Software (design) for Data Scientists

ISEA Session 3

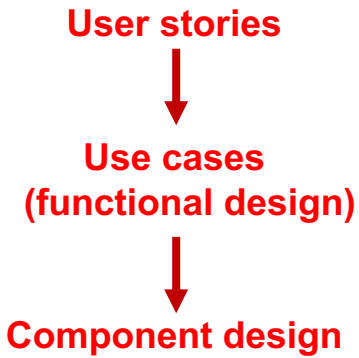
David Beck
University of Washington
2.9.2024

Questions

- For 2025, is there a better example than an ATM?
- How to give feedback on the homework?
- Yours?

Overview of last week and today

1. Review of last week
    1. Users and their stories inform design
    2. Use cases describe the function of software
2. Components implement the use cases
3. Testing and testing strategies
4. Debugging
5. Continuous integration



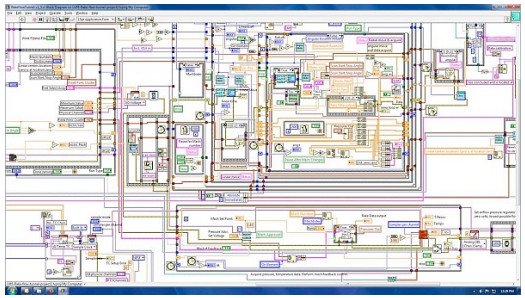
Design fails

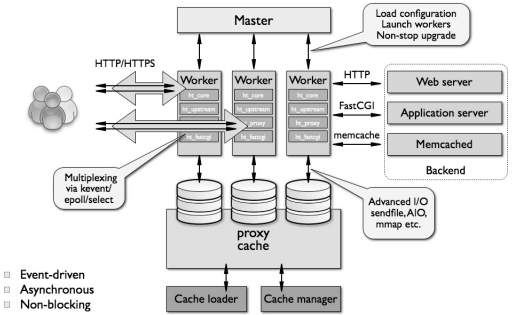
AECL
Atomic Energy of Canada Limited (AECL)

- Therac-25
  - Built in 1982, 6 accidents from 1985-1987
  - Design did not specify what **limits** were set in hardware vs. software (why not both?!?!)
  - Patients were hit with 100x intended radiation dose (50% fatal)
  - Error handling was a mess
    - "Oh Error-54 occurred again? I'll just clear it."
  - AECL had never tested the Therac-25 with the combination of software and hardware until it was assembled at the hospital.

https://en.wikipedia.org/wiki/Therac-25

What makes a design understandable?





- Few components with clear roles
- Few interactions between components
- Carefully choose the features and interfaces
- Similarity with other designs
- Uses design patterns (user interfaces, parallel computing, message observers, …)

Simple where possible is better

Running Example: Design of ATM

Start by writing a user story

Who is the user. What do they want to do with the tool.
What needs and desires do they want for the tool.
What is their skill level.

Start by writing a user story

Ram is a bank customer. Ram wants to check his balance, deposit money. He rarely uses cash. Ram wants a safe and secure interface for interacting with the ATM. Ram's job does not involve technical skills and he values a simple user interface.

Start by writing a user story

> How are Ram and Asma the same?

> How are Ram and Asma different?

> What are the key takeaways from their user stories?

Ram is a bank customer. Ram wants to check his balance, deposit money. He rarely uses cash. Ram wants a safe and secure interface for interacting with the ATM. Ram's job does not involve technical skills and he values a simple user interface.

Asma is a bank customer. Asma wants to check her balance and take out cash. She uses auto-deposit for her paychecks. She wants a safe and secure interface for interacting with the ATM. Asma is quite technical, but she wants to minimize her time interacting with the ATM and values a simple interface

Use Cases

Functional Design

IES Institute of Education Sciences
AmplifyLearn.AI
UNIVERSITY of WASHINGTON eScience Institute
ADVANCING DATA-INTENSIVE DISCOVERY IN ALL FIELDS
UNIVERSITY OF OREGON

How to find use cases? In the user stories!

Ram is a bank customer. Ram wants to check his balance, deposit money. He rarely uses cash. Ram wants a safe and secure interface for interacting with the ATM. Ram's job does not involve technical skills and he values a simple user interface.

- Check balances
- Deposit checks

What do we do with ATMs?

- Check balances
- Deposit checks

} Ram and Asma

- Get cash

} Asma

- **These are examples of *Use Cases*.**
- **They describe the functional potential of software.**

Describing a Use Case (one way)

- What are the inputs and what are the outputs?

- **Adding two numbers**
  - **Inputs: two numbers**
  - **Outputs: one number, the sum of the two inputs**

  - Can we add more detail? What kind of numbers? Positive **and** negative?

Describing a Use Case (Check Balance)

- What are the inputs and what are the outputs?

- **Check balance**
  - **Input:** User selects an account
  - **Output:** ATM displays the current account balance
  - The account information is looked up in the account database and the current balance is retrieved.

Describing a Use Case (Authentication)

- What information the user provides (inputs)
- What responses the system provides (outputs)

Authenticate User Use Case

User: Put ATM card in reader
ATM: Display 'Enter PIN'
User: Enters PIN on keyboard
ATM: [if correct] Show main menu
[if incorrect] Display 'Enter PIN'

Component Design

IES Institute of Education Sciences
AmplifyLearn.AI
UNIVERSITY of WASHINGTON eScience Institute
ADVANCING DATA-INTENSIVE DISCOVERY IN ALL FIELDS
UNIVERSITY OF OREGON

What is a component?

- Software (or other kinds) components “do the work”
- Components store data
- Components calculate values
- Components “interact” with each other
- Components “interact” with the user
- Components can be functions, databases, interfaces, external web sites, ..

Developing component specifications

1. Use case by use case: what are the components required for this use case?
2. Are those components used for another use case?
   - **Good, we can reuse them!**
3. Can the component be further divided in complexity for sub-components?
   - **Good, we can simplify them!**

Specification of a component

- Describe components with sufficient detail so that someone with modest knowledge of the project can implement the code for the component.
  - Name
  - What it does
  - Inputs (with type information)
  - Outputs (with type information)
  - How it uses other components
  - Side effects

Identify shared components

> **Authenticate user**
  - Database with user info including ATM card # and PIN
  - Card reader that reads ATM card
  - User interface that shows information (80x24)
  - User interface that reads user PIN
  - Authenticate control logic

> **Check balance**
  - Database with user info including account balances
  - User interface that reads account selection
  - User interface that shows information (80x24)
  - Check balance control logic

> **User interface (output)?**

> **User interface (input)?**

> **Database?**

> **Control logic?**

Identify shared components

- **Authenticate user**
  - Database with user info including ATM card # and PIN
  - Card reader that reads ATM card
  - User interface that shows information (80x24)
  - User interface that reads user PIN
  - Authenticate control logic

- **Check balance**
  - Database with user info including account balances
  - User interface that reads account selection
  - User interface that shows information (80x24)
  - Check balance control logic

- **User interface (output)? Yes!**
- **User interface (input)? Yes!**
- **Database? After subcomponents.**
- **Control logic? No!**

Overview of today

1. Review of last week
    1. Users and their stories inform design
    2. Use cases describe the function of software
2. Components implement the use cases
3. Testing and testing strategies
4. Debugging
5. Continuous integration

Identify components for a Use Case

- Some use cases appear to only have one component
  - Use case “add numbers”

    Adding two numbers
    - **Inputs: two numbers**
    - **Outputs: one number, the sum of the two inputs**

  - What components are necessary?
    - “Add numbers function”
    - Others?

- Name
- What it does
- Inputs (with type information)
- Outputs (with type information)
- How it uses other components
- Side effects

Specify a component

Name: **Add numbers function**

What it does:

- Computes the sum of two numbers. The numbers can be integers or numbers with decimals. They may be positive or negative.

Inputs (with type information)

- *a,* a number which can be an integer, decimal, positive & negative, must be finite and real
- *b,* a number which can be an integer, decimal, positive & negative, must be finite and real

Outputs (with type information)

- *sum*, a number (integer, decimal, positive, negative, finite, real) that is the sum of *a* and *b*

Components used: None.

Side effects: None.

ATM components by Use Case

- **Authenticate user**
  - Database with user info including ATM card # and PIN
  - Card reader that reads ATM card
  - User interface that shows information (80x24)
  - User interface that reads user PIN
  - Authenticate control logic
    - What is this?

1

1. https://www.deviantart.com/love-rainbowflower/art/Magic-Happens-Here-lineart-292524154

- Name
- What it does
- Inputs (with type information)
- Outputs (with type information)
- How it uses other components
- Side effects

Specify components

> Authenticate user control logic

> Take 2-3 minutes to sketch out a specification for the authenticate user control logic component.

- Name
- What it does
- Inputs (with type information)
- Outputs (with type information)
- How it uses other components
- Side effects

Specify components

Name: **Authenticate user control logic**

What it does:

- Verifies a user is in the database and that the PIN supplied by the user matches the PIN in the database

Inputs (with type information)

- *Card number,* a **string** that is the user's card number
- *PIN*, an integer

Outputs (with type information)

- Boolean: True if success, False if failure

Components used: ATM card reader supplies *Card number* input, user inputs *PIN* via keypad, verification is performed by database

Side effects: If successful, all other bank customer use cases are enabled for the User matching the *Card number.*

- Name
- What it does
- Inputs (with type information)
- Outputs (with type information)
- How it uses other components
- Side effects

Specify components

> Check balance control logic

> Take 2-3 minutes to sketch out a specification for the check balance control logic component.

- Name
- What it does
- Inputs (with type information)
- Outputs (with type information)
- How it uses other components
- Side effects

Specify components

Name: **Check balance control logic**

What it does:

- Looks up a user's account that they provided in the account database and returns the current balance associated with that account in the database.

Inputs (with type information)

- *Account number,* a **string** that is the user's account number

Outputs (with type information)

- False if account does not exist or a floating-point number equal to the account balance

Components used: User selects an *Account Number* shown on the display with keypad input and the verification and balance lookup is provided by the database

Side effects: None.

Digression: Card number as a string?

- Bank card number, e.g.
  - 5534 1234 1234 1234

- Should / could this be a number?
- Should / could this be a string?
  - Test for equality
  - Use the ordinality?
  - Test for number of digits

```
>>> a = "5534123412341234"
>>> b = 5534123412341234
>>> len(a)
16
>>> len(b)
[…]
TypeError: object of type 'int' has
no len()
```

- Name
- What it does
- Inputs (with type information)
- Outputs (with type information)
- How it uses other components
- Side effects

Specify components

Name: **Check balance control logic**

What it does:

- Looks up a user's account that they provided in the account database and returns the current balance associated with that account in the database.

Inputs (with type information)

- *Account number,* a **string** that is the user's account number

Outputs (with type information)

- False if account does not exist or a floating-point number equal to the account balance

Components used: User selects an *Account Number* shown on the display with keypad input and the verification and balance lookup is provided by the database

Side effects: Erases the current content on the user facing display.

Steps in Design[1]

Iterate. Iterate. Iterate.

1. Identify the users and their needs
2. Functional design
   - Describe what the system does (use cases)
3. Component design
   - Components are the “software artifacts” that implement the specific features of the use cases
   - Components are often hierarchical and reused

1. There are many paradigms of software design. This is one. It is focused on humans.

Overview of today

1. Review of last week
    1. Users and their stories inform design
    2. Use cases describe the function of software
2. Components implement the use cases
3. Testing and testing strategies
4. Debugging
5. Continuous integration

Component specifications get you to code

> Code needs review

Code Review Template

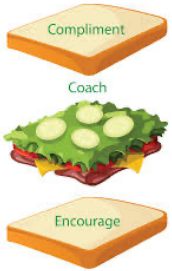
- **Dispassionate third party**
  - Tour of the code with necessary context
  - Improving code quality and find bugs with questions
    - "Why did you call this variable **Snuffleupagus** when it stores the average grade?"
    - "This code is repeated, can it be a function?"
    - "What is happening HERE?!!?!"

- Background
  - Describe what the application does
  - Describe the role of the code being reviewed
- Comment or question
  - Choice of variable and function names
  - Readability of the code
  - How improve reuse and efficiency
  - How use external software packages

Code Review Template

- **Dispassionate third party**
  - Tour of the code with necessary context
  - Improving code quality and find bugs with questions

Assume good will



- Background
  - Describe what the application does
  - Describe the role of the code being reviewed
- Comment or question
  - Choice of variable and function names
  - Readability of the code
  - How improve reuse and efficiency
  - How use external software packages

IES Institute of Education Sciences

AmplifyLearn.AI

UNIVERSITY of WASHINGTON eScience Institute
ADVANCING DATA-INTENSIVE DISCOVERY IN ALL FIELDS

UNIVERSITY OF OREGON

https://www.teamwork.com/blog/disagreeing-sandwich-technique/

Component specifications get you to code

- Code needs review
- Code needs testing

- **Has this happened to you:**
  - You wrote some code. It works! You are happy.
  - You add a neat little feature. You think it works. You are happy.
  - It doesn't work, You found out before anything bad happened.

What is software testing?

- Code that checks if other code (code under test) is working properly
- If the “code under test”
  - Runs successfully
  - Fails gracefully (as expected, when expected)
- The tests pass and the code is “accepted” as working

What is software testing?

```
def code_under_test(a, b):
    return a + b
```

What is the code under test?
What is the code under test doing?
Could we name the code under test function better?

What is software testing?

```
def add(a, b):
    return a + b
```

```
if add(0, 0) == 0:
    return True
else:
    return False
```

What is a test we can do for add?

Is this sufficient? Why or why not?

What is software testing?

```
def add(a, b):
    return a + b
```

```
if add(0, 1) == 1:
    return True
else:
    return False
```

Can we test a different a, b pair?

Is this sufficient?

```
if add(0, 2) == 2:
    return True
else:
    return False
```

What is software testing?

```
def add(a, b):
    return a + b
```

```
for i in range(10):
    if add(0, i) != i:
        return False
```

Can we test many a, b pairs?

Is this sufficient?

Testing is hard! Testing is FUN!

Some “types” of testing

- “Smoke” testing
  - Does it “catch fire and burn” when you try to run the “code under test”
  - Most basic, limited information about system

Code under test

```
def add(a, b):
    return a + b
```

Test code

```
add(0, 0)
```

Some “types” of testing

- “One-shot” testing
  - Does the code under test perform correctly for a specific set of inputs
  - Tests the correctness of code

Code under test

```
def add(a, b):
    return a + b
```

Test code

```
if add(0, 0) == 0:
    return True
else:
    return False
```

Some “types” of testing

- “Pattern” testing
  - Does the code under test perform correctly for a pattern of input cases
  - Tests the correctness of code across a range of inputs

Code under test

```
def add(a, b):
    return a + b
```

Test code

```
for i in range(10):
    if add(0, i) != i:
        return False
return True
```

Some “types” of testing

- “Edge” testing
  - Does the code under test perform correctly for invalid and “special” inputs
  - Tests the error handling and “singular” value handling

Code under test

```
def add(a, b):
    return a + b
```

Test code

```
if add(“4”, 0) != 0:
    return True
else:
    return False
```

Some “types” of testing

- “Edge” testing
  - Does the code under test perform correctly for invalid and “special” inputs
  - Tests the error handling and “singular” value handling

Code under test

```
def add(a, b):
    return a + b
```

Test code

```
assertRaises(TypeError,
    add(“4”, 0))
```

Some “types” of testing

- **“Smoke” test**
  - Calling the code under test to see if it “catches fire”
- **“One-shot” test**
  - Calling the code under test with known inputs expecting specific outputs
- **“Pattern” test**
  - Calling the code under test with a pattern of known inputs expecting a pattern of outputs
- **“Edge” test**
  - **Calling the code under test in “edge cases” and in predefined failure modes to make sure it fails gracefully. FUN!**
- These names are for our convenience.

How many tests for each component?

- **There is no magic number**
- **Patterns** > *One-shot* > Smoke
  - Not all tests offer the same “value” in testing
  - Patterns reveal more possible failure modes

- Number of tests is less important than **code coverage**

Test coverage and code coverage is an important concept in software engineering and can make and break job interviews.

What is code coverage?

- **Fraction (%) of lines of code "exercised" by a test**
  - How many lines in **`add`**?
  - What percentage of those lines are executed by our test?

    100% (1 of 1 lines of add were executed)

Code under test

```
def add(a, b):
1.      return a + b
```

Test code

```
add(0, 0)
```

What is code coverage?

- **Fraction (%) of lines of code "exercised" by a test**
  - What percentage of those lines are executed by our test?

100% (1 of 1 lines of add were executed)

Code under test

```
def add(a, b):
1.      return a + b
```

Test code

```
if add(0, 0) == 0:
    return True
else:
    return False
```

What is code coverage?

- **Fraction (%) of lines of code "exercised" by a test**
  - What percentage of those lines are executed by our test?

100% (1 of 1 lines of add were executed)

Code under test

```
def add(a, b):
1.      return a + b
```

Test code

```
for i in range(10):
    if add(0, i) != i:
        return False
return True
```

What is code coverage?

Code under test

```
   def add_or_multiply(op, a, b):
1.    if op == "+":
2.          return a + b
3.    else:
4.          return a * b
```

Code coverage?

50%

Test code

```
if add_or_multiply("+", 0, 1) != 1:
      return False
```

UNIVERSITY OF OREGON

What is code coverage?

Code under test

```
   def add_or_multiply(op, a, b):
1.    if op == "+":
2.          return a + b
3.    else:
4.          return a * b
```

Code coverage?

50%

Test code

```
if add_or_multiply("*", 0, 1) != 0:
      return False
```

UNIVERSITY OF OREGON

What is code coverage?

Code under test

```
   def add_or_multiply(op, a, b):
1.    if op == "+":
2.          return a + b
3.    else:
4.          return a * b
```

Test code

```
if add_or_multiply("+", 0, 1) != 0:
    return False
if add_or_multiply("*", 0, 1) != 0:
    return False
```

Code coverage?

100%

UNIVERSITY OF OREGON

Is code coverage alone enough?

Code under test

```
   def add_or_multiply(op, a, b):
1.    if op == "+":
2.          return a + b
3.    else:
4.          return a * b
```

Code coverage?

100%

Test code

```
if add_or_multiply("+", 0, 1) != 0:
      return False
if add_or_multiply("/", 0, 1) != 0:
      return False
```

UNIVERSITY OF OREGON

Testing is hard! Testing is fun!

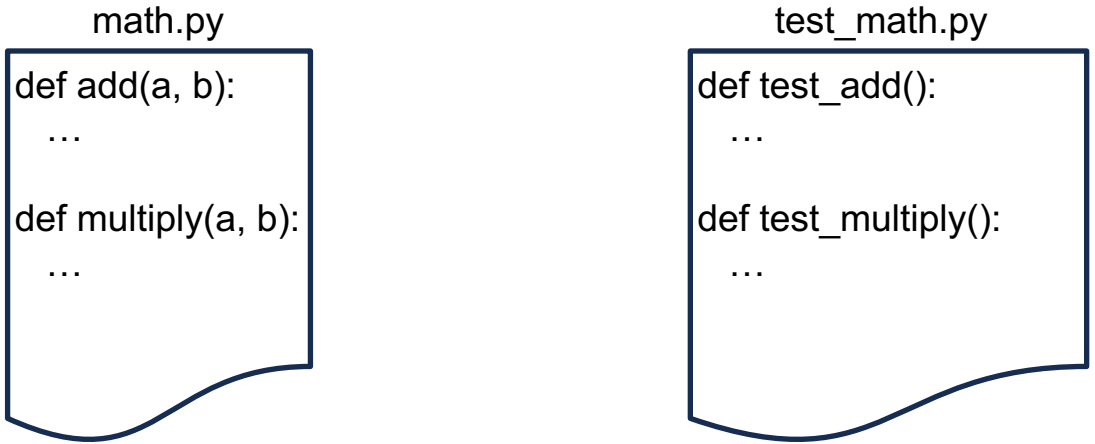
- Mode of thinking: **How can I break this?**
- Want to get better?
  - Practice!
- “Software engineer in test” is a specific job
  - Software development engineering in test (SDET)
  - U.S. Bureau of Labor Statistics predicts > 25% job growth
  - Pathway to a Software Development Engineer

Testing is only as good as your imagination!

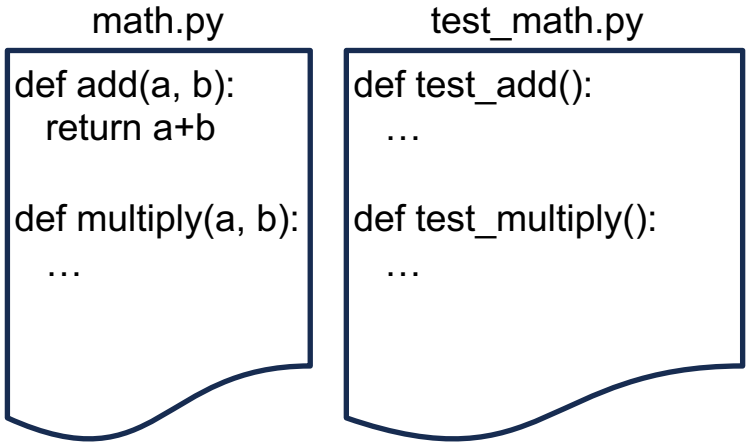
- **“Software engineer in test” is a specific job**
  - Software tester walks into a bar…
  - Software tester run into a bar…
  - Software tester crawls into a bar…
  - Software tester walks into a bar and orders a drink…
  - Software tester runs into a bar and orders a drink…
  - Software tester crawls into a bar and orders a drink…
  - …
  - User walks into a bar and asks for the bathroom.

What does this look like in files?

- **One source file for code, one source file for tests**
  - `math.py` has an accompanying `test_math.py`

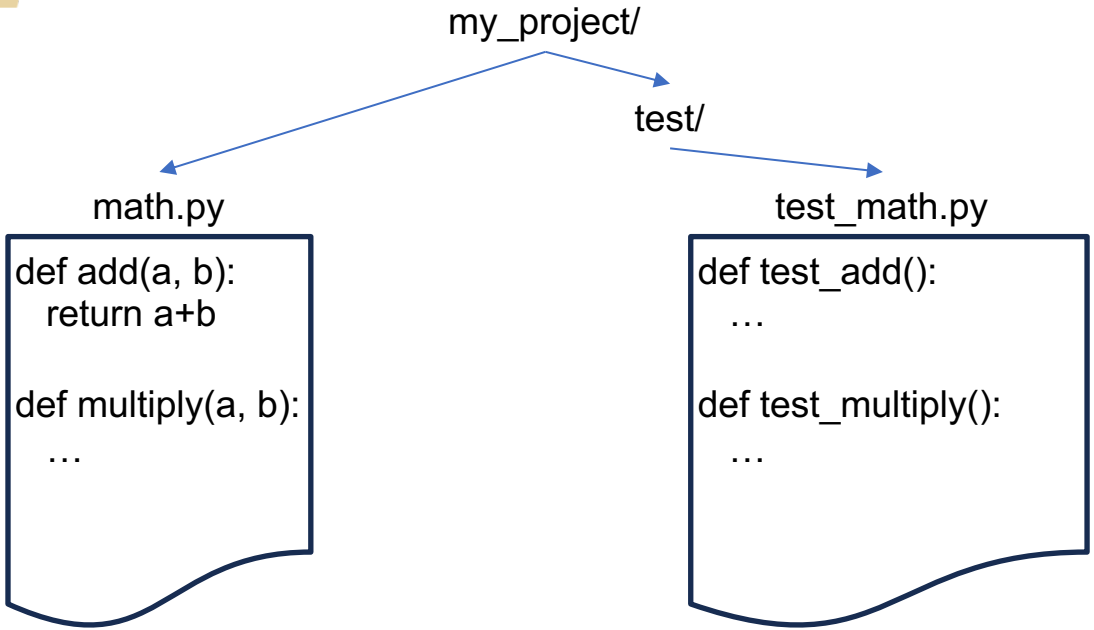


What does this look like in files?

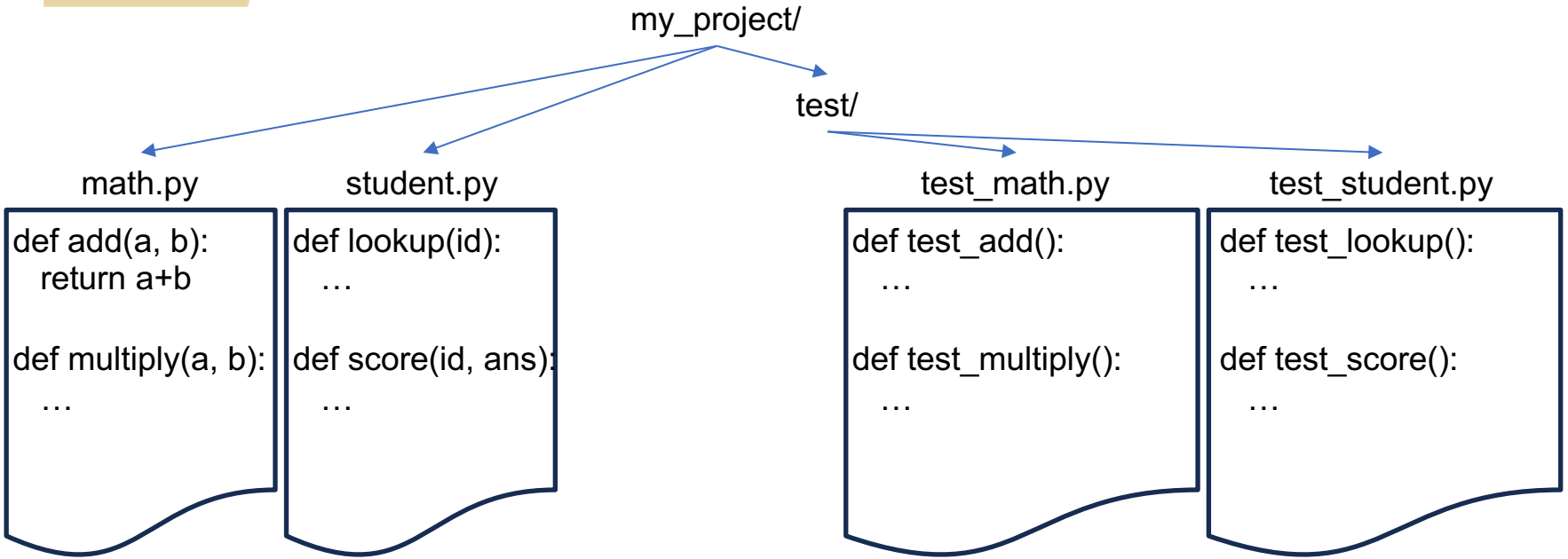



```
def test_add():
    for i in range(10):
        assert add(0, i) == i
```

What does this look like in projects?

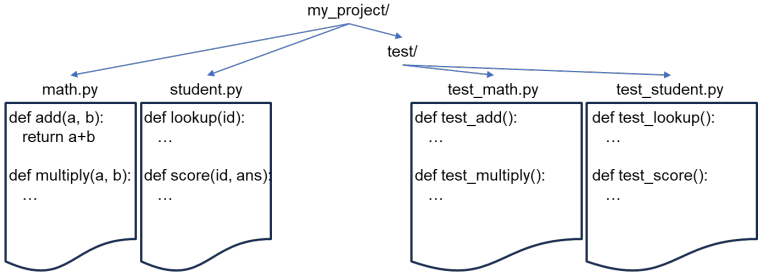


What does this look like in real projects?

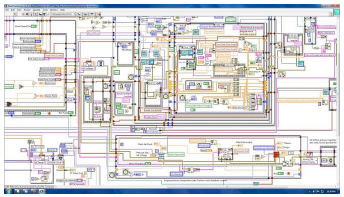


What does this look like in day to day?

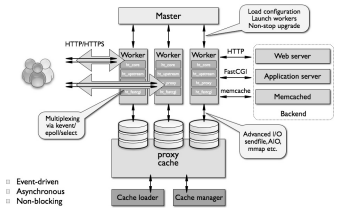
- After each change to a function or file (e.g. `math.py`)
  - Rerun the tests (`test_math.py`)
  - **Do we need to run all tests?**



Yes!

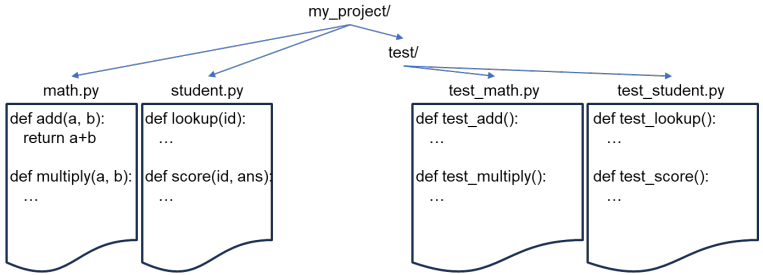


Maybe?



What does this look like in day to day?

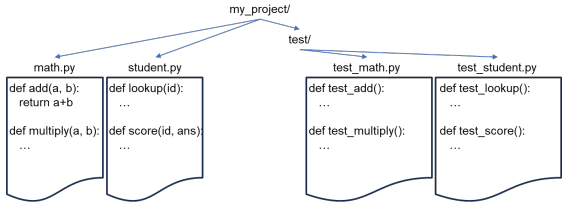
- After each change to a function or file (e.g. `math.py`)
  - Rerun the tests (`test_math.py`)



  - **Do we need to run all tests?**
    - The design should tell us
    - Component specification includes component interactions
    - Component specification includes side effects

Testing is highly automated with tooling

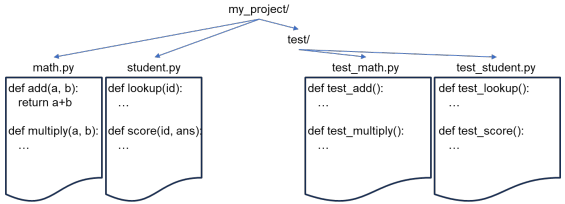
- After each change to a function or file (e.g. `math.py`)
  - Rerun the tests (`test_math.py`)



  - **Do we need to run all tests?**
    - If components in `math.py` call components in student.py, **yes**
    - If components in `student.py` call components in math.py, **yes**

Testing can be a big lift for large software

- After each change to a function or file (e.g. `math.py`)
  - Rerun all the tests (`test_math.py, test_student.py`)
  - **Testing is automated**, so run them all
  - **Big software projects**
    - Testing can be expensive (money & time)
    - Automated tests can be run 100s of times a day



Test-driven development

- Testing is automated
  - `pytest`, `nosetest`, `python -m unittest`
- Tests can be written from component specifications
- Tests can be written without “code under test”

Test-driven development

> Tests can be written without “code under test”

Code under test

Test code

```
def test_add():
    for i in range(10):
        assert add(0, i) == i
```

Test code output

```
>>> test_add()
[…]
NameError: name 'add'
is not defined
```

Test-driven development

> Tests can be written with empty “code under test”

Code under test

```
def add(a, b):
    pass
```

Do nothing, take no action, no operation, return None

Test code

```
def test_add():
    for i in range(10):
        assert add(0, i) == i
```

Test code output

```
>>> test_add()
[…]
AssertionError
```

Test-driven development

> Tests can be written to drive writing “code under test”

Code under test

```
def add(a, b):
      return a + b
```

Test code

```
def test_add():
      for i in range(10):
            assert add(0, i) == i
```

Test code output

```
>>> test_add()
>>>
```

No errors so a success

Test-driven development

- Tests are written against the component specification
- All components are implemented as '`pass`'
- All tests:
  - **FAIL!**
- Write code until all tests pass
- Release software bug free!

```
def add(a, b):
    pass
```

Overview of today

1. Review of last week
    1. Users and their stories inform design
    2. Use cases describe the function of software
2. Components implement the use cases
3. Testing and testing strategies
4. Debugging
5. Continuous integration

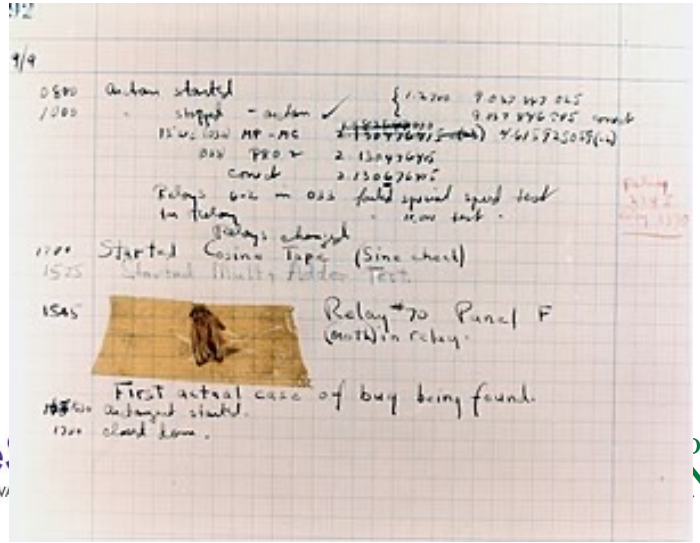
When tests fail: Debugging

- What is a bug?
  - “A software bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or behave in unintended ways.” – Wikipedia
- Where did the term come from?
  - Thomas Edison (1878, letter to associate) - Wikipedia

> ... difficulties arise—this thing gives out and [it is] then that "Bugs"—as such little faults and difficulties are called—show themselves

When tests fail: Debugging

- Where did the term come from?
  - Grace Hopper, USN rear admiral, 1906-1992
  - Two PhDs, developed the UNIVAC I, COBOL



- 1947
- Mark II mechanical computer
- Her team discovered a moth stuck in a relay
- The relay would not function until the moth was removed
- Thus, a computer was “debugged”

Wikipedia

When tests fail: Debugging

- Two types of debugging
  - Print debugging
    - Using print statements to inspect the state of your code
  - Debugger based tools
    - Using tooling to inspect the state of your code at run time

When tests fail: Debugging

- Two types of debugging
  - Print debugging

```
def add_or_multiply(op, a, b):
    if op == "+":
        return a + b
    else:
        return a * b
```

When tests fail: Debugging

```
def add_or_multiply(op, a, b):
  print("add_or_multiply called")
  print(op)
  print(a)
  print(b)
  if op == "+":
       print("we are adding")
       return a + b
  else:
       print("we are multiplying")
       return a * b
```

IVERSITY OF
REGON

When tests fail: Debugging

```
def add_or_multiply(op, a, b):
    print("add_or_multiply called")
    print(op)
    print(a)
    print(b)
    if op == "+":
        print("we are adding")
        return a + b
    else:
        print("we are multiplying")
        return a * b
```

```
>>> add_or_multiply("+", 0, 0)
add_or_multiply called
+
0
0
we are adding
```

IVERSITY OF
REGON

When tests fail: Debugging

- Two types of debugging
  - Print debugging
    - WORKS!!!!!
    - Time consuming

```
def add_or_multiply(op, a, b):
    if op == "+":
        return a + b
    else:
        return a * b
```

```
def add_or_multiply(op, a, b):
    print("add_or_multiply called")
    print(op)
    print(a)
    print(b)
    if op == "+":
        print("we are adding")
        return a + b
    else:
        print("we are multiplying")
        return a * b
```

When tests fail: Debugging

- Two types of debugging
  - Debugger based tools
    - Using tooling to inspect the state of your code at run time

```
 def add_or_multiply(op, a, b):
1.    if op == “+”:
2.         return a + b
3.    else:
4.         return a * b
```

When tests fail: Debugging

- Two types of debugging
  - Debugger based tools
    - Using tooling to inspect the state of your code at run time

```
add_or_multiply("/", 0, 0)
```

```
def add_or_multiply(op, a, b):
1.    if op == "+":
2.        return a + b
3.    else:
4.        return a * b
```

When tests fail: Debugging

What line is executed next?

- Two types of debugging
  - Debugger based tools
    - Using tooling to inspect the state of your code at run time

```
add_or_multiply("/", 0, 0)
```

```
 def add_or_multiply(op, a, b):
1.    if op == "+":
2.         return a + b
3.    else:
4.         return a * b
```

Variables

```
op = "/"
a = 0
b = 0
```

When tests fail: Debugging

What line is executed next?

- Two types of debugging
  - Debugger based tools
    - Using tooling to inspect the state of your code at run time

```
add_or_multiply("/", 0, 0)
```

```
 def add_or_multiply(op, a, b):
1.    if op == "+":
2.         return a + b
3.    else:
4.         return a * b
```

Variables

```
op = "/"
a = 0
b = 0
```

When tests fail: Debugging

What line is executed next?

- Two types of debugging
  - Debugger based tools
    - Using tooling to inspect the state of your code at run time

```
add_or_multiply("/", 0, 0)
```

```
 def add_or_multiply(op, a, b):
1.    if op == "+":
2.         return a + b
3.    else:
4.         return a * b
```

Variables

```
op = "/"
a = 0
b = 0
```

When tests fail: Debugging

- Two types of debugging
  - Print debugging
    - Using print statements to inspect the state of your code
    - Easy at first, slow and difficult later
  - Debugger based tools
    - Using tooling to inspect the state of your code at run time
    - Hard at first, easy and fast later
- **Career progression in SDE will require Debugger skill**

Overview of today

1. Review of last week
    1. Users and their stories inform design
    2. Use cases describe the function of software
2. Components implement the use cases
3. Testing and testing strategies
4. Debugging
5. Continuous integration

What is continuous software testing?

```
add(a, b)
multiply(a, b)
```

} Version 1 of our simple math library

In version 2, we want to support complex numbers. How can we be sure that our changes don't break things?

"Continuous integration" or continuously integrating new code into your software **after** testing.

The tests pass and the code is "accepted" as working

Homework for next week

- Thinking about your assignment and homework for Session 2, can you
  - Identify a software component for the design you proposed last week
    - Can you subdivide the component?
    - Why or why not?
  - Describe, using the standard types of tests introduced, the testing challenges and strategies you might opt to employ for that component
    - Are there pattern tests available? Are you sure? Really?
    - What edge tests are appropriate?