Software Specifications

17-313: Foundations of Software Engineering

https://cmu-313.github.io

Josh Sunshine and Michael Hilton

Fall 2026

Gentle Reminder

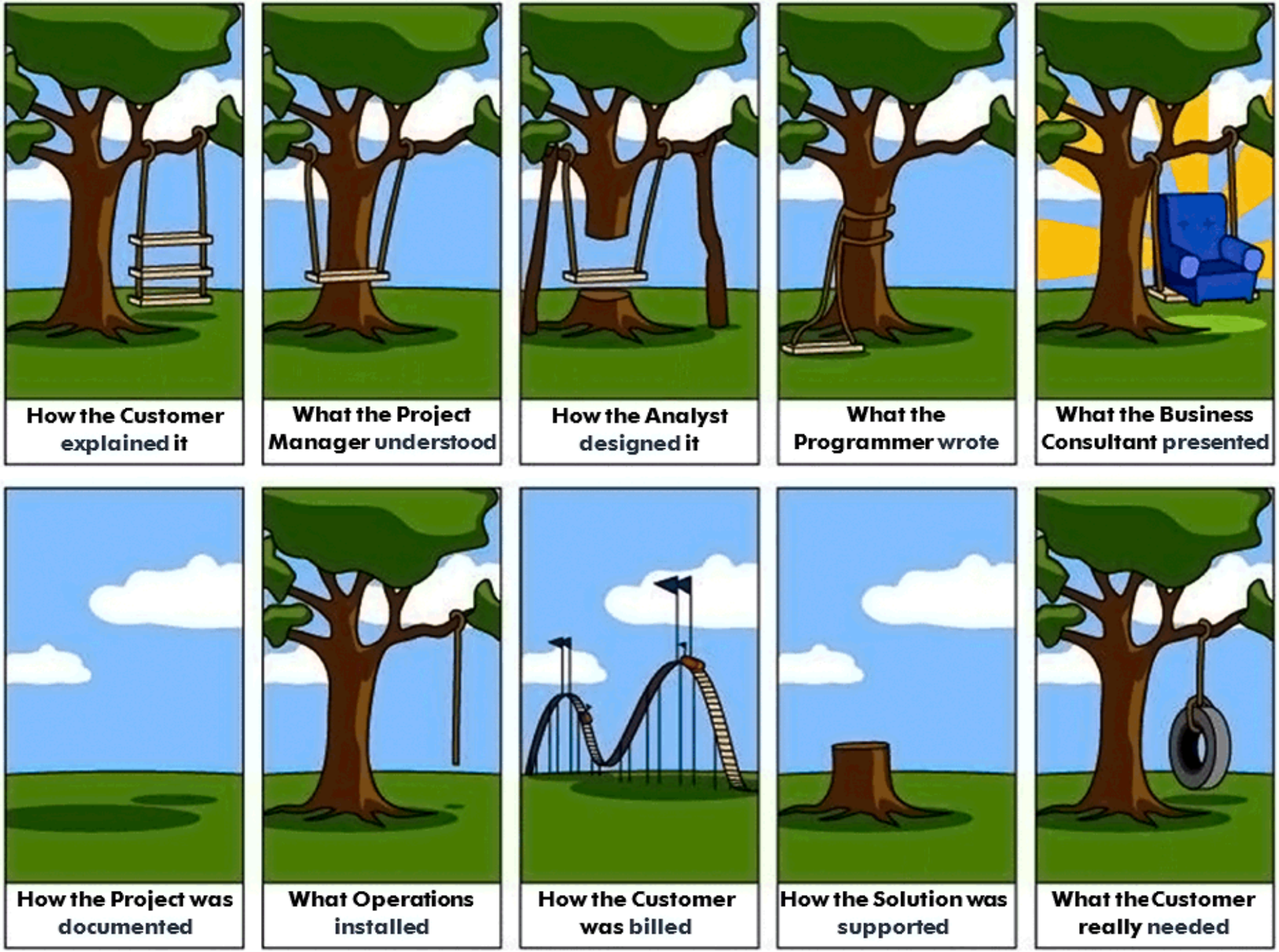
- Please close laptops
- Tablets with stylus ok

Administrivia

- Problem Set 1 on Canvas, due September 2 @ 11:59pm
- Quiz on Thursday
- Reading and participation activity due ***before*** class on Thursday

Today's Learning Goals

- Recognize the importance of process
- Identify why software development has project characteristics
- Understand the elements of Scrum
- Create and evaluate user stories
- Use milestones for planning and progress measurement
- Understand the difficulty of measuring progress

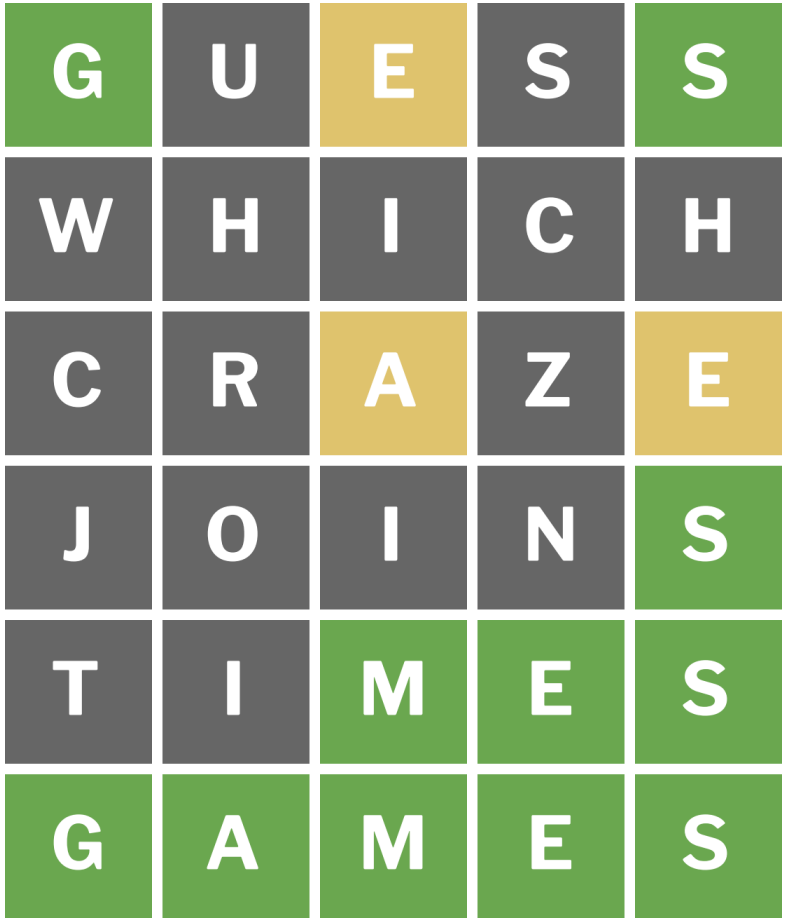

How the Customer explained it
What the Project Manager understood
How the Analyst designed it
What the Programmer wrote
What the Business Consultant presented
How the Project was documented
What Operations installed
How the Customer was billed
How the Solution was supported
What the Customer really needed

What are specifications?

- Establish the basis for an agreement between customers and software engineers on how the software product should function
- Provide a realistic basis for estimating product costs, risks, and schedules
- To derive the specifations, the engineers need to have a clear and thorough understanding of the products under development

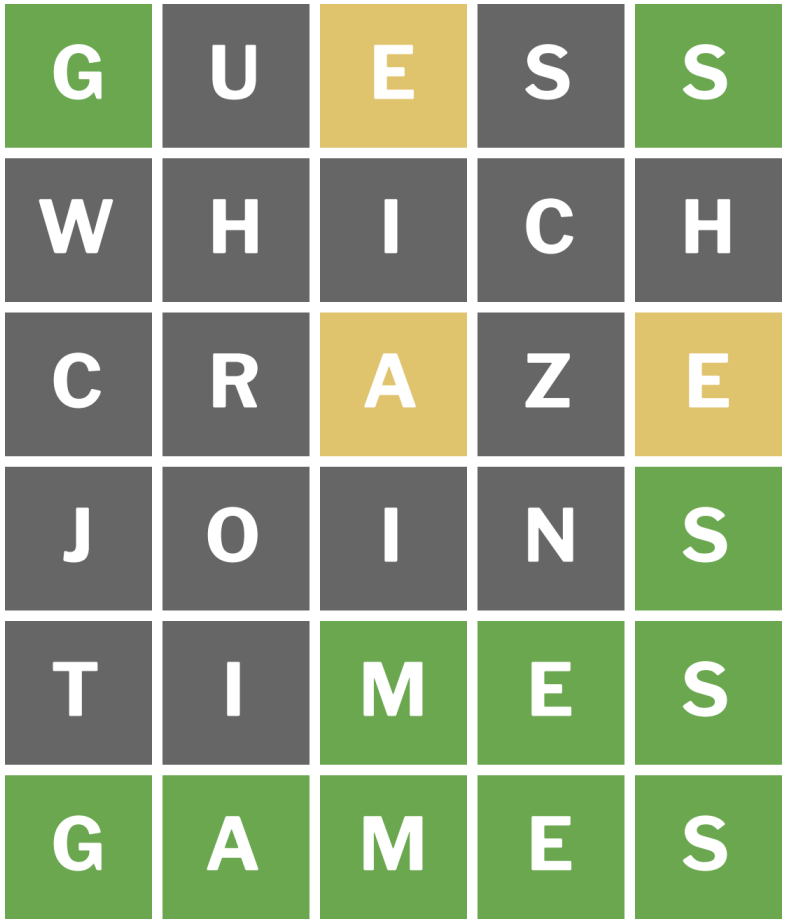
Example: Wordle

- Guessing game
- User guesses a 5-letter English word
- After each guess, the tiles change color to show how close guesser is to secret word



Wordle Rules

- **Green:** The letter is in the word and in the **correct spot.**
- **Yellow:** The letter is in the word but in the **wrong spot**.
- **Black:** The letter is **not in the word** in any spot.



Making functions easy to test

- Side-effect free
- Determinism
- Well-defined input and output sets
- Small input and output sets

Activity: Write Wordle Functions

- Write the signature of at least five functions that can be used to implement Wordle
- Isolate hard-to-test functions (easy to test: Side-effect free, Determinism, Well-defined input and output sets, Small input and output sets)

Key rules:

- **Green:** The letter is in the word and in the **correct spot.**
- **Yellow:** The letter is in the word but in the **wrong spot**.
- **Black:** The letter is **not in the word** in any spot.

Activity 2: Natural Language Specs

- Write a prompt to generate the wordle functions you created before
- Find an example where the function's behavior is very different from between the two LLMs.
- Write down the differences

Common Properties: Invariants

- Something **stays the same** or **within allowed bounds**
  - Sorting: same multiset of elements (i.e., nothing is added or dropped)
  - Sets and strings: |AB| = |A| + |B|
  - Cart totals: never negative
  - Cart: membership tier discount never increases price!

Common Property: Metamorphic

- Property holds under **input transformations**
  - Sorting: reversing/shuffling input does not change the sorted output
  - Cart: reordering items does not change the total
  - JSON parsing: reordering object keys produces the same parsed result
  - Shipping: changing address / ZIP doesn’t affect the price

Common Property: Differential

- Two implementations (or versions) **agree on outputs and errors**
  - Slow but trusted reference solution vs. optimized version
  - Old vs. new implementation after refactoring
  - Third-party library vs. your code

Conditional Properties

- **If** condition A is met, **then** property B must hold.
  Mathematical logic: $A \Rightarrow B$
- Examples:
  - **Wordle**: If the guess word matches the secret word the output should be (G,G,G,G,G).
  - **Compilers**: If the program includes an invalid keyword, the parser should throw a parse error.

Conditional Properties Approach 1: Filtering

Most PBT frameworks allow you to "discard" inputs that don't meet your criteria. In fast-check you use the fc.pre() method.

- **How it works:** The generator creates a random withdrawal amount. If that amount is ≤ the balance, the test framework simply throws that case away and tries again with a new random number.
- **The Pro:** It's very easy to write.
- **The Con:** It can be inefficient. If your condition is very rare (e.g., "only when the amount is exactly 1,000,000.03"), the framework might exhaust its retry limit before finding enough valid cases.

Conditional Properties Approach 2: Targeted Generators

Instead of generating *any* number and throwing away the bad ones, you create a custom generator that *only* produces "invalid" inputs.

- **How it works:** You define a generator specifically for the error case: amount = random_int(min=balance + 1).
- **The Pro:** Every single test run is a "hit." It's incredibly efficient and ensures you are testing the error handling thoroughly.
- **The Con:** It requires more setup code and a deeper understanding of your data constraints.

Conditional Properties Approach 3: Branching Properties

- **How it works:** The test itself has branches (e.g. if and else) and the property is defined across the branches.
- **The Pro:** This creates a "complete" specification of the function in one place.
- **The Con:** It can make the test code more complex and harder to debug if it fails.