware limitations
etc.

3.5 Software System Attributes
3.5.1 Reliability
3.5.2 Availability
3.5.3 Security
3.5.4 Maintainability
3.5.5 Portability

3.6 Other Requirements

[Adapted from IEEE-STD-830-1993. See also, Blum 1992, p160]

Common SRS Mistakes

Noise
- text that carries no relevant information to any feature of the problem.

Silence
- a feature that is not covered by any text.

Over-specification
- text that describes a detailed design decision, rather than the problem.

Contradiction
- text that defines a single feature in a number of incompatible ways.

Ambiguity
- text that can be interpreted in at least two different ways.

Forward reference
- text that refers to a terms or features yet to be defined.

Wishful thinking
- text that defines a feature that cannot possibly be *verified*.

Requirements on users
- Cannot require users to do certain things, can only assume that they will

Jigsaw puzzles
- distributing key information across a document and then cross-referencing

Duckspeak requirements
- Requirements that are only there to conform to standards

Unnecessary invention of terminology
- E.g. ‘user input presentation function’

Inconsistent terminology
- Inventing and then changing terminology

Putting the onus on the developers
- i.e. making the reader work hard to decipher the intent

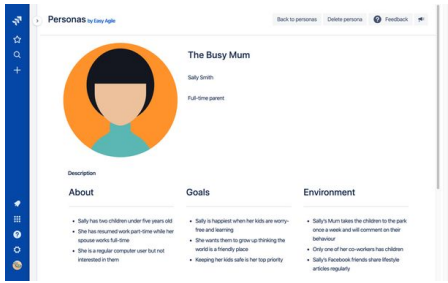
Writing for the hostile reader
- There are fewer of these than friendly readers

User stories

- Scenarios with explicit acceptance criteria
- Try not to make assumptions
- Definitive, implementable, and specific
- User story is determined to be finished when it meets acceptance criteria.

Writing a User Story

- Written from the point of view of the end user
  - Often on index cards
- Epics: larger user stories
- User personas to describe interactions



- *"As a* [persona type]*, I want to* [action] *so that* [benefit]*."*

Running Example

From the Course Project Ideas List

- **Standup Bot:** A software bot to automatically schedule standup meetings between teammates.
  - *Scrum master*: Professional to ensure scrum processes for development team

TODO: Complete a stand-up meeting!

- **What I did.**
- **What I need to do next.**
- **What is blocking me.**

Example: Stand-Up Bot

As a *Scrum Master*, I want to *find availability with a list of usernames* so that I can *schedule team daily stand-ups*.

Acceptance criteria:

Given Scrum Master has admin permissions
When Scrum Master types team member usernames
And <User can perform multiple actions here>
Then System finds list of available times between users

User stories vs. Use cases

"A user story is to a use case as a gazelle is to a gazebo" [Cockburn]

- They sound similar, but are very different.

- **Use case:** Description of ways a user may want to interact with the system (business).
  - Includes goal, actors, preconditions, steps, alternatives, and postconditions
- **User story:** Who, what, and why of a goal or outcome that the user wants to achieve (user).
  - Meant to be as simple as possible

Domain Models

- A visual representation of conceptual classes or real-situation objects showing:
  - Domain objects or conceptual classes
  - Relationship between conceptual classes
  - Attributes of conceptual classes
- Illustrated with a set of UML *Class diagrams*

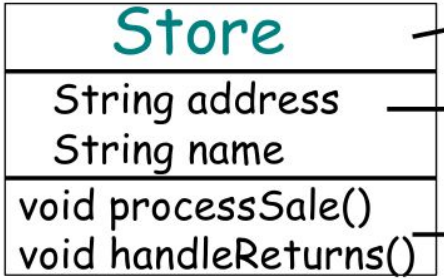
Role of Domain Models

- Build upon use cases
  - More emphasis on relationship
- Basis for design and implementation
  - Starting to move towards requirements specification, and design
- The most important and classic model in OO analysis

Class Diagram

- A visual representation of main objects and their relations for a system.
- Elements
  - Classes containing: Attributes, Operations
  - Various relationships: Association, Aggregation, Composition, Generalization

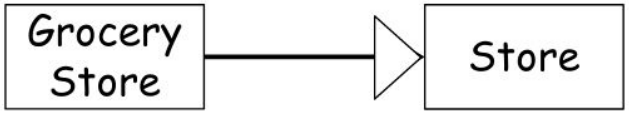
Class Diagram Example



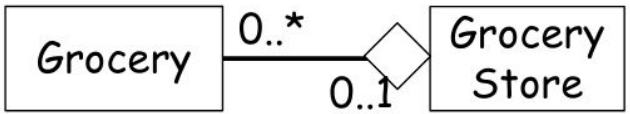
Class name: abstract concept

Attributes: properties relevant to the use case

Operation (Method signatures): behaviors of the class



Generalization: “is-a” relationship. A sub-class inherits all attributes and operations of its super class.



Aggregation: “has-a” relationship. The container and elements can exist independently from each other

Building a Class Diagram

- Step 1: Identify conceptual classes
- Step 2: Decide attributes
- Step 3: Identify associations between classes

Note: Step 1 and 2 may occur together, iteratively

1. Identify Conceptual Classes

- Reuse or modify existing partial models created by experts
  - “recipes” for well-known problems and domains (e.g., accounting, stock market, ...)
- Consider common categories
- Identify nouns and noun phrases from fully dressed use cases
  - Ex) ScrumMeeting

2. Decide Attributes

- Properties of the conceptual classes relevant to the problem domain
  - Nouns and noun phrases that the requirement suggest or imply a need to remember

– *E.g.*, date, time, location, members, etc. are all relevant to a ScrumMeeting conceptual class

3. Identify Associations

- Relationship between instances of conceptual classes
- Think of it as a mathematical relation
  - Typically a binary relation: R $\subseteq$ S1 x S2
    - S1 = set of instances of the first class
    - S2 = set of instances of the second class

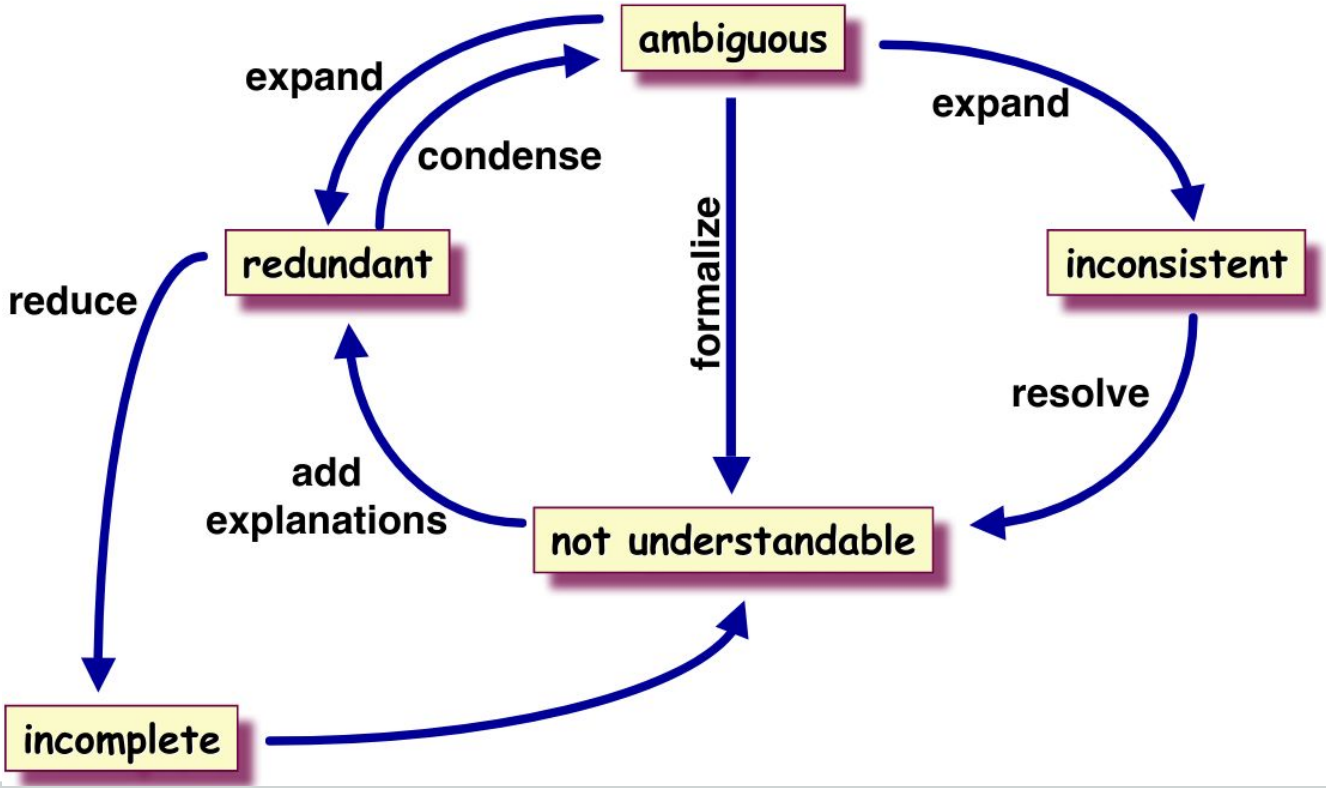
Typical Associations

- A is a physical/logical part of B
- A is physically/logically contained in B
- A is recorded/reported/captured in B
- A is a description of B
- A uses or manages B
- A is related to a transaction B
- A is owned by B…

Key Points on Class Diagram

- UML is just annotation
- UML class diagram means different things in different contexts
  - – Conceptual perspective: description of the domain model
  - – Specification perspective: description of software abstractions or components
  - – Implementation perspective: description of Java classes
- A small set of UML class diagram elements
  - – Classes: Attributes, Operations
  - – Relationship: Generalization, Aggregation, Composition, Association

There is no Perfect SRS!



Summary

Requirements Specs have several purposes:

- Communication
- Contractual
- Basis for Verification
- Basis for Change Control

Requirements Specs have several audiences:

- Technical and non-technical

Good Specs are hard to write

- Complete, consistent, valid, unambiguous, verifiable, modifiable, traceable…
- No perfect specifications: SRS, user stories, domain models, class diagrams…

Project needs vary

- The amount of effort put into getting the spec right should depend on the possible consequences of requirements errors

Project Milestone 1

Due Friday (9/22) at 11:59pm

- Project Proposal document
- Lightning Talk slides
  - Upload separate pdfs on Canvas

- **Goal: Propose your group's project idea to the class and the instructor.**

Project Milestone 1: Lightning Talk

- **Slides due Friday (9/22) at 11:59pm**
- Presentations will be in-class September 25, 27, and 29
  - Order will be chosen randomly before class on Monday (9/25)
- 5 minute talk
  - 1 student to entire group
- ~1 minute for Q&A (high-level) and transition between groups
- Please bring adapter if you don't have HDMI

Project Milestone 1: Lightning Talk

Team	Points
Presenter(s) spoke clearly	10
Presentation slides are readable	5
Team name and members are clearly presented	5
Presenter clearly described the problem they will try to solve	20
Presenter(s) outlines how their project addresses the problem	20
At least one *specific* use case with main acceptance criteria (formal or informal)	20
Presenter explains how their project is relevant to teamwork and collaboration in SE	10
Presentation is 5 minutes long, +/- 30 seconds	10
	100

Project Milestone 1: Proposal

- **Document due Friday (9/22) at 11:59pm**
- Should be no more than two pages (excluding references) in the correct format
- Needs to include the following:
  - a relevant *title* and all group members listed as *authors*;
  - an *abstract* briefly describing the problem and proposed solution;
  - an *introduction* that further explains the problem and motivates the need for the proposed solution;
  - *related work* with relevant tools or research studies;
  - a brief description of the _*software engineering process*_ your team plans to use for the project and why;
  - and *references*

Project Milestone 1: Proposal

Team	Points
Proposal is clearly written and organized	10
Spelling, grammar, etc.	5
Relevant title and team members are clearly presented	5
Proposal is in the correct paper format	5
Abstract provides overview of the project	10
Introduction explains the problem and motivates proposed solution	10
Related work section discusses existing related tools and/or research	10
Proposal has a section describing SE process the team will adopt (discussed in class)	20
Proposal explains why the team chose the specific SE process	20
References are included	5
	100

Next Class

- **Requirements Engineering (9/20)**

- **Project Workday Friday (9/22) in class**
- **PM1 due Friday at 11:59pm**
  - Lightning Talk slides
  - Project proposal

References

Na Meng, Barbara Ryder. Virginia Tech

Sarah Heckman, Chris Parnin. NC State

Steve Easterbrook. University of Toronto