CSE 403

Software Engineering

Spring 2023

#5: Requirements

Project assignments are done!

Logistics

WEEK 2		
04/03	L: Dev. Cycle	DUE: PP_1.1!!!
04/04	T: Proposals	DUE: PP_1.2!!!
04/05	L: Requirements	Project Requirements (PR)
04/06	P: Requirements	
04/07	L: Use-Cases	
WEEK 3		
04/10	L: SCRUM	
04/11	T:	DUE: PR!!!
04/12	L: Version Control	GitHub Project Setup (GPS)
04/13	P:	
04/14	LX: GIT	

Requirements

Recap: Life-cycle stages

Virtually all SDLC models have the following stages:

- **Requirements ← Our focus this week**
- Design
- Implementation
- Testing
- Maintenance

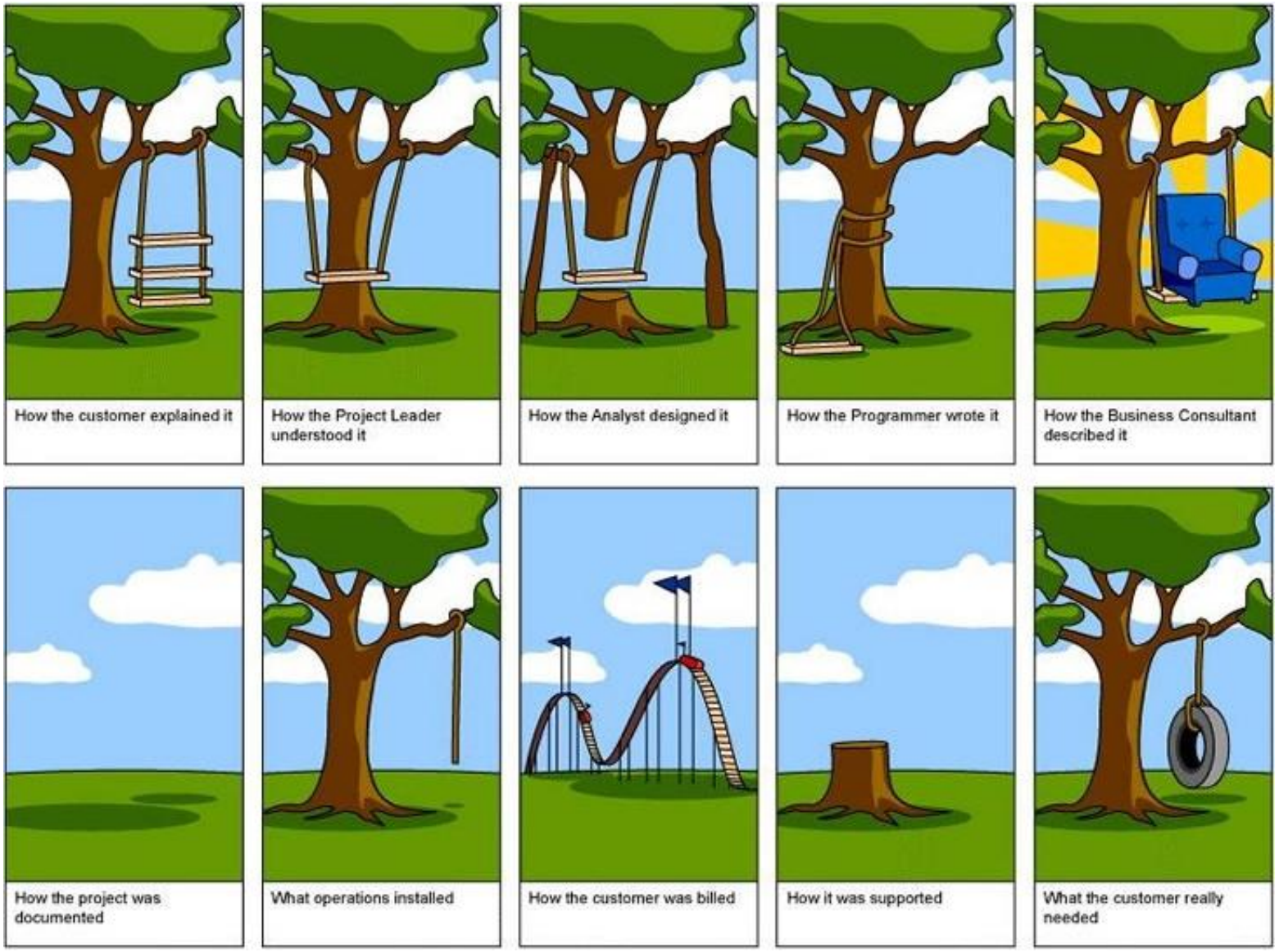
Traditional models:

- Waterfall, Prototyping, Spiral, etc.

Agile models:

- *eXtreme Programming, Scrum, etc.*

Requirements in one picture



Software requirements

Requirements specify what to build

- describe **what, not how**

Software requirements

Requirements specify what to build

- describe **what, not how**
- describe the problem, not the solution

Software requirements

Requirements specify what to build

- describe **what, not how**
- describe the problem, not the solution
- reflect system design, not software design

Software requirements

Requirements specify what to build

- describe **what, not how**
- describe the problem, not the solution
- reflect system design, not software design

Let's try this out?

Software requirements

Requirements specify what to build

- describe **what, not how**
- describe the problem, not the solution
- reflect system design, not software design

How did it go?

“What” vs. “how” is relative

One person’s **what** is another person’s *how*:

- Search for students is the **what**
  - JS.search() is the **how**
- JS.search() is the **what**,
  - binary search is the **how**
- To implement a binary search…

“What” vs. “how” is relative

One person’s **what** is another person’s *how*:

- Search for students is the **what**
  - JS.search() is the **how**
- JS.search() is the **what**,
  - binary search is the **how**
- To implement a binary search…

How to solve this?

Goals of Requirement elicitation

- **Understand** precisely what is required of the software.
- **Communicate** this understanding precisely to all involved parties.
- **Control** production to ensure that system meets specification.

Goals of Requirement elicitation

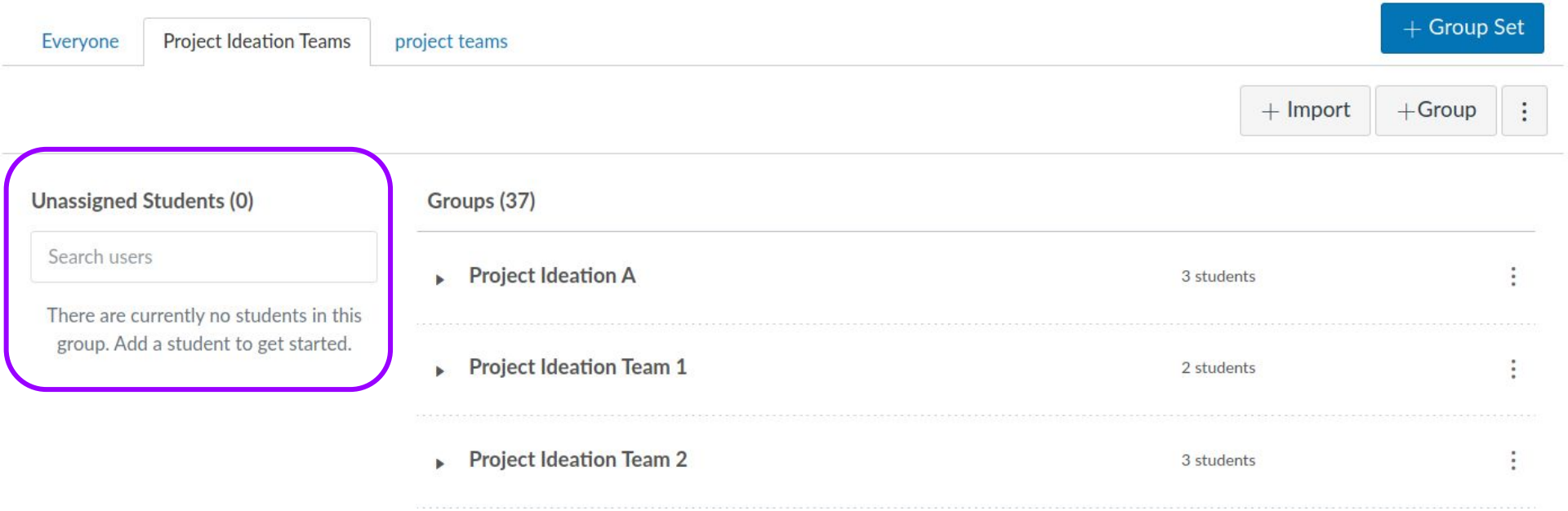
- **Understand** precisely what is required of the software.
- **Communicate** this understanding precisely to all involved parties.
- **Control** production to ensure that system meets specification.

Example: Search for individual students on the groups page.

Goals of Requirement elicitation

- **Understand** precisely what is required of the software.
- **Communicate** this understanding precisely to all involved parties.
- **Control** production to ensure that system meets specification.

Example: Search for individual students on the groups page.



Requirements' Roles

- **Customers**: what should be delivered (contractual base).
- **Managers**: scheduling and monitoring (progress indicator).
- **Designers**: a spec to design the system.
- **Coders**: a range of acceptable implementations.
- **QA / Testers**: a basis for testing, verification, and validation.

Requirements' Roles

- **Customers**: what should be delivered (contractual base).
- **Managers**: scheduling and monitoring (progress indicator).
- **Designers**: a spec to design the system.
- **Coders**: a range of acceptable implementations.
- **QA / Testers**: a basis for testing, verification, and validation.

Keep the WHAT and HOW in mind!

How to elicit requirements?

Do:

- Talk to the users -- to learn how they work.
- Ask questions throughout the process -- "dig" for requirements.
- Think about why users do something in your app, not just what.
- Allow (and expect) requirements to change later.

How to elicit requirements?

Do:

- Talk to the users -- to learn how they work.
- Ask questions throughout the process -- "dig" for requirements.
- Think about why users do something in your app, not just what.
- Allow (and expect) requirements to change later.

Don't:

- Be too specific or detailed.
- Describe complex business logic or rules of the system.
- Describe the exact user interface used to implement a feature.
- Try to think of everything ahead of time. (You will fail!)
- Add unnecessary features not wanted by the customers.

Strategies for eliciting requirements

Common strategies

- Interviews
- Observations
- **Use cases**
- Feature list
- Prototyping (e.g., UI)

Cockburn's requirements template

1. Purpose and scope
2. Terms (glossary)
3. **Use cases (the central artifact of requirements)**
4. Technology used
5. Other
    a. Development process: participants, values (fast-good-cheap), visibility, competition, dependencies
    b. Business rules (constraints)
    c. Performance demands
    d. Security, documentation
    e. Usability
    f. Portability
    g. Unresolved (deferred)
6. Human factors (legal, political, organizational, training)

See file on Canvas for comprehensive write up and examples.

Requirements engineering

The process of eliciting, analyzing, documenting, and maintaining requirements.

One way to classify requirements

- Functional requirements
  - E.g., input-output behavior
- Non-functional requirements
  - E.g., security, privacy, scalability
- Additional constraints
  - E.g., programming language, frameworks, testing infrastructure

Requirements engineering

The process of eliciting, analyzing, documenting, and maintaining requirements.

One way to classify requirements

- Functional requirements
  - E.g., input-output behavior
- Non-functional requirements
  - E.g., security, privacy, scalability
- Additional constraints
  - E.g., programming language, frameworks, testing infrastructure

Requirements engineering

The process of eliciting, analyzing, documenting, and maintaining requirements.

One way to classify requirements

- Functional requirements
  - E.g., input-output behavior
- Non-functional requirements
  - E.g., security, privacy, scalability
- Additional constraints
  - E.g., programming language, frameworks, testing infrastructure

Requirements engineering

The process of eliciting, analyzing, documenting, and maintaining requirements.

One way to classify requirements

- Functional requirements
  - E.g., input-output behavior
- Non-functional requirements
  - E.g., security, privacy, scalability
- Additional constraints
  - E.g., programming language, frameworks, testing infrastructure

Challenges and common mistakes

Challenges

- Unclear scope and unclear requirements.
- Changing/evolving requirements.
- Finding the right balance (depends on customer):
  - Comprehensible vs. detailed.
  - Graphics vs. tables and explicit and precise wording.
  - Short and timely vs. complete and late.

Common Mistakes

- Implementation details instead of requirements.
- Projection of own models/ideas.
- Feature creep/bloat.

Feature creep/bloat

Feature creep:

- Gradual accumulation of features over time.
- Often has a negative overall effect on a large software project.

Why does feature creep happen? Because features are fun!

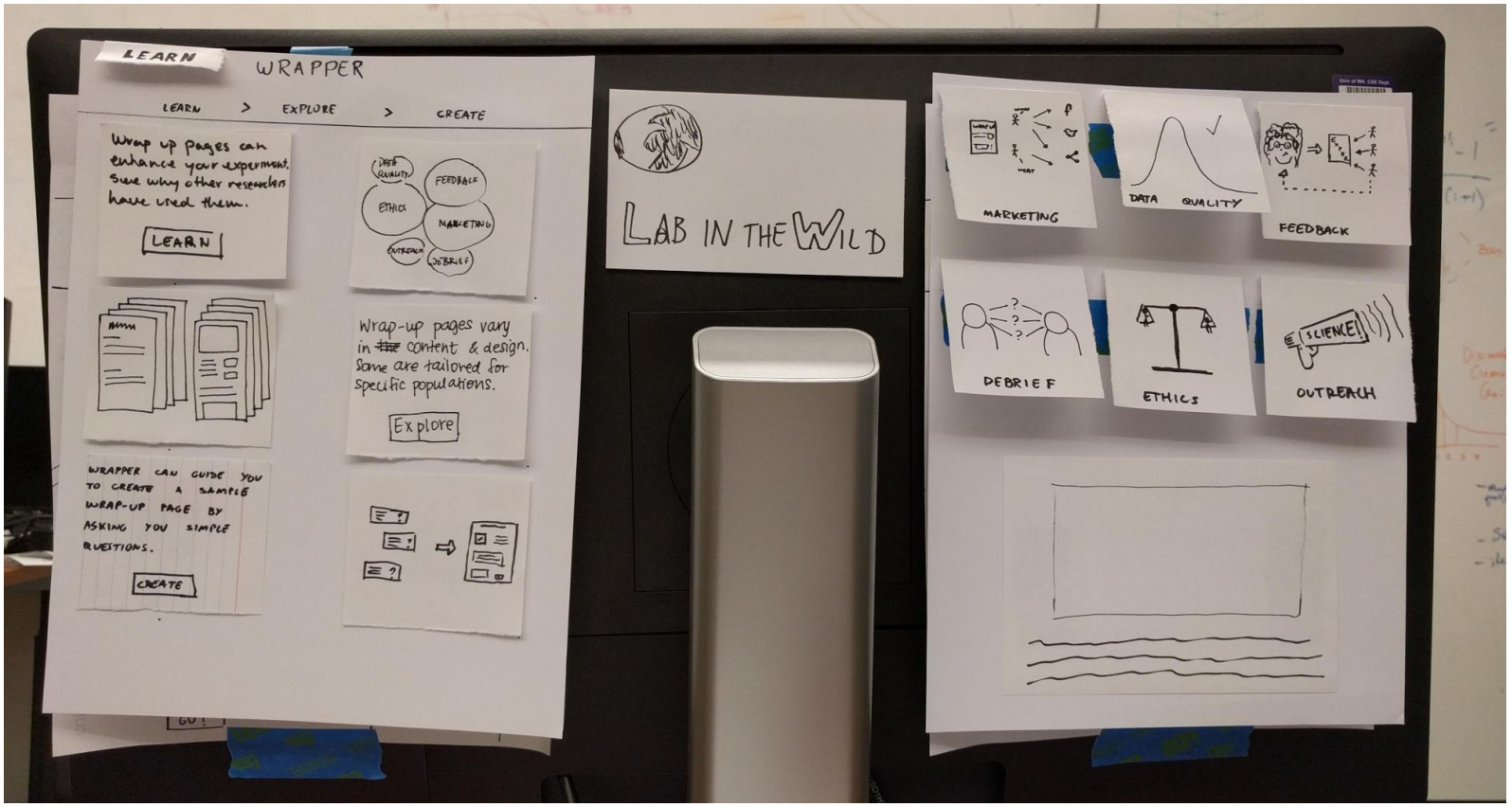
- Developers like to code them.
- Sales teams like to brag about them.
- Users (think they) want them.

Why is it bad?

- Too many options, more bugs, more delays, less testing, …
- "Boiled frog" analogy.

Can you think of any products that have had feature creep?

Requirement testing challenge!



What’s next?

WEEK 2		
04/03	L: Dev. Cycle	DUE: PP_1.1!!!
04/04	T: Proposals	DUE: PP_1.2!!!
04/05	L: Requirements	Project Requirements (PR)
04/06	P: Requirements	
04/07	L: Use-Cases	

What’s next?

WEEK 2		
04/03	L: Dev. Cycle	DUE: PP_1.1!!!
04/04	T: Proposals	DUE: PP_1.2!!!
04/05	L: Requirements	Project Requirements (PR)
04/06	P: Requirements	
04/07	L: Use-Cases	

Question, please!