Requirements Engineering

JD Kilgallin

CPSC:480

09/14/22

Pressman Ch 7-8

If you don't know where you're going, you're unlikely to end up there.
-Forrest Gump. Paramount Pictures. 1994.

Notes

- Exercise 1 walkthrough posted to GitHub. Now due Friday. See Brightspace.
- Project 1 due Sunday.
- Guest speaker Monday: Gary Galehouse, Keyfactor VP of Engineering
  - Be prepared with questions on starting a career and being an effective engineer.
- Project 2 assigned next week.
  - Tentatively, posted Monday and discussed Wednesday.
  - Will be due Oct 16.
  - Submit group requests by Tuesday. Groups will be finalized Wednesday.
  - Project groups will be used for in-class exercise 2 next Wednesday.
- Quiz numbering changed to align with Brightspace. Quiz 1 = 8/22 survey, quiz 2 = 8/31 quiz (originally marked as quiz 1), quiz 3 today. Expect one more quiz before midterm, probably Wednesday 9/28.

Learning Objectives

- Process of establishing requirements for a software project.
- Roles involved in establishing requirements.
- Functional vs non-functional product requirements.
- Software specification.
- Modeling requirements.

What is a software requirement?

- IEEE Standard Glossary of Software Engineering Terminology:
  1. A condition or capability needed by a user to solve a problem or achieve an objective.
  2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
  3. A documented representation of a condition or capability in 1 or 2.

- A singular documented need that identifies a necessary attribute, capability, characteristic, or quality of a system in order for it to have value and utility to a user.

What is requirements engineering?

- "Requirements engineering (RE) is the process of discovering the purpose for which a software system is intended, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation."

- "A disciplined mechanism to understand the customer's desires, negotiate for a reasonable solution, describe the intended behavior of the proposed system & constraints associated with it for transfer to the working system."

- Requirements engineering is a function of project management and product ownership that occurs throughout the lifetime of a product and is integral to success of the people, process, project, & product.

Why are requirements *engineered?*

- Software requirements are complex and difficult to capture.
- Stakeholders have differing levels of sophistication and understanding of what is feasible for software systems to do, and an individual's vision may be difficult to translate into actionable development tasks.
- Users in a specific domain are rarely familiar with software development processes, and developers are frequently unfamiliar with the application domain, leading to communication barriers.
- Without a working product to reference, there are an endless number of options for reaching a solution to a problem stated in basic terms.
- Poor understanding and assumptions about needed software behavior leads to products unsuitable for their intended purpose, resulting in lower rates of adoption and commercial success.

When are requirements engineered?

- Requirements engineering *always* encompasses the first task in development of a new product.

- The largest amount of requirements engineering work occurs at the beginning of each iteration, and especially of the first iteration.

- Requirements engineering continues throughout each iteration cycle, as requirements and constraints are discovered, refined, validated, updated, and reprioritized.

- Requirements engineering happens both during design and implementation of functionality meeting the requirements and well in advance.

What types of requirements are there?

- *Functional requirements* define what the software must do. "The software must allow users to search for a student by id number".

- *Non-functional requirements* define *how* the software must do what it does. "The software must support more than 1000 users at a time."

- A requirement may blur the lines, but this frequently means it can be decomposed into two or more requirements "The software must display a list of students in three seconds or less"

  1. The software must display a list of students
  2. The software must perform an operation in 3 seconds or less.

- Requirements may be explicit or implicit (e.g. "Must not crash during routine use", or "Must allow keyboard input for text").

Who is involved in requirements engineering?

- Engineering team building the software.
- Product management team designing and planning the project.
- Executive team responsible for engineering organization activity.
- Vendor personnel who will market, sell, and support the product.
- Representative set of end users who will operate the software.
- Customers who pay for the product.
  - Frequently, the people who *buy* the software will not be the ones using it. e.g. Keyfactor software is for IT personnel who usually don't have authority to make large software purchase agreements. Or consider Facebook users vs advertisers.
- A *stakeholder* is anyone who will benefit from the developed software.

How are requirements engineered?

- Inception – Someone identifies a need for software to address a specific use case.

- Elicitation – Product team works to define basic requirements.

- Elaboration – Requirement details filled in to understand exact needs.

- Negotiation – Budgets, timelines, and resource availability considered

- Specification – A full(ish) statement of the work to be performed.

- Validation – Specification is confirmed to meet customer needs.

Initial requirements – inception

- Software always begins with an *idea* for a product.
- Initial conception may not even consider software, or at least new software ("We need to maintain a list of digital certificates and keys.")
- Many software concepts amount to "do _____, but on a computer."
- Software concepts may originate inside or outside of the developers' business and that of the end customer.
- Software concepts may be based on other software, typically software that doesn't work well or doesn't quite meet the use case.

Inception – common scenarios

- Management at an organization identifies a problem that can be solved with software and hires/assigns software engineers in the company to build it. Customers may include existing clients as well as marketing to new business opportunities.

- A professional services consulting firm may observe a software need from one or more clients and contract with a software vendor to build a solution for their market.

- Organization identifies a problem and contracts with a software vendor to build a custom solution, becoming sole customer of the new product. Common in web development.

- An individual or small team develops a product, discovers high demand, and builds a new software engineering organization to continue development (e.g. facebook).

Elicitation and Elaboration

- Identify goals for the project; what should users be able to accomplish with the software when it is released.

- Stakeholders will describe what they *think* they want. They may specify it in great detail. But what they describe may not be the best way to accomplish what they really want. The real requirements must be *discovered.*

- Customers will usually focus on functional requirements. Non-functional requirements must also be elicited.

- When goals of the product are established, each goal should be spelled out in one or more use cases that describe how a user will interact with the system.

- Iteratively refined and prioritized.

Negotiation

- Stakeholders will ask for more than can be done with the given time, budget, and other constraints.

- Many users do not have a good grasp of what makes one requirement easy and another challenging.

- Different groups of stakeholders may have very different priorities or even mutually exclusive requirements.

- Uncertainty about what is feasible complicates the process, and risk factors must be weighed and contingency plans considered.
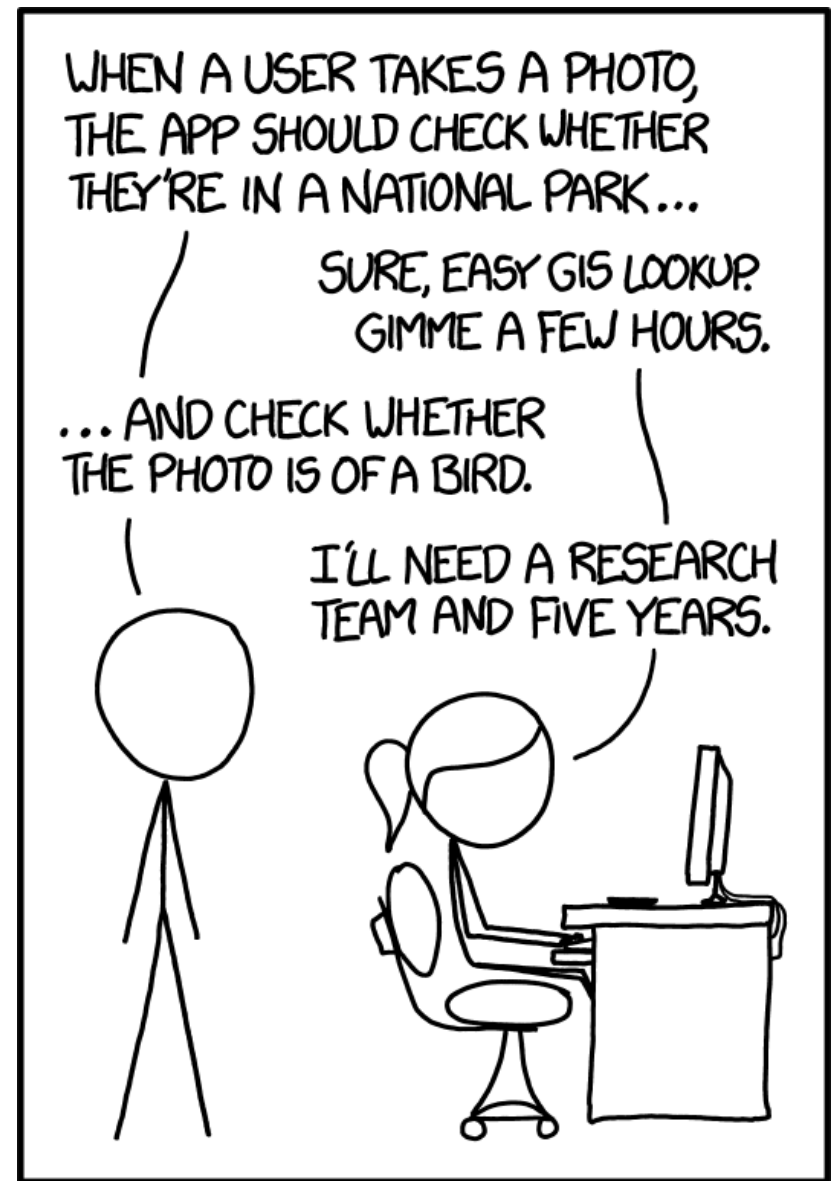
Specification

- After goals are defined, described, and prioritized, with consensus among stakeholders, requirements engineering begins to intersect with product design.

- A draft specification of the software to be built should be written at this time, usually by a product manager or owner, including:
  - Product scope and definition, and business justification.
  - Product functions and features (functional requirements)
  - Operating environment assumptions and constraints
  - Interfaces to users and other systems.
    - Diagrams and mockup drawings are recommended.
  - Non-functional requirements (scalability, security, compliance, etc)

Class reg. example – scope and definition

- Before software - "Students must register for classes by viewing a printed course catalog and bringing their course selections to the university registrar. This system does not consistently allow students to take the classes they want or need most, decreasing student satisfaction and education outcomes and sometimes delaying graduation. It is also expensive to maintain a large staff of deputy registrars developing class schedules and managing registrations."

- "Students today expect to be able to quickly find available classes and determine if they meet the prereqs, build a schedule that meets their needs, confirm their registration in a class, and change the schedule until after classes start, all from home. Our proposed software system will allow this, improving the experience at a lower operating cost."

Class reg. example – functions and features

- The software must support these functions:
  - As a department head, I can submit the classes in my department to be taught next semester and their proposed times.
  - As a registrar, I can confirm that proposed class schedules meet requirements and do not cause instructor schedule conflicts, and add them to the catalog.
  - As a student, I can view the classes I'm enrolled in, ones I've taken previously, and how they fulfill my degree requirements.
  - As a student, I can search and enroll in open classes that I want to take.
  - As an instructor, I can view the classes I am teaching and the student enrollment.
  - As an instructor, I can override registration requirements to allow a specific student to enroll in my class.

Class reg. example – operation and interfaces

- The software must run on a university-owned Windows server.
- The software must durably store all records of enrollment using a SQL database with the ability to save and restore snapshots.
- The software must allow students and teachers to access their functions using web forms through any standard web browser.
- The software must integrate with the student identification system for user authentication when performing functions as a student
- The software must integrate with faculty & staff HR system for user authentication when performing non-student functions.

Search

Term

Class number

Class name

Instructor name

Show open classes only ☐

Submit

Class reg. example – non-functional reqs

- The software must function with more than one million students and ten thousand faculty over at least sixty course terms.
- The software must support at least 10,000 simultaneous users.
- The software must maintain data in a format that can be audited and can be migrated to other systems.
- The software must not allow users to view any information that they are not authorized to access.
- The software must perform operations atomically that guarantee students cannot register for full classes and can confirm their registration was accepted.
- The software must allow use in both English and Spanish.
- The software must comply with all requirements of the Family Education Rights and Privacy Act and Ohio Revised Code.

Validation

- Final confirmation that specification meets all parties' approval.
- Inconsistencies and ambiguities resolved. Terminology should be consistent throughout and synonyms should not be used when they refer to the same concept.
- Assumptions made by the design team may be found to be invalid.
- New requirements will frequently be discovered that weren't originally captured.
- Underspecified requirements may also need further revision.
- When scope and functionality of a project is fully specified, engineering team may raise concerns about feasibility of some components.

Capturing requirements

- What is the requirement and how does it support the goals of the product?
- How will we know this requirement has been met?
- What component of the product does this requirement belong to?
- How difficult is this requirement to meet? What challenges are expected?
- Who will benefit from meeting this requirement?
- How important is this requirement to the users who will benefit?
- When does this requirement need to be met?
- Who defined this requirement? How has the requirement changed over time? What discussion has taken place regarding this requirement?
- What are the related requirements, supporting documents and diagrams, code commits, and test cases?

398   Cancel order form

Christie Church   💬 1   [Phone ✕] [Web ✕] [+]   🖫 Save & Close ▾   👁 Follow   •••

State	🔵 Active	Area	Fabrikam Agile Project
Reason	Implementation started	Iteration	Fabrikam Agile Project

Updated 8/19/2016

Details   🕘   🔗 (4)   📎

Description

B I U A̶ 🔗 🔗̶ ☰ ☰ ⇤ ⇥ 🖼

Provide a **cancellation order** form similar to the screen shown. See the attached storyboard for details.

Order details here
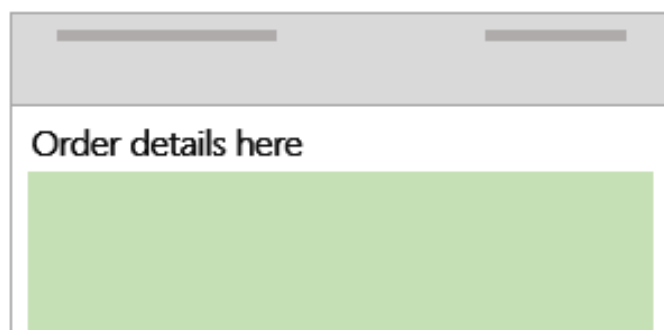
Acceptance Criteria

Discussion

Add a comment. Use # to link a work item or @ to mention a person

Helena commented 2 months ago
Reassigning this to Christie

Planning

Story Points

Priority
2

Risk

Classification

Value area

Business

Development

➕ Add link

Development hasn't started on this item.

Create a new branch

Related Work

➕ Add link ▾

Parent
340 Improve User Experience
Updated 7/31/2014, 🔵 Active

Child
466 Develop standards guid...
Updated 7/12/2016, 🔵 Active

Related
468 Customer account history
Updated 8/19/2016, 🔵 Active

Meeting requirements

- As construction and later phases of software development occur, work should be able to be traced back to the requirement or portion of the software specification it meets.

- Work that does not map to a requirement usually indicates either a missed requirement or unnecessary work. However, some work must be done to address technical debt and maintainability of the code.

- Requirements will almost always be added, modified, or reprioritized as development continues. Good software architecture makes it easier to meet changed requirements.

- Some changes to requirements cannot feasibly be accommodated. This is why a rigorous specification is needed in software development contracts. The later development is, the harder change is. Agile development *helps*.

Modeling requirements

- *Class models* describe objects to be modeled within the software, the operations that can be done with them, and their relationship. Object-oriented programming is to some extent based on class diagrams from requirements modeling.

- *Scenario models* describe a use case and the sequence of actions a user might take to accomplish the task at hand. There are many ways to represent this such as a *sequence* (or *swimlane*) diagram.

- *Behavioral models* describe what input and output the software may consume or produce at a given point, and how the software transitions to other states over time. Usually uses state diagrams.
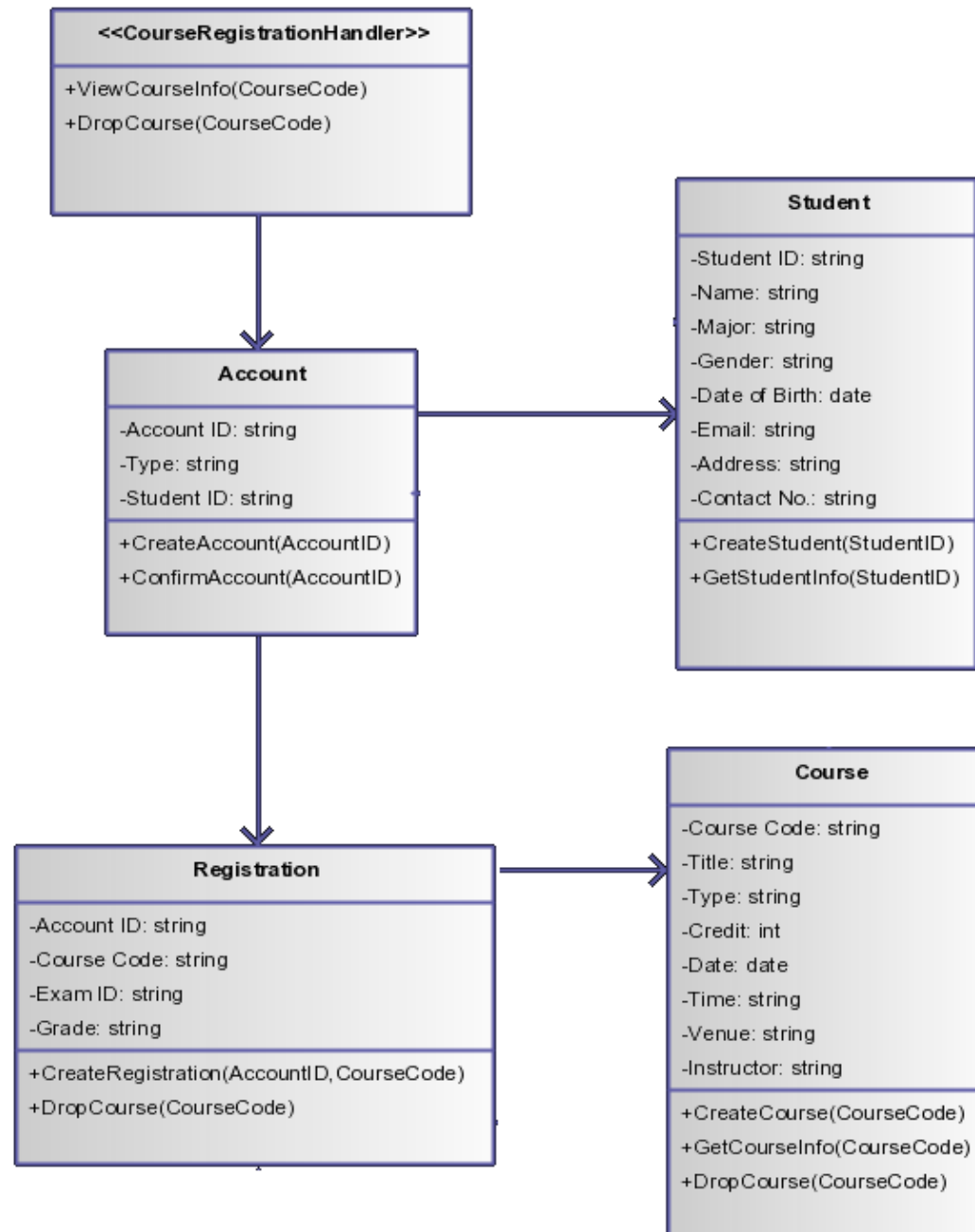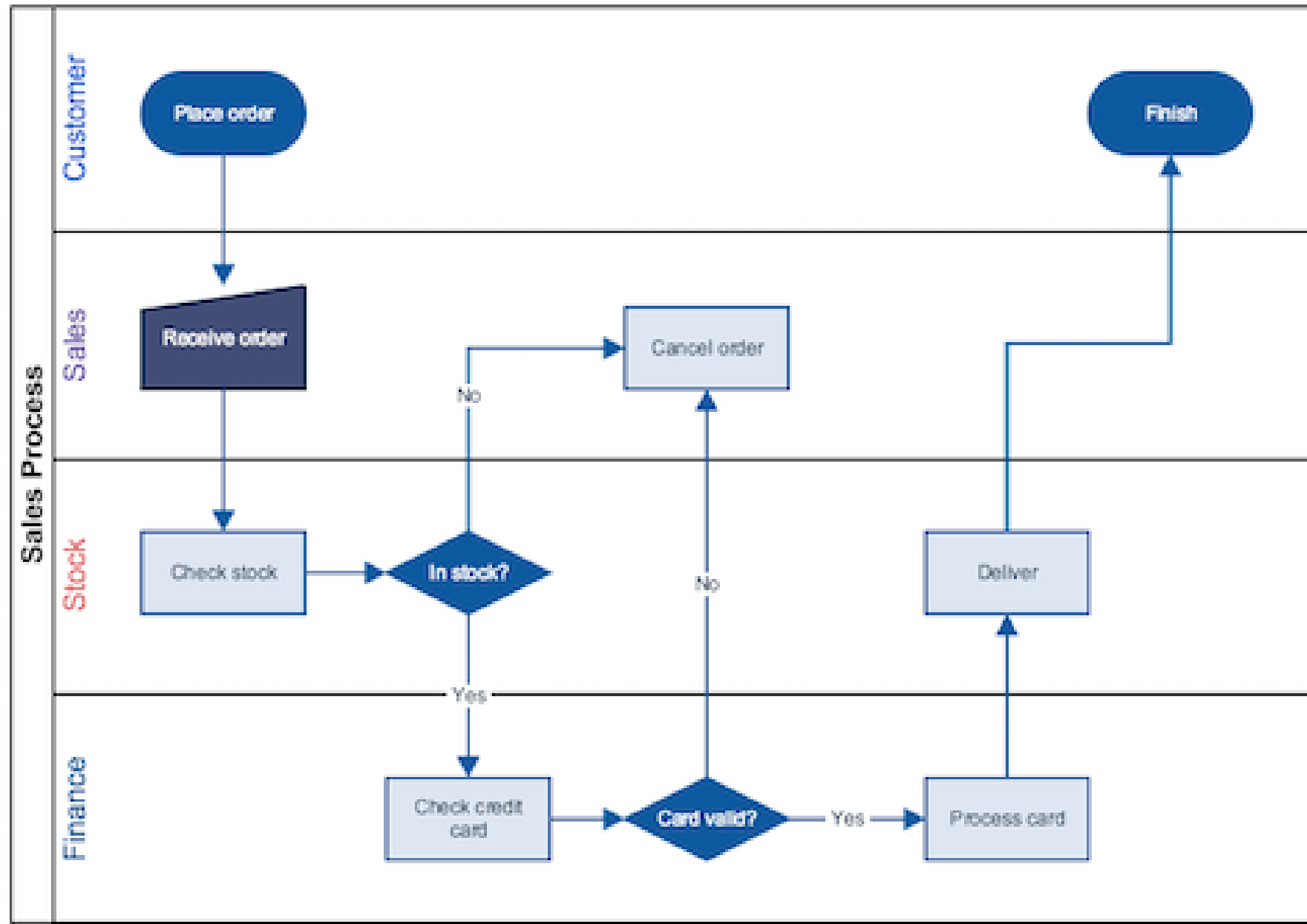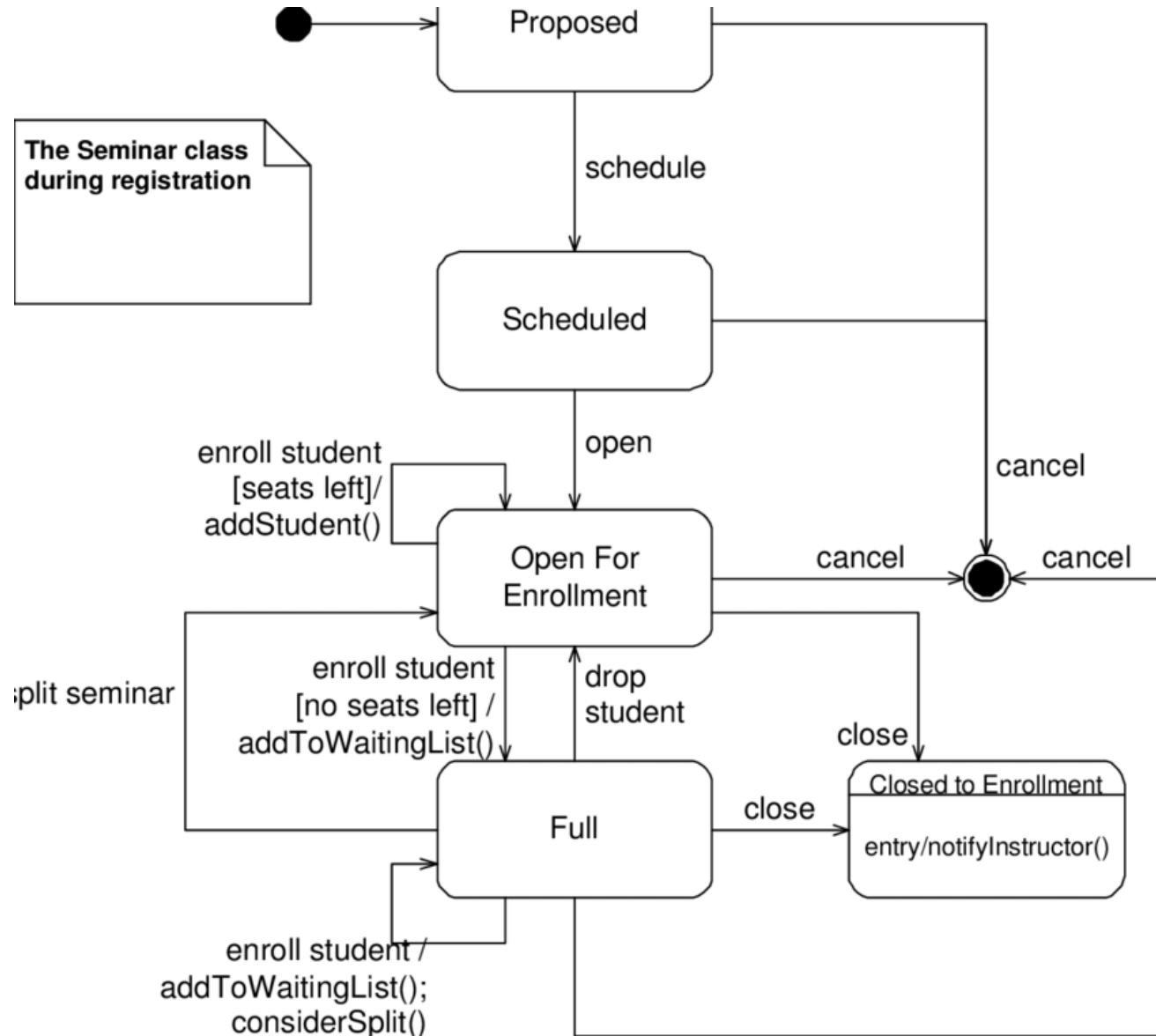
Requirements Modeling Principles

- The *information domain* of a problem must be represented and understood. i.e. capture the input of data into the system, its movement through the system, its storage and access, and the output of data to users and other systems.

- The functions that the software performs must be defined. This applies whether the functions are visible to end users or not.

- The behavior of the software as a consequence of external events must be defined. i.e. state changes as a result of input from users, network, and other sources must be represented.

- Models must be partitioned in a manner that uncovers details in a layered or hierarchical fashion. i.e. High-level models can be decomposed into more detailed models of individual components of a system or scenario.

- The analysis task should move from essential information to implementation detail. i.e. iterations of a model should include progressively more detail; initial models do not need to capture implementation details of *how* a task is accomplished, but this needs to be captured in the final model.

Identifying objects to model

- Find the nouns used in plain-English requirements statements related to the *problem domain* (the terms that would still apply in a non-software solution to the purpose of the application). These likely become objects to be modeled in class diagrams.

- Find the nouns that represent *roles* within the problem domain. These become actors who would participate in a modeled scenario.

- Find the nouns used in the *infrastructure domain* (those that refer to components of the software solution. These will not be fully defined until product design is complete). These will be labels in swimlane diagrams and nodes in scenario diagrams.

- Find the verbs used in plain-English requirements, along with abstract nouns representing *events*. These become transitions in state diagrams and methods in class diagrams.

Identifying classes – principles

- An object should be modeled as a class if it meets these 6 criteria.
- Retained information: Information about the object needs to be stored in the system in order to meet requirements.
- Needed services: Operations can and must be defined that change attributes of an object.
- Common operations: Operations apply to all instances of a class.
- Multiple attributes: If an object would only have one attribute, it should probably itself be an attribute of another object.
- Common attributes: Defined attributes apply to all instances of a class
- Essential requirements: Requirements cannot be met without the system producing or consuming information of the modeled type.

Class-Responsibility-Collaborator

- Class attributes and operations are defined as in OOP: attributes are things an object of a class *has* and operations are what it *does* – manipulate data, compute new data, or retrieve data.

- After classes are modeled with their attributes and operations, relationships between classes should be identified.

- A common way to do this is the CRC model: list each *responsibility* of a class. A class is responsible for *managing* attributes and *performing* operations. For each responsibility, list the classes that must be used to fulfill the responsibility. These are the collaborators of the modeled class.

References

- IEEE standard glossary of software engineering terminology. IEEE Standards Coordinating Committee. Sept 1990.

- Requirements Engineering. MITRE. May 2014. MITRE Systems Engineering Guide.

- Requirements Engineering: A Roadmap. Nuseibeh and Easterbrook. June 2000. Proceedings of the conference on the future of Software engineering.

- Requirement Engineering. Shobha Shivakumar. 2022. eduCBA.

- Requirements Engineering. Jill Seaman. Apr 2012. Texas State University.

- Course Registration Diagram. Creately. Aug 2022. Class Diagram Templates to Instantly Create Class Diagrams.

- Swim Lane Diagram Software. Smart Draw.

- The Object Primer: Introduction to Techniques for Agile Modeling. Scott Ambler. Disciplined Agile Consortium.

- Object Oriented Design. Edward Yourdon and Peter Coad. 1991. Prentice Hall.