

Software Testing

17-313: Foundations of Software **Engineering**

<https://cmu-313.github.io>

Michael Hilton and Josh Sunshine

Fall 2026

Learning Goals

- Distinguish between verification and validation
- Articulate the value of testing
- Evaluate trade-offs in the testing pyramid
- Design and refactor code for testability

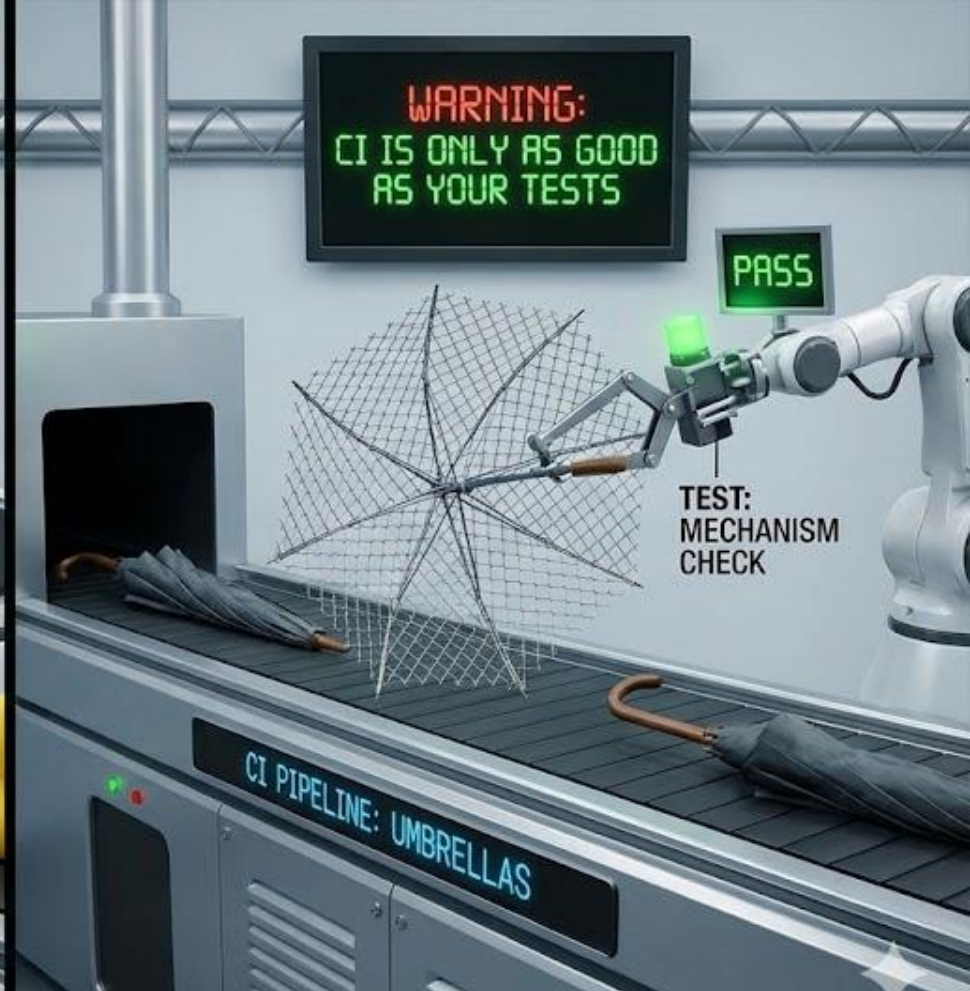
Project 2B and 2C

- Project 2B Due Monday, September 28 @ 11:59pm
 - We expect this to be challenging. PLEASE DON'T WAIT TILL THE LAST MINUTE TO START
- Problem set 3 is out on Canvas. It's due Tuesday (tonight), quiz in-class Thursday
 - Note: on Wednesday morning we will post an answer key for problem set 3 so that you can study for Thursday's quiz. One important note: The questions on Thursday's quiz will **not** be exact duplicates of the problem set problems but the questions will be conceptually similar.

Gentle Reminder

- Please close laptops
- Tablets with stylus ok

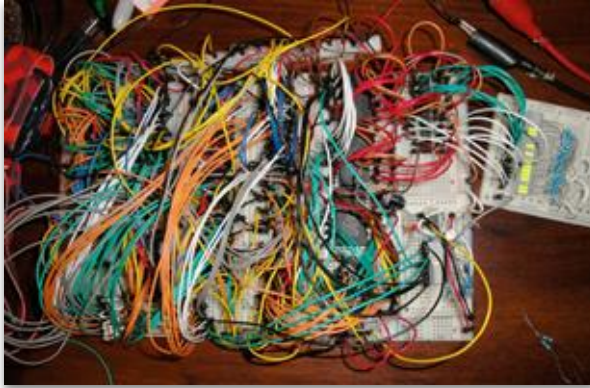




Software Quality



Internal Quality



- Is the code well structured?
- Is the code understandable?
- How well documented?

External Quality



- Does the software crash?
- Does it meet the requirements?
- Is the UI well designed?

Testing Focuses on External Quality

Specification Testing

Tests are based on the specification

Advantages:

- Avoids implementation bias
- Robust to changes in the implementation
- Tests don't require familiarity with the code
- Tests can be developed before the implementation

```
1 """
2 Compute the price of a bus ride:
3     - Children under 2 ride for free.
4     - Children under 18 and senior citizens over 65 pay half the fare
5     - All others pay the full fare of $3.
6     - On weekdays (Monday to Friday), between 7am and 9am and
7       between 4pm and 6pm, a peak surcharge of $1.5 is added
8       to the fare.
9     - During weekends (Saturday and Sunday), there is a flat rate
10      of $2 for all riders, except for children under 2.
11     - Short trips under 5 minutes during off-peak times are free,
12      except on weekends.
13     - If the trip occurs on a public holiday, a special holiday surcharge
14      of $2 is added, ignoring other surcharges and the weekend flat rate.
15 """
16 def bus_ticket_price(age: int,
17                     ride_datetime: datetime,
18                     ride_duration: int,
19                     is_public_holiday: bool) -> float:
20     ...
```

Testing Pyramid

- **Unit Testing:** Testing the smallest testable parts (e.g. functions, objects, modules, or services) in isolation.
- **Integration Testing:** Verifying that different units work together correctly.
- **System/End-to-End (E2E) Testing:** Testing the fully integrated application from the user's perspective.
- **The Pyramid Trade-off:** We want many unit tests (fast, cheap, specific) and fewer E2E tests (slow, expensive, brittle).

Program Under Test: 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

G	U	E	S	S
W	H	I	C	H
C	R	A	Z	E
J	O	I	N	S
T	I	M	E	S
G	A	M	E	S

Program Under Test: Wordle

- **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.

G	U	E	S	S
W	H	I	C	H
C	R	A	Z	E
J	O	I	N	S
T	I	M	E	S
G	A	M	E	S

Activity: Write Wordle Examples

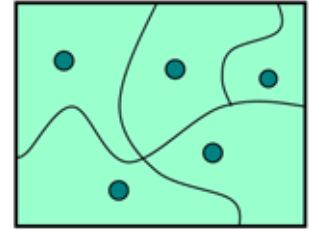
- Your job is to create unit tests for the `mark_guess` function.
- `mark_guess` takes two arguments: `guess_word` and `secret_word`.
- `mark_guess` returns either a tuple 5 “colors” (G for green, Y for Yellow, and B for Black) or Error

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.

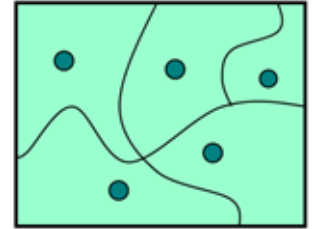
What makes a test good?

Equivalence Partitioning



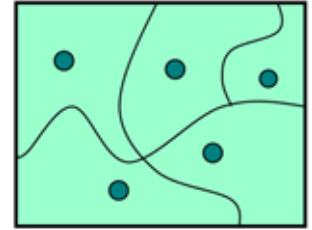
- Identify sets with same behavior (**equivalence class**)
- Try one input from each set
- Equivalence classes derived from specifications (e.g., cases, input ranges, error conditions, fault models)
- Requires domain-knowledge

Example: Equivalence Classes?



```
1 """
2 Compute the price of a bus ride:
3   - Children under 2 ride for free.
4   - Children under 18 and senior citizens over 65 pay half the fare
5   - All others pay the full fare of $3.
6   - On weekdays (Monday to Friday), between 7am and 9am and
7     between 4pm and 6pm, a peak surcharge of $1.5 is added
8     to the fare.
9   - During weekends (Saturday and Sunday), there is a flat rate
10     of $2 for all riders, except for children under 2.
11   - Short trips under 5 minutes during off-peak times are free,
12     except on weekends.
13   - If the trip occurs on a public holiday, a special holiday surcharge
14     of $2 is added, ignoring other surcharges and the weekend flat rate.
15 """
16 def bus_ticket_price(age: int,
17                     ride_datetime: datetime,
18                     ride_duration: int,
19                     is_public_holiday: bool) -> float:
20     ...
```

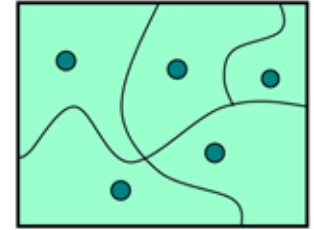
Boundary-value analysis



Key Insight: Errors often occur at the boundaries of a variable value

- For each variable, select:
 - minimum,
 - $\text{min}+1$,
 - medium,
 - $\text{max}-1$,
 - maximum;
 - possibly also invalid values $\text{min}-1$, $\text{max}+1$

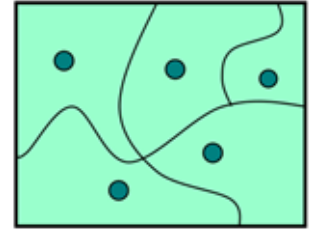
Boundary-value analysis



```
1 """
2 Compute the price of a bus ride:
3 - Children under 2 ride for free.
4 - Children under 18 and senior citizens over 65 pay half the fare
5 - All others pay the full fare of $3.
6 - On weekdays (Monday to Friday), between 7am and 9am and
7   between 4pm and 6pm, a peak surcharge of $1.5 is added
8   to the fare.
9 - During weekends (Saturday and Sunday), there is a flat rate
10  of $2 for all riders, except for children under 2.
11 - Short trips under 5 minutes during off-peak times are free,
12   except on weekends.
13 - If the trip occurs on a public holiday, a special holiday surcharge
14   of $2 is added, ignoring other surcharges and the weekend flat rate.
15 """
16 def bus_ticket_price(age: int,
17                     ride_datetime: datetime,
18                     ride_duration: int,
19                     is_public_holiday: bool) -> float:
20     ...
```

Variable	Domains
age	<2, [2,17], [18,65], >65
ride_datetime	weekdays peak and off-peak, weekends peak and off-peak ...
ride_duration	<5, >=5
is_public_holiday	F, T

Pairwise testing



Key Insight: some problems only occur as the result of an interaction between parameters/components

- Examples of interactions:
 - The bug occurs for senior citizens traveling on weekends (pairwise interaction)
 - The bug occurs for senior citizens traveling on weekends during peak hours (3-way interaction)
 - The bug occurs for adults traveling long trips during public holidays that are weekends. (4-way interaction)
- **Claim: Considering pairwise interactions finds about 50% to 90% of defects**

Test Driven Development

Tests first!

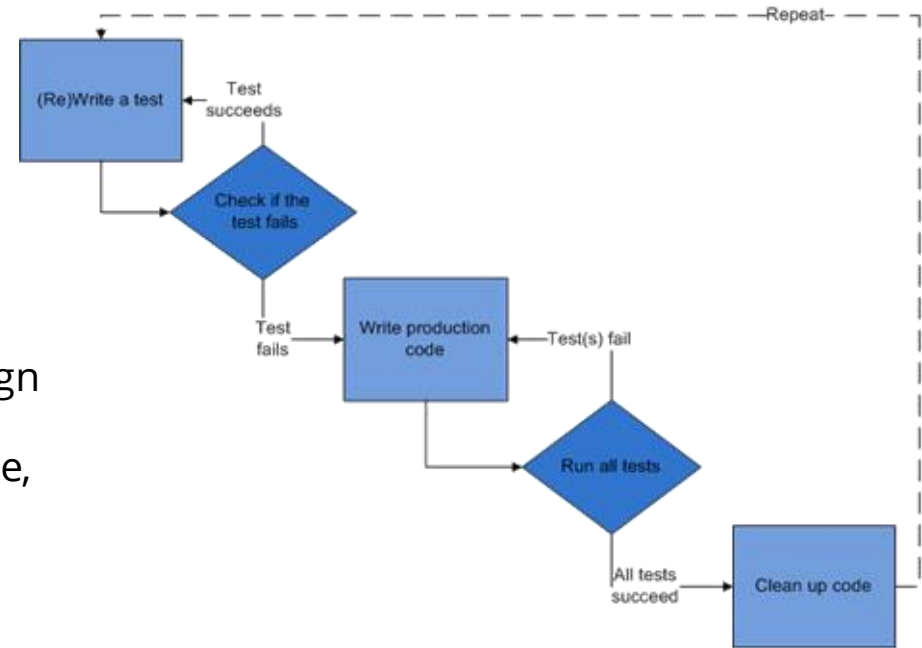
Popular agile technique

Write tests as specifications before code

Never write code without a failing test

Claims:

- Design approach toward testable design
- Avoid writing unneeded code
- Higher product quality (e.g. better code, less defects)
- Higher test suite quality
- Higher overall productivity



**Discussion: How do I
design my program so it is
testable?**

Design testing principles

- Purity
- Determinism
- Small input and output sets
- Well-definite input and output sets

Principles of Testing

Principles of Testing #1:

Avoid the *absence of defects* fallacy

- Testing shows the presence of defects
- Testing does not show the absence of defects!
- “no test team can achieve 100% defect detection effectiveness”



Effective Software Testing: A developer's guide. Maurizio Aniche

What about exhaustive testing?

Idea: Try all values!

- **age: int** (2 - 117) years
- **datetime: DateTime** (hh:mm + M/D/Y)
- **rideTime: int** (in minutes, 1 - 2 Hours)
- **is_public_holiday: bool** (2 values)

116 x 1440 (minutes per day) x 1826 (days in the next 5 years)
x 120 (ride time) x 2 ~ **72 Billion test cases**

What about exhaustive testing?

Exhaustive testing is usually impractical – even for trivially small problem

Key problem: choosing test suite

- **Small enough** to finish in a useful amount of time
- **Large enough** to provide a useful amount of validation

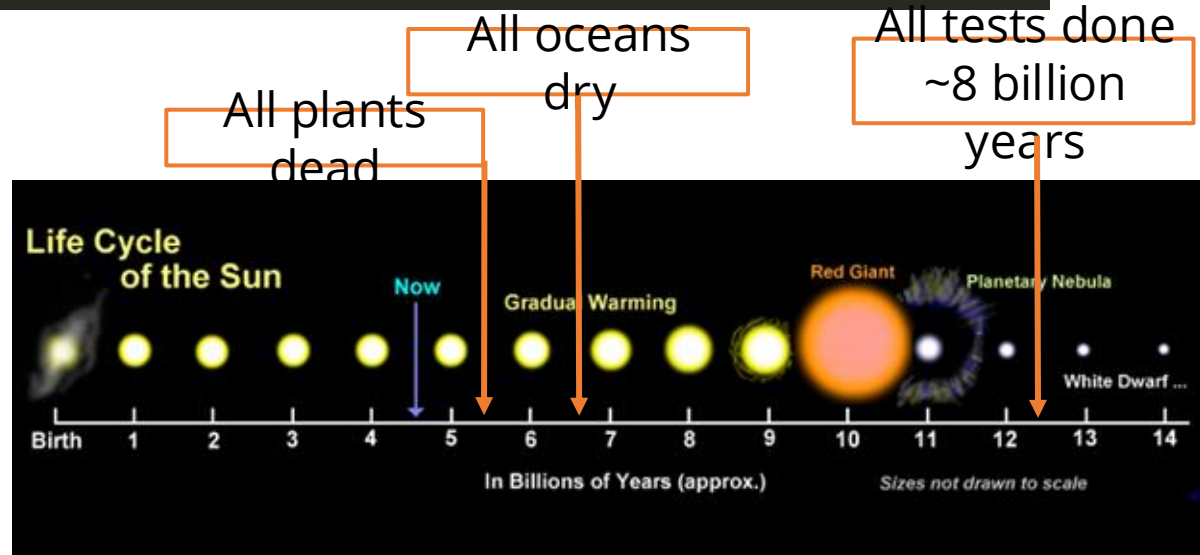
Alternative: **Heuristics**

Principles of Testing #2:

Exhaustive testing is impossible

```
1 def is_valid_email(email: str) -> bool:
2     ...
```

- A simple function, 1 input, string, max. 26 lowercase characters + symbols (@, ., _, -)
- Assume we can use 1 zettaFLOPS: 10^{21} tests per second



Effective Software Testing: A developer's guide. Maurizio Aniche

Principles of Testing #3:

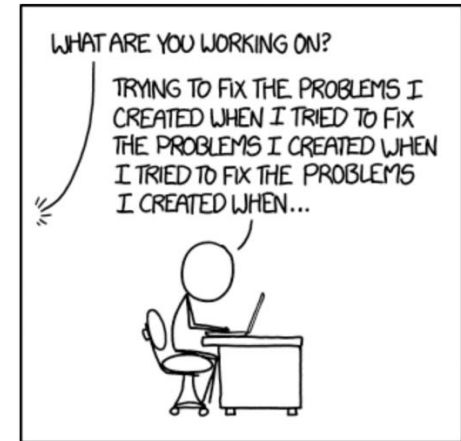
Start testing early

- To let tests guide design
- To get feedback as early as possible
- To find bugs when they are cheapest to fix
- To find bugs when have caused least damage

Principles of Testing #4:

Defects are usually clustered

- “Hot” components requiring frequent change, bad habits, poor developers, tricky logic, business uncertainty, innovative, size, ...
- Use as heuristic to focus test effort

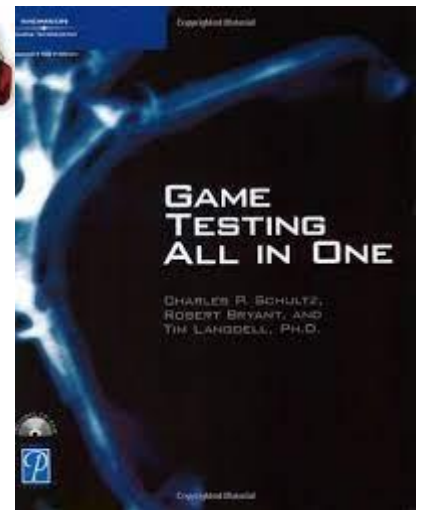
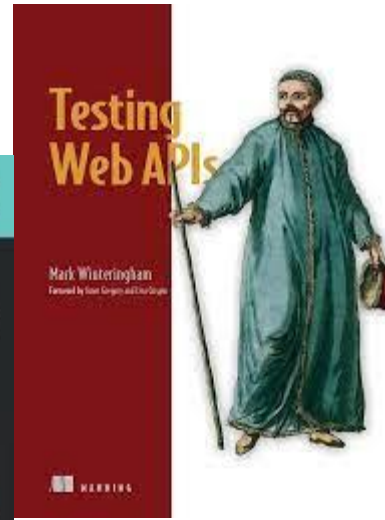
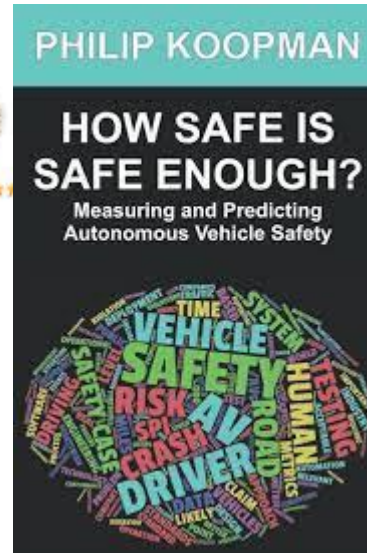
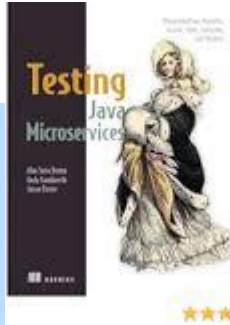


Principles of Testing #5: The pesticide paradox

“Every method you use to prevent or find bugs leaves a residue of subtler bugs against which those methods are ineffectual.”

- Re-running the same test suite again and again on a changing program gives a false sense of security
- Variation in testing

Principles of Testing #6: Testing is context-dependent



Effective Software Testing: A developer's guide. Maurizio Aniche

Principles of Testing #7: Verification is not validation

Verification

- Does the software system meet the requirements specifications?
- Are we building the **software right**?

Validation

- Does the software system meet the user's real needs?
- Are we building the **right software**?



Credit: Philip Koopman

Effective Software Testing: A developer's guide. Maurizio Aniche

Manual testing isn't the only way to assess quality

- Later in this course:
 - Dynamic Analysis
 - Static Analysis
 - Property-based Testing
 - Fuzzing