Announcements, 3/7/2023

Today: Requirements and Use Case Modeling

Break around 11:15am

Acknowledgements

Some of these slides are from Prof. Alex Kuhn, and some are based on the lecture notes from Prof. Frank Tip and Prof. Michael Weintraub at Northeastern University and Prof. Emina Torlak at University of Washington and Prof. Scott Stoller at Stony Brook University, as well as Ian Sommerville's Software Engineering textbook and Martin Fowler's UML Distilled book.

The **hardest single part** of building a software system is **deciding precisely what to build**. No other part of the conceptual work is as difficult as establishing the **detailed technical requirements**…

No other part of the work so **cripples the resulting system** if done wrong. No other part is more difficult to rectify later.

- Fred P. Brooks
Received Turing Award + National Medal of Technology
Quote from No Silver Bullet — Essence and Accident in Software Engineering

Requirements analysis outline

- What are requirements?
- How do we gather requirements?
- How do we document requirements?

What are requirements?

Requirements – Definition from IEEE 610.12-1990

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 as in 1 or 2.

In summary: A description of the needs, functionality, or constraints that is focused on **what** needs to be done rather than **how** it should be accomplished.

Requirement style varies by context

- The customer can search for flights between any two airports on a given date.
- The system can handle 1,000 searches per second with an average response time of 100 ms or less.

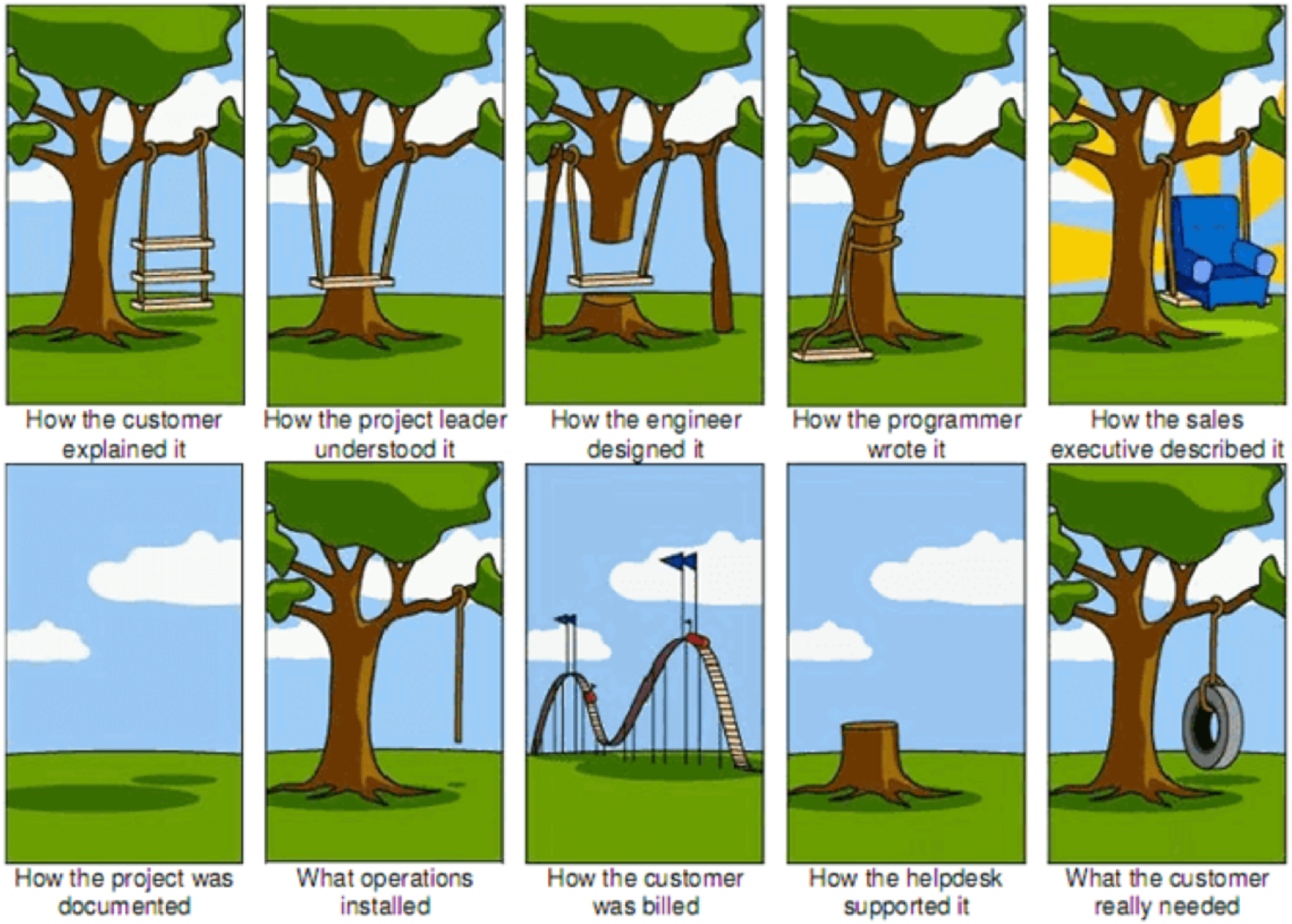
Use Case:	Book a flight
Primary Actor:	Customer
Priority	Essential
Scenario:	1. Customer is viewing a list of flights and selects "Book" for their desired flight. 2. System presents a screen to enter their name, credit card number, credit card expiration date. 3. Customer enters their name, credit card number, and credit card expiration date and confirms their details. 4. System verifies the credit card information and presents the details of their reservation. 5. Customer selects to submit the booking. 6. System submits the order to the credit card payment system. 7. Credit card payment system approves the transaction. 8. System displays the main homepage with a "Reservation successful" alert.
Extensions:	7a. Credit card transaction is denied. 7a.1 System presents an error message and proceeds back to step 2.

Why gather and document requirements?

- Understand what is needed
- Communicate the needs to everyone involved
- Ensure what is produced meets the specifications

Requirements used in many ways

- Customer – what should be delivered / contract
- Managers – schedule and determine progress
- Designers – know what to design for
- Developers – understand acceptable implementations
- QA – how to test and verify


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Example scenario

- You work for a book publisher who sells paper and electronic books.
- You are given the requirement:
  - Shipping should be free on all orders costing $50 or more
- What are questions that comes to mind about this requirement?

Note: This scenario and more great information can be found in the book, The Pragmatic Programmer (20th Anniversary Edition) by by David Thomas & Andrew Hunt

Some potential questions

- Does the $50 include tax?
- Does the $50 include current shipping charges?
- Does the $50 have to be from for all paper books? Or can it include ebooks?
- What kind of shipping is offered? Priority? Standard?
- What about international orders?
- How often will the $50 limit change in the future?

Requirements critical to success

~44% of projects found to fail due to requirements issues (Chaos Report):

- Incomplete requirements (13%)
- Lack of user involvement (12%)
- Changing requirements (9%)
- Unrealistic expectations (10%)

The numbers are % of respondents who had this opinion, out of 100%.

Types of requirements

- **Functional:** maps input to output / what the product must do to be useful
  - Allows customer to pay via credit card.
  - Customer can track individual payments.

- **Non-functional**: properties the product must have or other constraints (dependability, security, performance, accessibility, safety, etc.)
  - The product should be accessible in Korean and English.
  - Will be able to handle 1,000 searches per second.
  - Must follow governmental accessibility requirements.

Requirements vs. system design

- System design is often **more detailed** than requirements
- Functional requirement example:
  - System can backup data to cloud storage.
- System design example:
  - System provides a GUI that allows administrator to manage backups stored on Amazon S3. Full and incremental backups are supported. Backups can be created on demand or scheduled to occur periodically, every *n* days, at a user-specified time of day.
- We will discuss the design aspect more later

Characteristics of requirements

- Ensure requirements are:
  - Describing something the customer needs
  - Complete
  - Consistent
  - Realistic
  - Verifiable

Goals vs. requirements

- **Goal**
  - The system should be easy to use by the administrative staff and minimize the number of user errors made.

- **Testable non-functional requirement**
  - The administrative staff should be able to use the system after 2 hours of training. After training, on average users shall not make more than 4 errors per hour.

Metrics for specifying non-functional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

From Ian Sommerville's Software Engineering 10th Edition

Defining requirements (RFC 2119)

- ***Must*** – This word, or the terms "***required***" or "***shall***", mean that the definition is an absolute requirement of the specification.
- ***Must not*** – This phrase, or the phrase "***shall not***", mean that the definition is an absolute prohibition of the specification.
- ***Should*** – This word, or the adjective "***recommended***", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- ***Should not –*** This phrase, or the phrase "***not recommended***" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- ***May –*** This word, or the adjective "***optional***", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option ***must*** be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option ***must*** be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

https://www.ietf.org/rfc/rfc2119.txt

How do we gather requirements?

Requirements gathering

- Requirements gathering varies depending on the project context
  - Working with stakeholders and end users is key
  - Interviews are one of the best ways to elicit requirements

- Be systematic in gathering requirements

Who are stakeholders?

- People affected by the project directly or indirectly, or who can influence its outcome:
  - End users
  - Support staff
  - Decision makers around the project
  - Regulatory agencies (e.g. financial, health and safety)

- Part of the process can involve identifying **relevant** stakeholders for your project

How to determine requirements

- Interview the customer and other stakeholders
- Understand end users **needs**, **motivations**, and **behaviors**
  - Interview and observe users to learn how they work
  - Understand “why” users want to achieve their goals
- Creating prototypes can elicit more requirements
- Be aware of changing requirements over time (and expect it)

User and stakeholder involvement is key

- The Chaos Report found the #1 factor for project success was **user involvement** (surveying over 8000 projects)
- Constant iteration with users and stakeholders can help when requirements are unknown or evolving over time
- Working with users can speed up development processes and satisfy customers more

https://www.projectsmart.co.uk/white-papers/chaos-report.pdf

Gathering requirements is hard

- Difficult as stakeholders and users may:
  - Not know what they want or need and what is possible
  - Be unclear or ambiguous
  - Give conflicting requirements
  - Not understand the product context fully
  - Fail to mention “obvious” details
  - Change their needs and desires over time (particularly once they try a prototype)

Common mistakes gathering requirements

- Not being complete, consistent, or useful to what the stakeholders want
- Being too specific or detailed (e.g., describing complex business logic)
- Adding unnecessary features
  - Extra features can accumulate over time that are negative to the product (“Feature Creep”)
  - Can make the product harder to use, slow development, and have more bugs

How do we document requirements?

How to specify requirements

- Goal is to specify requirements clearly with the proper level of detail

- Many formats:
  - Contractual Style / Formal Specifications
  - Functional requirements & use cases
  - Feature lists
  - Prototypes

Use case modeling

Styles of use cases

- Informal use cases
- Formal use cases (structured text)
- Use Case Diagrams in Unified Modeling Language (UML)

Terminology

- Actor - someone that acts on the system (whether a person, organization, or another system)
- Scenario – a specific series of interactions between actors (at least one actor is a system)
- Use case – a collection of related scenarios (includes a success scenario and potentially variations or “extensions”)

Informal use case

- **Use case**: Book a flight

- The customer can book a flight from the list of flights they have found while searching for a flight. Upon selecting a flight, they will be taken to a new page and be prompted to enter their name, credit card number and expiration date. The customer can then review the entire order and needs to submit the booking for the order to be finalized.

Functional requirements vs. use cases

- Typically write functional requirements and then use cases
- Use cases are **more operational** and typically **more detailed** than functional requirements.
- Example: Check-out (for an online store)
  - Functional Requirement: Customer can check-out by providing payment information and confirming the order.
- Use Case:
  - System displays shopping cart and prompts user to select payment method: credit card or gift card.
  - User selects payment method.
  - If payment method is credit card, system prompts user for credit card information. If payment method is gift card, ...
  - ...

Why are use cases useful?

- Clarifies and define functional requirements
  - Determine what steps lead to successful scenarios
  - Showcases alternative scenarios of use
  - Describes common errors and issues and how the system should respond

Writing good use cases

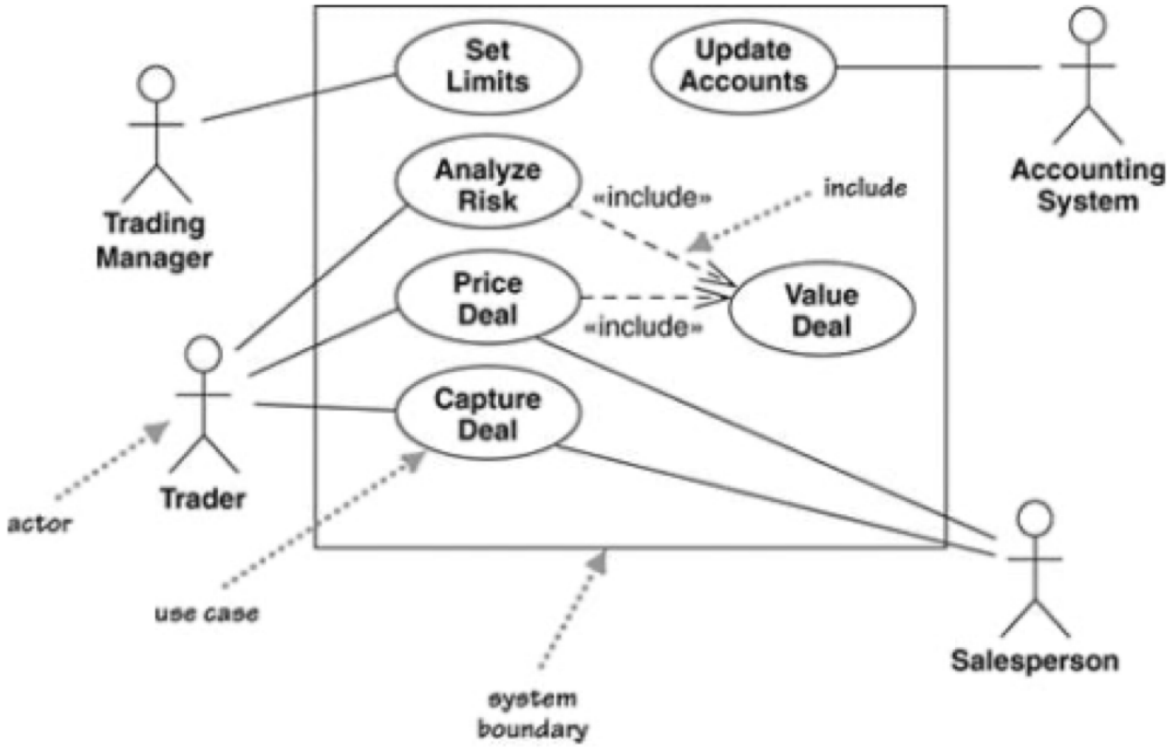
- Be concise and clear to developers and non-developers
- Focus on interactions and essential behaviors. Write from the user's point of view
- Omit inessential implementation details
  - The system saves the order information.
  - 🚫 The system saves the order information in a relational database.
- Omit GUI Details (GUI is important but described separately)
  - User selects payment method.
  - 🚫 User selects payment method from drop-down list.

Writing good use cases

- Express error cases as extension flows, not separate use cases.
- Include interesting application-specific error cases. Omit uninteresting generic error cases.
  - Transaction denied due to insufficient funds.
  - 🚫 Invalid format for telephone number.
- Omit detailed specifications of data and algorithms (they belong elsewhere). Focus on actions.
  - Customer enters new password.
  - 🚫 Customer enters new password. A password must contain 8 to 20 characters, including at least one uppercase letter, one lowercase letter, and one number.

Use case diagrams

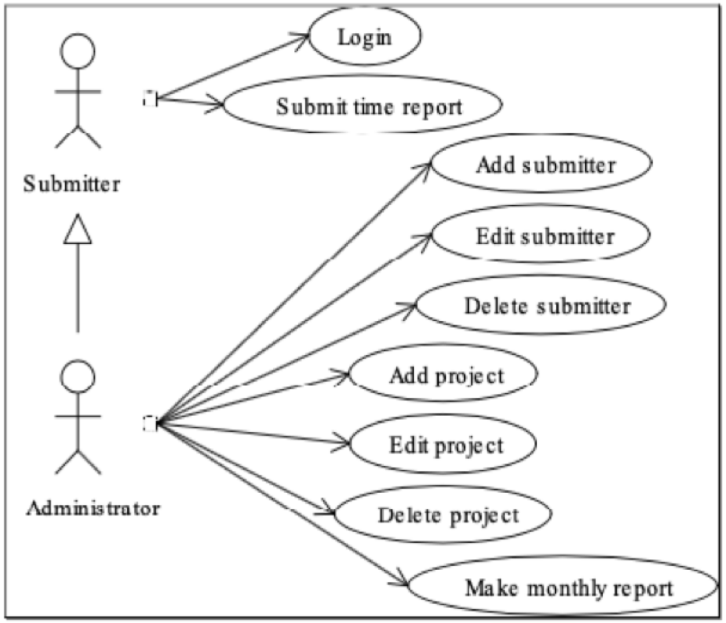
- Provides an overview of use cases, describing:
  - Which actors carry out what use cases
  - Which use cases include other uses cases



From Martin Fowler's UML Distilled

Use case diagrams

- Can show inheritance if an actor can do everything another actor does



Actor	Goal
Submitter	Log in
	Submit time report
Administrator	Log in
	Submit time report
	Add submitter
	Edit submitter
	Delete submitter
	Add project
	Edit project
	Delete project
	Make monthly report

- Note that use case diagrams can also be represented in a table

From Alistair Cockburn's "Writing Effective Use Cases"

Methodology for creating use cases in 416

1. Write out all of your functional requirements (including stretch goals)
   - Submit functional requirements to the instructor for feedback before proceeding
2. Identify the major use cases by name and actors involved
3. Write the primary scenario for each use case
4. Add any extension scenarios
5. Revise as needed

Use case practice

Exercise 1

- A chain of brick-and-mortar bookstores wants to add on-line book sales. The new Internet Sales System (ISS) should allow on-line orders to be fulfilled by mail or in-store pickup.

- Vendors (wholesalers that sell books to the chain) should be able to supply marketing materials (such as reviews and author biographies) directly to the ISS.

- The ISS must interface with the existing IT system. The existing system has a database of book information (title, author, price, inventory, etc.). It supports in-store sales and has a mail-order subsystem that supports orders placed by phone and fulfilled by mail. The mail-order subsystem has a database of mail orders.

Your task

- Download `exercise1-use-cases.txt` for the full requirements
  - On shared Google Drive folder under In-class Activities

- Write use cases (include primary scenario with the flow of events, etc.) for **customer** use cases related to on-line shopping
  - Do not write textual descriptions for other use cases (e.g., vendors supplying marketing information).

- One person per team email me your use cases (cc'ing all members of your group)
  - Submit before the end of the day

Questions?