Week 2 - Wednesday

COMP 3100

Last time

- What did we talk about last time?
- Continued introduction to software engineering
- Git, GitHub, and version control systems

Questions?

Requirements

Terminology

- A **requirement** is a property or behavior a product must exhibit
- A **specification** is a precise description
- A **requirements specification** is (unsurprisingly) a precise description of the properties or behaviors a product must have

Stakeholders

- **Stakeholders** are anyone affected by a product or its development
  - **Customers** are the people that pay for a product
  - **Users** are people who interact with the product
  - **Clients** are people for whom software was created (includes both customers and users)
  - **Developers** are all the people who work on the project
  - **Regulators** are responsible for ensuring that software meets standards
  - **Marketers** stand in for clients when making mass-market products

Stakeholder needs

- All the stakeholders have needs, but not all of these needs can or should be requirements
- Stakeholder needs are frequently in conflict
  - It's a pain to put good privacy controls into a product
  - But regulators might require it
- Stakeholder needs are incomplete and sometimes incorrect
- Stakeholders needs are often abstract
  - "The game should be awesome."

More on stakeholder needs

- A **stakeholder need** is a feature that one or more stakeholders want
- Sometimes, these needs are written in descriptions called **needs specifications**
- Then, developers have to wrangle all of these conflicting, incomplete, and vague needs into a requirements specification
- Traditional methods may have a specific person who does this
  - Titles like **requirements analyst**, **requirements specialist**, **user interaction designer**

Functional and non-functional requirements

- It's common to divide requirements into functional and non-functional categories
- **Functional requirements** are about how software takes input and turns it into output, its **behavior**
  - Appearance
  - User interface actions
  - Input and output processes
- Most requirements are functional requirements, and they take the most time and effort to specify

Non-functional requirements

- **Non-functional requirements** describe the **properties** software must have
  - Speed of processing
  - Amount of memory used
  - How often failures can be permitted
  - Level of security
  - Ease of modification
  - Cost of development
  - Platforms the product must run on
- Non-functional requirements are more abstract than functional requirements
- Functional requirements are tied to specific pieces of code, but non-functional requirements are properties of the whole system

Abstraction in specification

- This example from the book shows the same specification from abstract to concrete:
  - The product must provide an online bookstore shopping experience.
  - The product must provide titles and descriptions of books from which users may choose books to purchase.
  - The product must allow shoppers to buy books by placing them in a shopping cart.
  - The product must display a button with each book description that when pressed places books in the shopper's cart.
  - The product must display a button labeled "Buy" that maintains a constant position (despite scrolling) at the upper right-hand corner of a book-description web page. When pressed, this button must place the book whose description is displayed into the shopper's cart.

Levels of abstraction

- **Business requirements specifications** are client objectives that must be met
  - They are abstract descriptions of the product
  - They might include deadlines or sales targets
  - They're usually non-functional requirements
- **User-level needs and requirements** are one step more concrete
  - They describe tasks or goals that the product would allow a user to perform
  - These tend to describe what the product does but not how
  - Could be functional or non-functional

More levels of abstraction

- **Operational-level needs and requirements** describe individual inputs and outputs
  - These are more specific than user-level needs and requirements
  - Example: The product must allow users to enter polynomial equations, trigonometric equations, logarithmic equations, and exponential equations, all of one variable.
- **Physical-level needs and requirements** are about the appearance and formatting of the user interface
  - These will describe what the product actually looks like (which might be several different descriptions if the same product works on a desktop and a phone)
  - Example: The product must provide a text box for equation input. The text box must display 50 characters and scroll vertically up to 800 lines.

Requirements in traditional processes

- Most of the terminology we've been using comes out of traditional software development processes like waterfall
- In that paradigm, requirements must be gathered first, followed by design, followed by implementation, testing, and maintenance
- The requirements must be **frozen** so that the next steps can take place
- Then, no one wants to change the requirements because you'll have to redo everything that comes after, which is expensive

Problems with requirements in traditional processes

- It's really hard to figure out all the requirements before doing any coding and looking at prototypes
- The world changes quickly, especially in technology, and people's desires change
- Writing all the requirements takes a lot of work, creates large documents, and costs a lot of money
- The waterfall process means that nothing is ready for a long time (often years) after the project starts, and some projects get canceled

Requirements in agile processes

- Agile developers try not to write requirements at all
  - But you have to start with something…
- Stakeholder needs are turned into lists called **product backlogs**
- A **product owner** adds to the product backlogs and prioritizes them
- High priority items are chosen for each **sprint**, the agile term for a development iteration

How Scrum tries to make changing requirements cheap and easy

- Delay choosing requirements as long as possible
  - Stakeholder needs can be easily added or removed from the product backlog
  - Requirements are set only for the product backlog items (PBIs) when they're implemented on a sprint
- Delay refinement as long as possible
  - PBIs are broken down until they're small enough and detailed enough for a single sprint
  - User-level requirements are refined into operational- and physical-level requirements for the sprint where they're implemented
- Avoid writing requirements altogether
  - Instead of writing down physical-level requirements, talk to the stakeholders and implement what they say in the sprint
- Determine requirements in light of current product features
  - Because agile methods iterate on an existing product, everyone can see which features would be most useful next

Stating specifications in traditional processes

- Specifications are usually made in declarative English (or appropriate natural language) sentences
- Problem: English is vague and confusing
- Rules for good technical writing:
  - Write complete, simple sentences in the active voice
  - Define terms clearly and use them consistently
  - Avoid synonyms
  - Group related material into sections
  - Use tables, lists, indentation, white space, and other formatting aids
- Use "must" or "shall" to describe behaviors the product must do

Testable requirements

- Requirements should be **testable** or **verifiable**
- This means that there can be a process for testing whether the product meets the requirement
- **Bad requirement:**
  - The product must display query results quickly.
- **Good requirement:**
  - The product must display query results in less than one second.
- The bad requirement isn't testable because "quickly" is subjective
- The good requirement is testable because we can time the finished system

Requirements traceability

- We want a clear relationship between a requirement, a part of the design, the code that implements this design, and the tests that verify it
- Being able to connect the requirements to later stages of development is called **requirements traceability**
- To make requirements more traceable, each specification should state only a single requirement
  - This kind of specification is called **atomic**
- **Non-atomic specification:**
  - The product must display a list of previous commands and the results of commands, each in its own window.
- The goal is simplicity and clarity
- A long list of simple requirements is better than a short list of confusing, complex requirements

Stating specifications in agile processes

- Agile developers have some documents like product vision statements and product backlog items
- A very common way to describe requirements is through **user stories**
- A user story describes a function that the product provides to users
- Sometimes a big story that is a huge chunk of the application is called an **epic**
- Sometimes a story that would take several sprints to implement is called a **feature**
- A story that can be implemented in a single sprint is a **sprintable story** or an **implementable story**
- **Note:** Some agile people *only* use the term user story for sprintable stories

User voice form

- A common way of expressing user stories is user voice form:
  - As a <*role*>, I want to <*activity*> so that <*benefit*>.
  - <*role*> is replaced by a user role, which is some category of user
  - <*activity*> is a function that the system does
  - <*benefit*> shows the value of the activity but is an optional part of user voice form
- Example:
  - As a payroll clerk, I want to enter salary data so that payrolls will use adjusted salaries.

Good user stories

- Like traditional requirements specifications, user stories should be testable
- Thus, sprintable user stories should have **acceptance criteria** or **conditions of satisfaction** that say what product behavior will count as satisfying the user story
- Some methods use the three Cs of user stories:
  - **Card:** An index card (or virtual equivalent) with a user story title and the user story itself on its front
  - **Conversation:** Discussions about the user story, which might not be written in any way, because agile emphasizes face-to-face discussion
  - **Confirmation:** Acceptance criteria for the user story, written on the back of the card

Eliciting stakeholder needs in traditional processes

- It can be difficult to discover what stakeholders actually want from a product
- Some approaches:
  - **Interviews:** Ask individual stakeholders what they want and record the answers
  - **Observation:** Watch the users doing tasks, asking them to describe the actions they're taking
  - **Focus groups:** Informal discussion with six to nine people and a facilitator
  - **Workshops:** A meeting focused on documenting the desires of many stakeholders
  - **Prototypes:** Let stakeholders respond to different versions of a product
  - **Document studies:** Read documents associated with the business that needs the product
  - **Competitive product studies:** Analyze similar existing products for strengths and weaknesses

Eliciting stakeholder needs in agile processes

- Agile processes don't focus on getting all the requirements up front
- Instead, a cornerstone of the agile approach is constantly getting feedback, allowing for quick responses
- The product itself becomes an evolving prototype that it easy to understand and unlikely to become obsolete
- Potential problems:
  - Stakeholders can overreact to current problems and lose sight of the big picture
  - Agile methods give a lot of power to the few stakeholders who give feedback, and others might be ignored

Upcoming

Next time…

- Work day on Friday
- Next Monday:
  - Verifying and validating requirements
  - Requirements management
  - Requirements and design
  - Requirements modeling and UML

CAREER JUMPSTART EVENT

Engineering & Computer Science

THURSDAY, SEPTEMBER 12TH FROM 4:45PM-7PM

Otterbein University @ The Point

Come and network with alumni and recruitment partners and learn how to be successful with your field.

OTTERBEIN
UNIVERSITY

SCAN the QR CODE to REGISTER

Reminders

- Keep reading Chapter 5: Software Product Requirements for Monday
- Keep working on your projects
  - Project 1 (Draft) is due next Friday, September 13