

COMS W4156 Advanced Software Engineering (ASE)

October 11, 2022

Agenda

1. Docker demo
2. finish Equivalence Partitions and Boundary Analysis
3. Service vs. App



Docker demo: Franck Mugisho

[demo_presentation](#)



Read The Docs

[Docker website](#)

[Docker for beginners](#)



Important note: Teams may use Docker if they want to, e.g., to deal with incompatible platforms, but no one is required to use Docker (or VMs) and Docker is not officially part of this course

Agenda

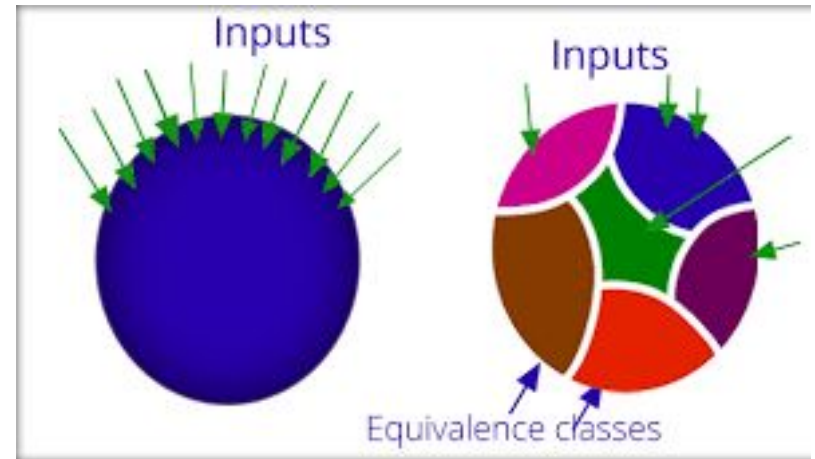
1. Docker demo
2. finish Equivalence Partitions and Boundary Analysis
3. Service vs. App



Equivalence Partitions

The simplest approach for representing all possible inputs to a system with a practically small number of examples is to divide the input space into *equivalence partitions* (also known as equivalence classes) based on **both**

- application/domain knowledge (e.g., required features)
- software engineering knowledge



Equivalence Partitions

Equivalence partition = set of alternative input values where the software can reasonably be expected to behave equivalently (similarly)

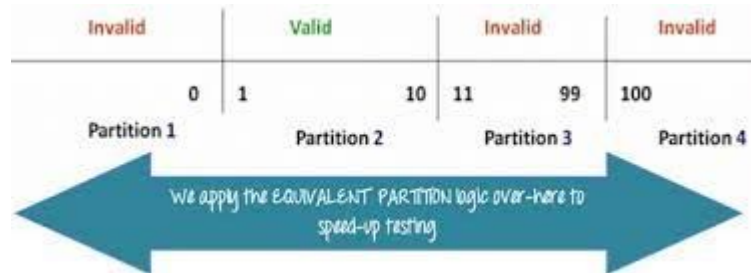
This does **not** mean literally the same output for every input in the class

Instead it means that if the software behaves correctly on one example input chosen from that equivalence class, it is reasonable to believe it will behave correctly on all the other inputs in the same class



Simple Example

If we are testing a “box” intended to accept numbers from 1 to 1000000, then there is no point in writing a million test cases for all 1000000 valid input numbers - and also no point in writing many more test cases for all invalid input numbers between MinInt and MaxInt

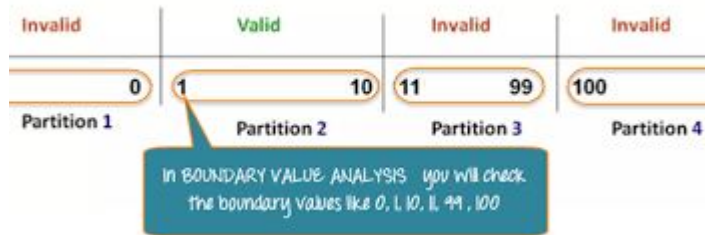


Instead, divide the test cases into three sets:

- 1) Input partition with all valid inputs. Pick a single value from range 1 to 1000000 as a valid test case. If we select other values between 1 and 1000000 the result “should” also be correct (or also be incorrect)
- 2) Pick a single value for the input partition with all values *below* the lower limit of the range, i.e., any value below 1
- 3) Pick a single value for the input partition with all values *above* the upper limit, i.e., any value above 1000000

Boundary Analysis

Equivalence partitions are typically extended with *boundary analysis* because input values at the extreme ends of the valid and invalid partitions are more likely to trigger bugs, e.g., “off by one”, “corner case”



- 1) Test cases with input exactly at the boundaries of the valid partition, i.e., values 1 and 1000000
- 2) Test data with values just below the extreme edges, i.e., values 0 and 999999
- 3) Test data with values just above the extreme edges, i.e., values 2 and 1000001
- 4) For numerical domains, make sure to include zero, positive and negative examples, as well as the underlying primitive extremes (integers). In this case, we already have 0 and some positive values, so we need to add some negative value, e.g., -1, plus MinInt and MaxInt
- 5) For numerical domains, also try some non-numerical input, e.g., “abcde” (if it is possible to provide such inputs)

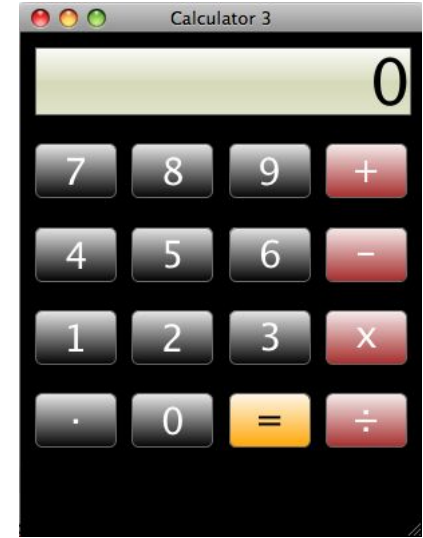
Another Example

Consider a calculator implemented in software, with a GUI as shown. The user clicks the buttons with a mouse, but cannot enter text into the display

In mathematics, the “+” operation takes two numeric operands

Say this software is tested by an external testing team separate from the developers

Based on *domain knowledge*, a tester might reasonably expect the “+” operation to behave similarly on all pairs of numbers
< first operand, second operand >

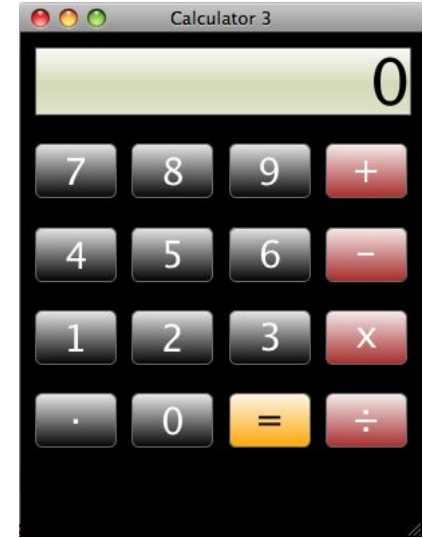


Another Example

So the tester tries an arbitrary pair of numbers:

< 3.141592653, 42 >

The calculator flashes 00000000. What does this mean?



Interpreting Test Outputs

Based on *software engineering knowledge*, the tester might realize the flashing 00000000 could indicate:

- An error message because the software cannot handle numbers with more than N digits (e.g., 8)
- An error message because the software cannot handle mixed operands, i.e., one operand is floating point and the other is integer
- A bug
- Something else (e.g., the user has reached the limit of their “free trial” and now has to pay to continue using the calculator software)



Continuing Calculator Example

Let's ignore the “.” key (planned for version 2.0) and assume the “+” operator is currently intended only for integer operands

A tester with software engineering knowledge, not just domain knowledge, might realize that a calculator implementation could behave differently for positive integers, zero, and negative integers

Thus they test with five different equivalence classes:

- < pos int, pos int >
- < pos int, neg int >
- < pos int, 0 >
- < neg int, neg int >
- < 0, 0 >

Is this sufficient?



Although the “+” operator is mathematically commutative, that does not mean that the implementation (which could be buggy) is commutative

Or, more generally, define equivalence partitions separately per parameter, and then extend to the cross-product

But notice the scaling problem with large numbers of parameters!

- $\langle \text{pos int}, \text{pos int} \rangle$
- $\langle \text{pos int}, \text{neg int} \rangle$
- $\langle \text{pos int}, 0 \rangle$
- $\langle \text{neg int}, \text{pos int} \rangle$
- $\langle \text{neg int}, \text{neg int} \rangle$
- $\langle \text{neg int}, 0 \rangle$
- $\langle 0, \text{pos int} \rangle$
- $\langle 0, \text{neg int} \rangle$
- $\langle 0, 0 \rangle$



Let's say the calculator is intended to handle operands only up to 8 digits (or to any other fixed number of digits)

This means there is a MaxInt and a MinInt (negative), probably specific to what the calculator GUI was designed to display rather than MaxInt/MinInt of the programming language, compiler, operating system, etc.

What additional tests are needed?

- < xxx, MaxInt >
- < MaxInt, xxx >
- < xxx, MinInt >
- < MinInt, xxx >
- < MaxInt, MinInt >
- < MinInt, MaxInt >



Invalid Inputs to Calculator Example



What if the user could enter text, using a conventional keyboard, into the display window in addition to mousing the keys

How does that change the set of equivalence partitions that need to be tested?

Another Example

What are the
equivalence partitions?

How would we test this
program?

Design-your-burger

Please make your selection

Burger:

Cooked: ☐ rare ☐ medium ☐ well

Cheese: ☐

Lettuce: ☐

Tomato: ☐

Onion: ☐

Ketchup: ☐

Mustard: ☐

Mayo: ☐

Secret sauce: ☐



[Nutrition facts](#) [About the chef](#)

Required vs Optional Inputs

Get

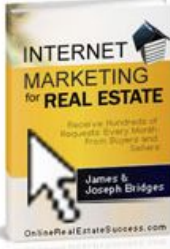
"Internet Marketing for Real Estate EBook"

via e-mail

Name*:

Email*:

Phone*:



* Indicates a required field

* Indicates an optional field

Equivalence classes need to address required vs. optional inputs, with or without default values

What are the equivalence partitions?

How would we test this program?

Dependencies between Inputs

Distinct parameters may be related to each other in forming equivalence classes

Let's say we had a function that took two variables (blood sugar level and level of frustration) to calculate "mood"



Assume valid inputs are
Blood sugar 70 - 100
Frustration is 50% - 100%

For $50 \leq F \leq 90$

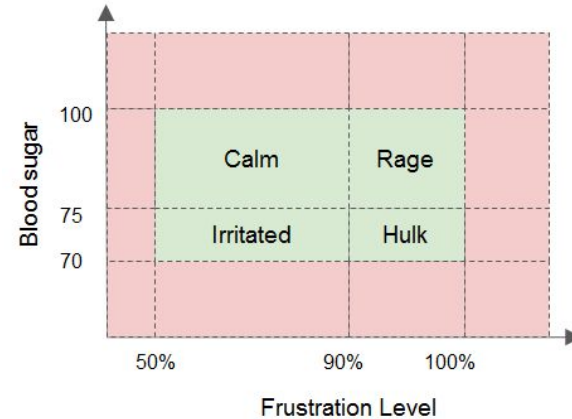
75 \leq BS \leq 100: Calm

BS < 75: Irritated

For $F > 90$

75 \leq BS \leq 100: Rage

BS < 75: [Hulk Smash](#)



More Dependencies between Inputs

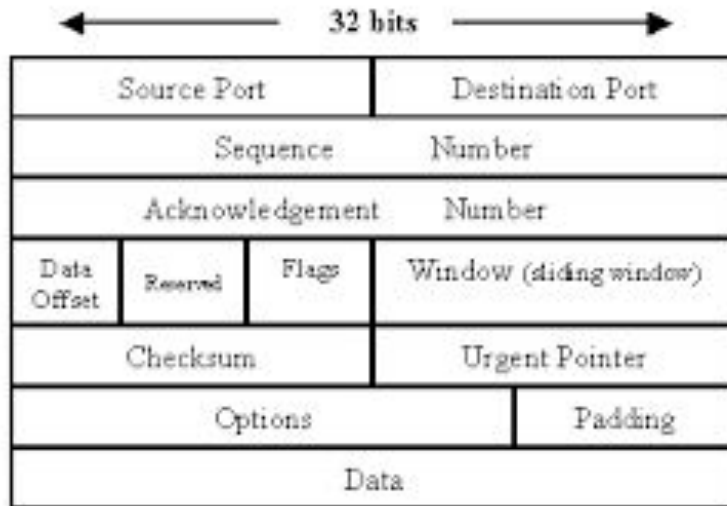
Equivalence classes may cut across multiple inputs:



Sometimes the value of one input restricts appropriate values for other inputs. For example, date of birth and age should be consistent (valid) or not (invalid)

Sometimes the value of one (small) input is intended as a sanity check on another (larger, more complicated) input, such as the checksum in the network packet

System to System Inputs (includes APIs)



JSON Object → {

String Value ↓

Object Inside Object → {

JSON Array → [

Array Inside Array → [

Number Value → 102

Null Value → null

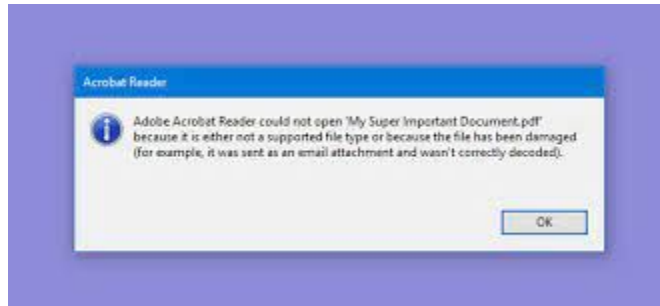
```
{
  "company": "mycompany",
  "companycontacts": {
    "phone": "123-123-1234",
    "email": "myemail@domain.com"
  },
  "employees": [
    {
      "id": 101,
      "name": "John",
      "contacts": [
        "email1@employee1.com",
        "email2@employee1.com"
      ]
    },
    {
      "id": 102,
      "name": "William",
      "contacts": null
    }
  ]
}
```

System and Service Inputs

At the full system and service levels, inputs are generally going to be numbers, strings (text), tuples (packets, json), and files

File inputs can:

- Exist or not
- Be below min or above max size
- Be readable, writeable, executable by current user (permissions)
- Correct, corrupted, garbled, etc. data and metadata formats - images, audio, video, ...



Unit Inputs

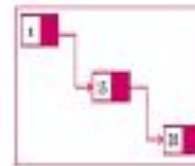
At the unit testing level, inputs/outputs can be arbitrary data structures, so equivalence classes can be more complex - and inputs more complicated to construct

Container data structures with content elements:

- In or not in container
- Empty or full container (or min/max number of elements)
- Duplicate elements may or may not be allowed
- Ordered or organized correctly, or not



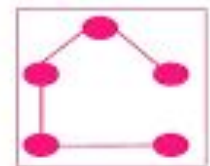
Sorting



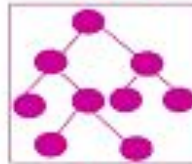
Link list



list



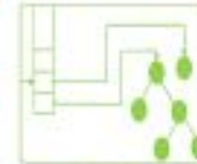
spanning tree



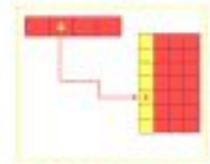
Tree



Graph



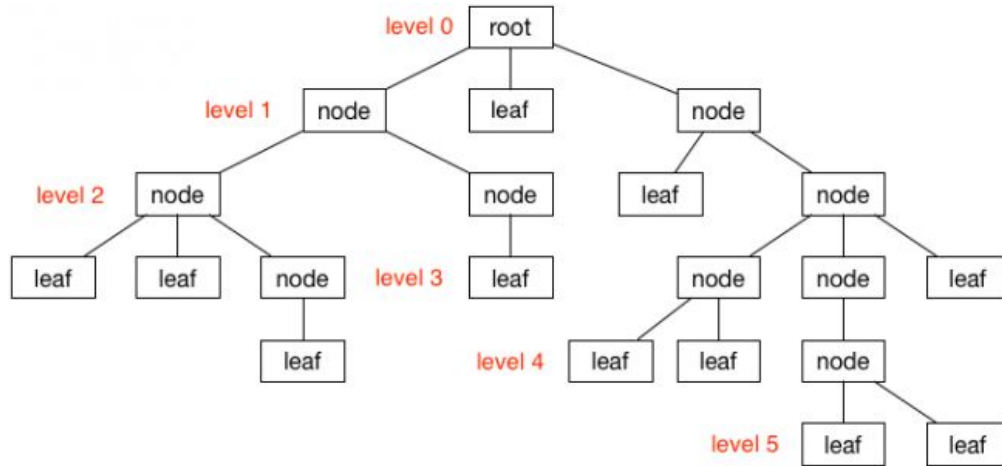
Stack



Hashing

Fig. 1.1.1. Data structures and algorithms

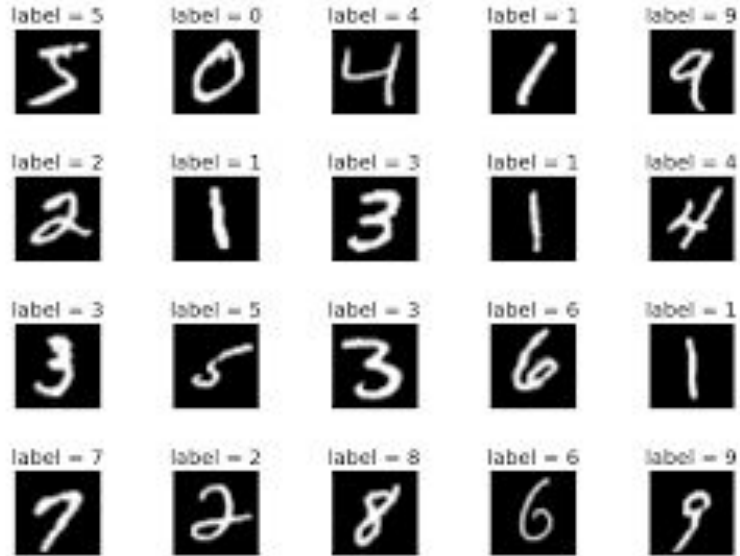
More Unit Inputs



Specific kinds of containers, e.g., tree - root node, interior node, leaf node, null tree, tree with exactly one node, “full” tree, balanced vs. unbalanced, various specialty trees - and graphs

How would test cases construct root, node and leaf objects to use as inputs?

Equivalence Partitions for Data Sets



Files containing “data sets” often correspond to a table (rows and columns)

This shows part of a well-known dataset for deep learning algorithms ([MNIST](#))

What are the equivalence classes?

Agenda

1. Docker demo
2. finish Equivalence Partitions and Boundary Analysis
3. Service vs. App



T2 Revised Proposal



I read all the revised proposals on Friday (these might not be your final versions)

Many of them describe an app (not a service), and discuss what users (not clients) can do with the software

Many of the answers to question 2b describe multiple user roles for the same app, not multiple different kinds of clients for the same service

Some of the T2 submissions do not answer question 2b at all. 2b asks **“Describe at least three different kinds of clients, including what you envision those clients would 'do' for their own clients or end-users.”**

I've provided notes to the TAs with my comments about the proposals, in some cases they may contact you to suggest changes for the first iteration (yes you can change)

Upcoming Assignments

[First Iteration](#) due October 24

[First Iteration Demo](#) due October 31



Next Class

Working in a Team

CheckStyle demo

Pair Programming



Ask Me Anything