SoftServe | Empowering your Business through Software Development

Equivalence Class Testing Technique Training

Yanina Hladkova

Agenda

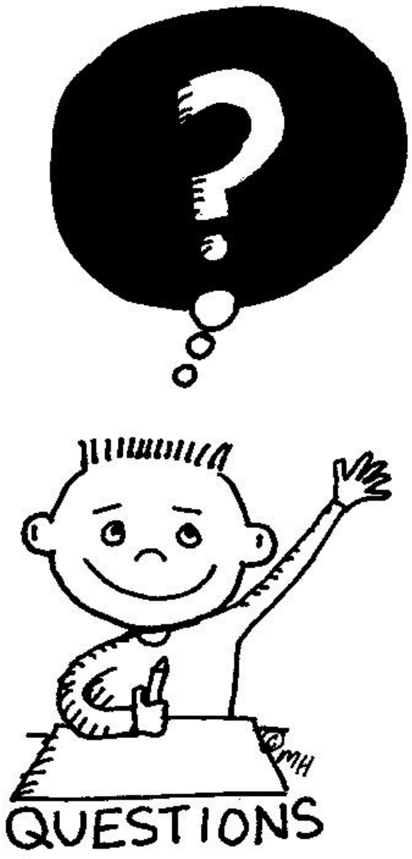
1. Introduction
2. Technique
3. Examples
4. Applicability and Limitations
5. Summary
6. Practice
7. References


The Beginning

Introduction

What is equivalence class testing?
What is it used for?

Equivalence class testing is a technique used to reduce the number of test cases to a manageable level while still maintaining reasonable test coverage.



Introduction: Situation

We are writing a module for a human resources system that decides how we should process employment applications based on a person's age. Our organization's rules are:

0–16 – Don't hire

16–18 – Can hire on a part-time basis only

18–55 – Can hire as a full-time employee

55–99 – Don't hire

Introduction: Coverage

Should we test the module for the following ages: 0, 1, 2, 3, 4, 5, 6, 7, 8, ..., 90, 91, 92, 93, 94, 95, 96, 97, 98, 99?

If we had lots of time (and didn't mind the mind-numbing repetition and were being paid by the hour) we certainly could.

100 values

Introduction: Solution 1

```
If (applicantAge == 0) hireStatus="NO";
If (applicantAge == 1) hireStatus="NO";
…
If (applicantAge == 15) hireStatus="NO";
If (applicantAge == 16) hireStatus="PART";
If (applicantAge == 17) hireStatus="PART";
If (applicantAge == 18) hireStatus="FULL";
If (applicantAge == 19) hireStatus="FULL";
…
If (applicantAge == 53) hireStatus="FULL";
If (applicantAge == 54) hireStatus="FULL";
If (applicantAge == 55) hireStatus="NO";
If (applicantAge == 56) hireStatus="NO";
…
If (applicantAge == 98) hireStatus="NO";
If (applicantAge == 99) hireStatus="NO";
```

Any set of tests passes tells us nothing about the next test we could execute. It may pass; it may fail.

Introduction: Let’s believe

Introduction: Solution 2

```
If (applicantAge >= 0 && applicantAge <=16)
        hireStatus="NO";
If (applicantAge >= 16 && applicantAge <=18)
        hireStatus="PART";
If (applicantAge >= 18 && applicantAge <=55)
        hireStatus="FULL";
If (applicantAge >= 55 && applicantAge <=99)
        hireStatus="NO";
```

It is clear that for the first requirement we don't have to test 0, 1, 2, ... 14, 15, and 16. Only one value needs to be tested. And which value? Any one within that range is just as good as any other one. The same is true for each of the other ranges. Ranges such as the ones described here are called **equivalence classes.**

Introduction: Benefits

Using the equivalence class approach, we have reduced the number of test cases

From 100 (testing each age)

To 4 (testing one age in each equivalence class)

A significant savings

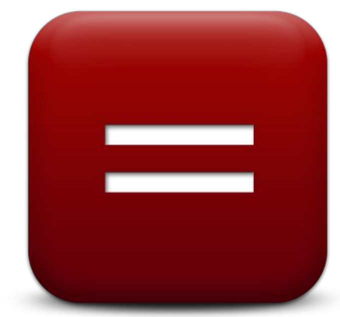
Introduction: Definition

An equivalence class consists of a set of data that is treated the same by the module or that should produce the same result. Any data value within a class is *equivalent,* in terms of testing, to any other value.

Introduction: Assumptions

Specifically, we would expect that:

- If one test case in an equivalence class detects a defect, *all* other test cases in the same equivalence class are likely to detect the same defect.
- If one test case in an equivalence class does not detect a defect, *no* other test cases in the same equivalence class is likely to detect the defect.



Introduction: Solution 3

```
If (applicantAge >= 0 && applicantAge <=16)
hireStatus="NO";
If (applicantAge >= 16 && applicantAge <=18) hireStatus="PART";
If (applicantAge >= 18 && applicantAge <=41) hireStatus="FULL";
// strange statements follow
If (applicantAge == 42 && applicantName == "Lee")
hireStatus="HIRE NOW AT HUGE SALARY";
If (applicantAge == 42 && applicantName <> "Lee")
hireStatus="FULL";
// end of strange statements
If (applicantAge >= 43 && applicantAge <=55) hireStatus="FULL";
If (applicantAge >= 55 && applicantAge <=99) hireStatus="NO";
```

just another example

Introduction: Ready?

Now, are we ready to begin testing?

Probably not.

What about input values like 969, -42, FRED, and &$#!@? Should we create test cases for invalid input?

The answer is, as any good consultant will tell you, "it depends“.

Technique

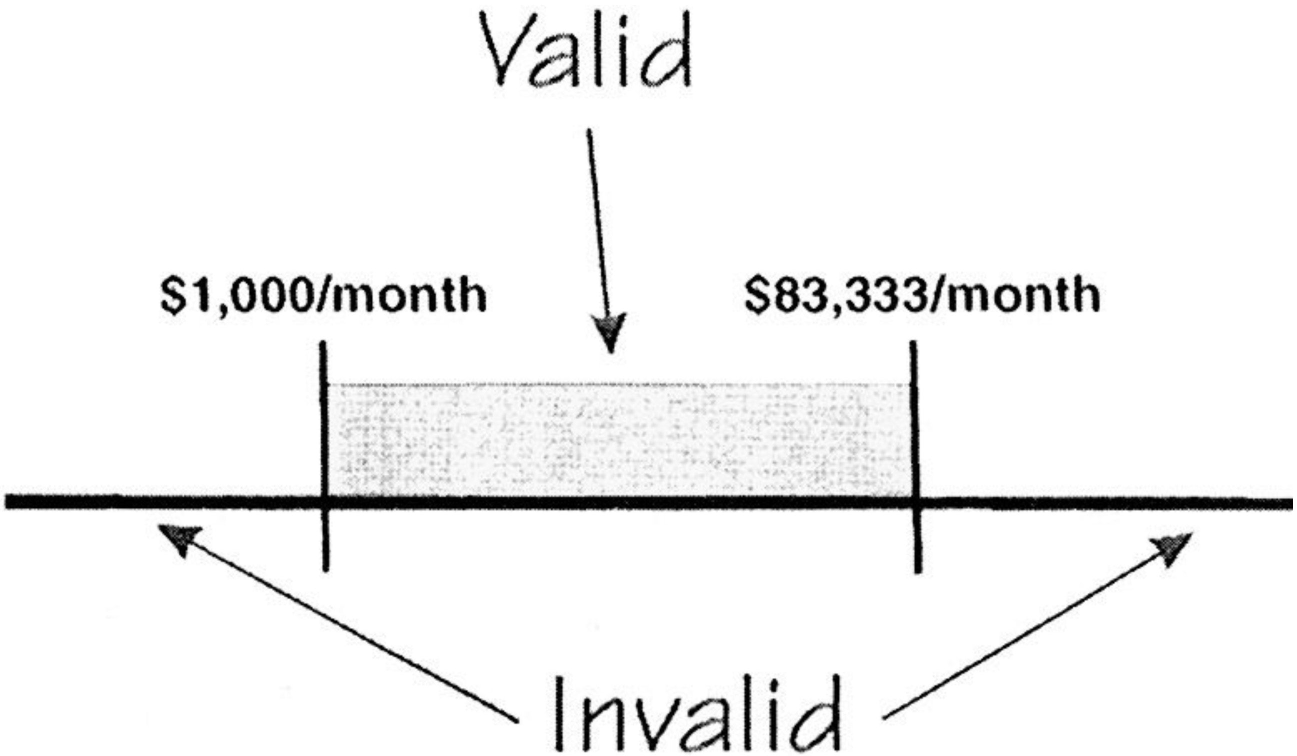
Technique: Steps

1. Identify the equivalence classes.
2. Create a test case for each equivalence class.

You could create additional test cases for each equivalence class if you have time and money.

Additional test cases may make you feel warm and fuzzy, but they rarely discover defects the first doesn't find.

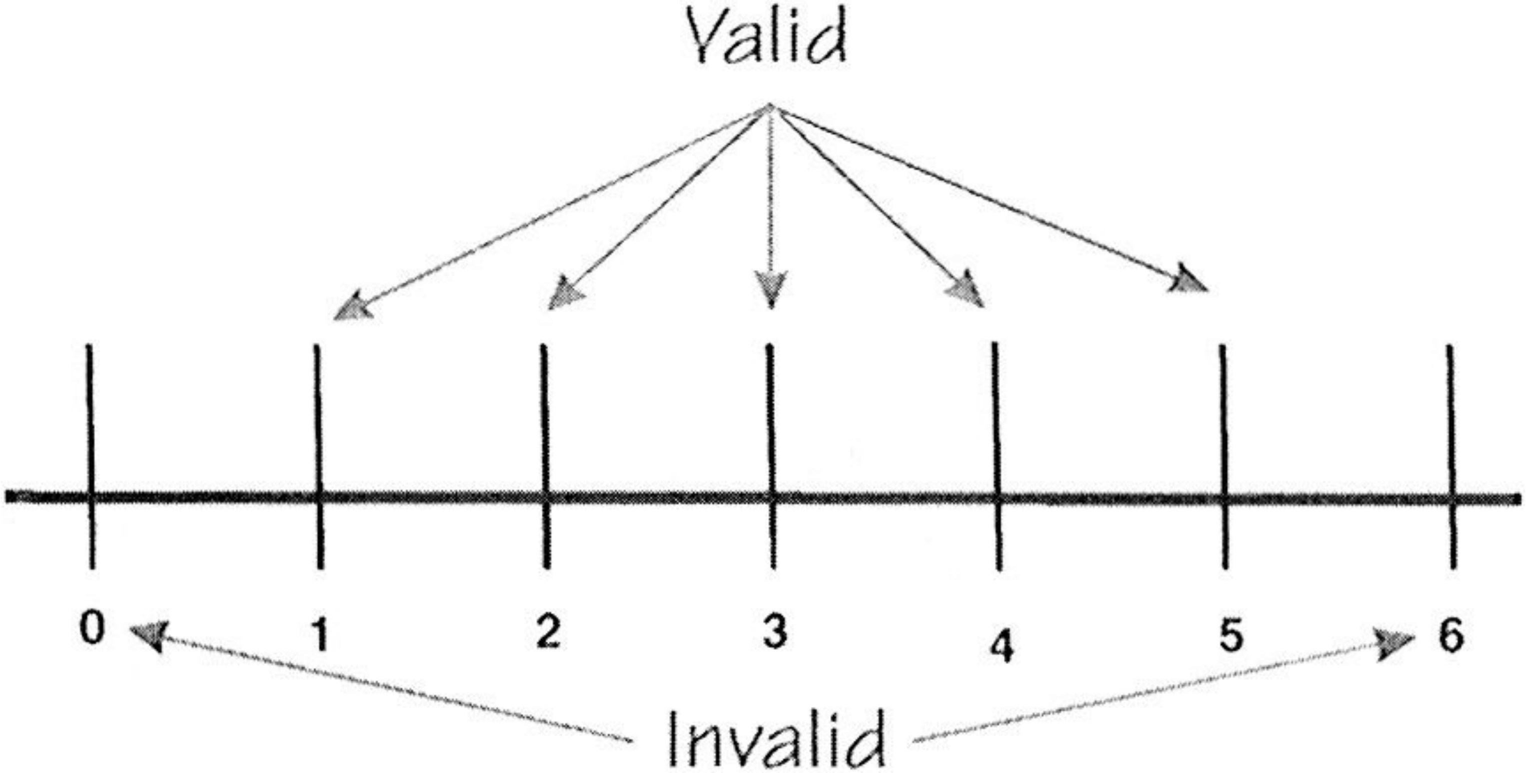
Technique: Continuous



Continuous equivalence classes

For a valid input we might choose $1,342/month. For invalids we might choose $123/month and $90,000/month.

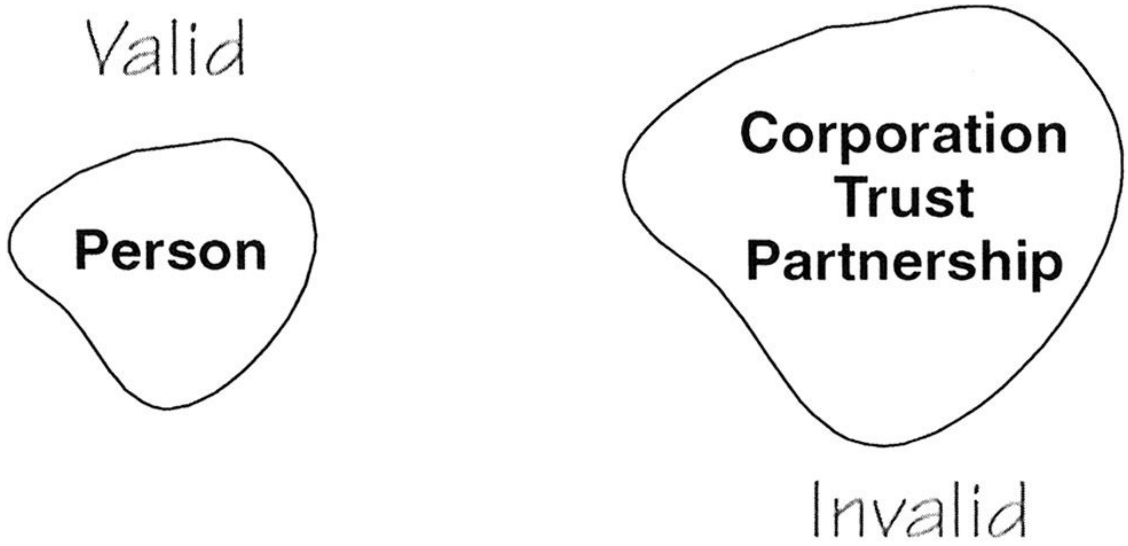
Technique: Discrete



Discrete equivalence classes

For a valid input we might choose 2 houses. Invalids could be –2 and 8.

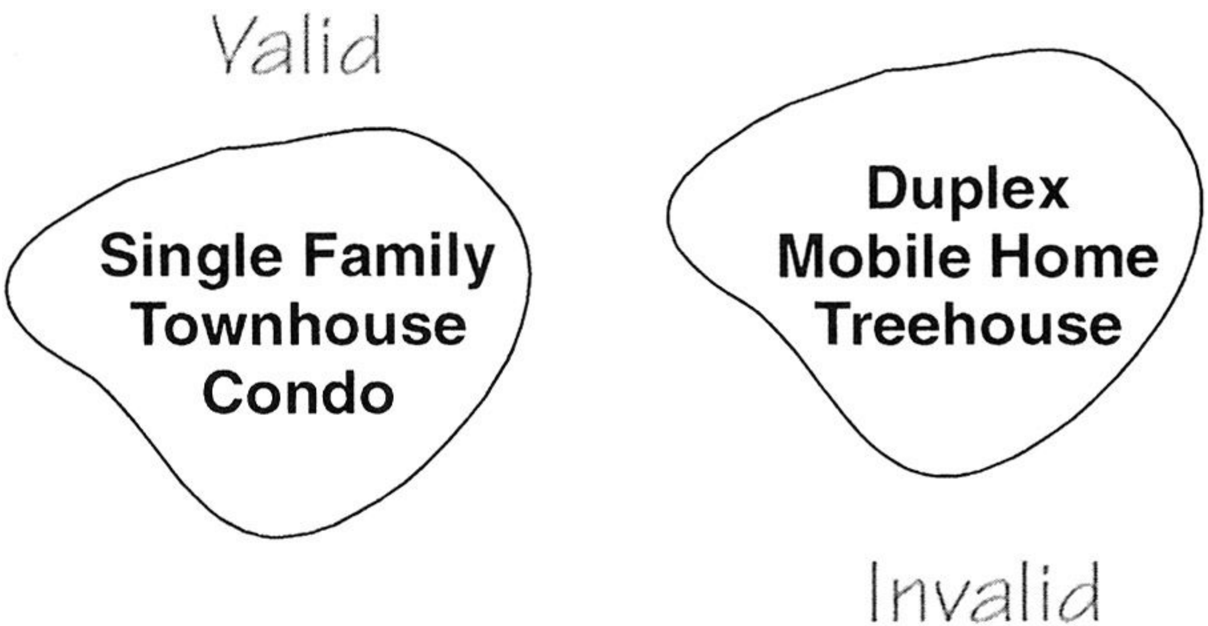
Technique: Array



Single selection equivalence classes

For a valid input we *must* use "person." For an invalid we could choose "corporation" or "trust" or any other random text string. How many invalid cases should we create? We must have at least one; we may choose additional tests for additional warm and fuzzy feelings.

Technique: Array



Multiple selection equivalence class

While the rule says choose one test case from the valid equivalence class, a more comprehensive approach would be to create test cases for each entry in the valid class. That makes sense when the list of valid values is small.

Technique: Contradictions

But, if this were a list of the fifty states, and the various territories of the United States, would you test every one of them? What if in the list were every country in the world?

The correct answer, of course, depends on the risk to the organization if, as testers, we miss something that is vital.

Technique: Combination

Rarely we will have the time to create individual tests for every separate equivalence class of every input value.

Test cases of valid data values.

Monthly Income	Number of Dwellings	Applicant	Dwelling Types	Result
$5,000	1	Person	Condo	Valid
$1,389	4	Person	SingleFam	Valid
$10,000	3	Person	Townhouse	Valid

Technique: All invalid

A test case of invalid data values.

Monthly Income	Number of Dwellings	Applicant	Dwelling Types	Result
$100	8	Partnership	Treehouse	Invalid

If the system accepts this input as valid, clearly the system is not validating the four input fields properly. If the system rejects this input as invalid, it may do so in such a way that the tester cannot determine which field it rejected. For example: **ERROR: 653X-2.7 INVALID INPUT**

Technique: One invalid

In many cases, errors in one input field may cancel out or mask errors in another field so the system accepts the data as valid. A better approach is to test one invalid value at a time to verify the system detects it correctly.

A set of test cases varying invalid values one by one.

Monthly Income	Number of Dwellings	Applicant	Dwelling Types	Result
$100	1	Person	SingleFam	Invalid
$1,342	**0**	Person	Condo	Invalid
$1,342	1	**Corporation**	Townhouse	Invalid
$1,342	1	Person	**Treehouse**	Invalid

Technique: Varying values

For additional warm and fuzzy feelings, the inputs (both valid and invalid) could be varied.

A set of test cases varying invalid values one by one but also varying the valid values.

Monthly Income	Number of Dwellings	Applicant	Dwelling Types	Result
$100	1	Person	SingleFam	Invalid
$1,342	**0**	Person	Condo	Invalid
$5,432	3	**Corporation**	Townhouse	Invalid
$10,000	2	Person	**Treehouse**	Invalid

Technique: Tips

Another approach to using equivalence classes is to examine the outputs rather than the inputs.

Divide the outputs into equivalence classes, then determine what input values would cause those outputs. This has the advantage of guiding the tester to examine, and thus test, every different kind of output. But this approach can be deceiving.

In the previous example, for the human resources system, one of the system outputs was NO, that is, Don't Hire. A cursory view of the inputs that should cause this output would yield {0, 1, …, 14, 15}. Note that this is *not* the complete set. In addition {55, 56, …, 98, 99} should also cause the NO output.

It's important to make sure that all potential outputs can be generated, but don't be fooled into choosing equivalence class data that omits important inputs.

Examples


REASONS TO STOP TESTING
THERE ARE LOTS OF REASONS WHY YOU MAY WANT TO STOP TESTING. HERE ARE A FEW...
THERE ARE BUGS EVERYWHERE
YOU NEED A BREATHER. TAKE A COFFEE BREAK
TIMES UP! RELEASE IT!
ONE BIG MAMA OF A BUG
IT'S HOME TIME
Miller
IT'S MILLER TIME. TIME TO PARTY!
No $
NO ONE IS PAYING YOU TO TEST
EVERYTHING YOU PLANNED IS COMPLETE
Of course, your plan might be rubbish, but that's not my problem.
YOU CAN'T FIND ANY MORE BUGS
THERE'S A NEW FAMILY MEMBER
AG
Andy Glover cartoontester.blogspot.com Copyright 2010

Examples: 1

Order Type
- Buy
- Sell

No invalid choices.

It reduces the number of test cases the tester must create.

Only the valid inputs {Buy, Sell} need to be exercised.

Order Type

Valid inputs: {Buy, Sell}.

Invalids: {Trade, Punt, ...}.

What about "buy", "bUy", "BUY"? Are these valid or invalid entries? The tester would have to refer back to the requirements to determine their status.

Examples: 2

Input to this field can be between one and four numeric characters (0, 1, ..., 8, 9) with a valid value greater or equal to 1 and less than or equal to 9999.

Valid inputs are {1, 23, 456, 7890}.

Invalid inputs are {-42, 0, 1.2, 12345, SQE, $#@%}.

Examples: 3

The valid symbols are {A, AA, AABC, AAC, ..., ZOLT, ZOMX, ZONA, ZRAN}. The invalid symbols are any combination of characters not included in the valid list.

Valid inputs are {A, AL, ABE, ACES, AKZOY}.

Invalid inputs are {C, AF, BOB, CLUBS, AKZAM, 42, @#$%}.

Examples: 4

Rarely will we create separate sets of test cases for each input. Generally it is more efficient to test multiple inputs simultaneously within tests. For example, the following tests combine Buy/Sell, Symbol, and Quantity.

A set of test cases varying invalid values one by one.

Buy/Sell	Symbol	Quantity	Result
Buy	A	10	Valid
Buy	C	20	Invalid
Buy	A	0	Invalid
Sell	ACES	10	Valid
Sell	BOB	33	Invalid
Sell	ABE	–3	Invalid

Applicability and Limitations

Applicability and Limitations

- Equivalence class testing can significantly reduce the number of test cases that must be created and executed. It is most suited to systems in which much of the input data takes on values within ranges or within sets. It makes the assumption that data in the same equivalence class is, in fact, processed in the same way by the system. The simplest way to validate this assumption is to ask the programmer about their implementation.
- Let your designers and programmers know when they have helped you. They'll appreciate the thought and may do it again.

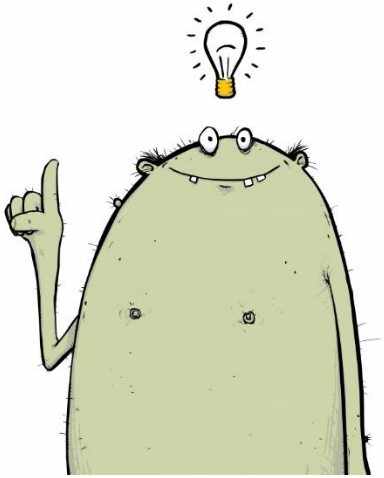
Applicability and Limitations

- Very often your designers and programmers use GUI design tools that can enforce restrictions on the length and content of input fields. Encourage their use. Then your testing can focus on making sure the requirement has been implemented properly with the tool.
- Equivalence class testing is equally applicable at the unit, integration, system, and acceptance test levels. All it requires are inputs or outputs that can be partitioned based on the system's requirements.

Summary

Summary

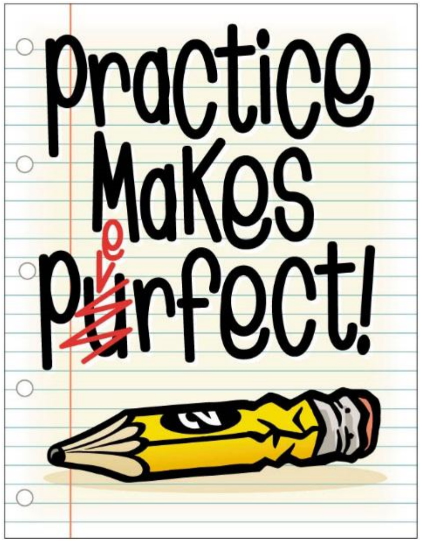
- Equivalence class testing is a technique used to reduce the number of test cases to a manageable size while still maintaining reasonable coverage.
- This simple technique is used intuitively by almost all testers, even though they may not be aware of it as a formal test design method.
- An equivalence class consists of a set of data that is treated the same by the module or that should produce the same result. Any data value within a class is equivalent, in terms of testing, to any other value.



Practice

Practice

- ZIP Code – five numeric digits.
- Last Name – one through fifteen characters (including alphabetic characters, periods, hyphens, apostrophes, spaces, and numbers).
- User ID – eight characters at least two of which are not alphabetic (numeric, special).
- Student ID – eight characters. The first two represent the student's home campus while the last six are a unique six-digit number. Valid home campus abbreviations are: AN, Annandale; LC, Las Cruces; RW, Riverside West; SM, San Mateo; TA, Talbot; WE, Weber; and WN, Wenatchee.



Practice: Answers 1

- ZIP Code – five numeric digits.

Length
Valid: 5
Invalid: 3; 20

Characters
Valid: numeric digits
Invalid: special; alphabetical

Is this Zip Code really valid? Is it real?

Example	Result	Comment
12345	Valid	Length, digits
AbcDZ	Invalid	Alphabetical
‘ –(:	Invalid	Special
129	Invalid	Length <
12345678901234567890	Invalid	Length >

Practice: Answers 2

- Last Name – one through fifteen characters (including alphabetic characters, periods, hyphens, apostrophes, spaces, and numbers).

Length

Valid: 7

Invalid: 0; 19

Characters

Valid: alphabetic; numeric; .; –; ; “

Invalid: all other special

Example	Result	Comment
Co.- 1”	Valid	Length, characters
	Invalid	Length <
ABCDEFghijklmnopqrs	Invalid	Length >
!@#;$%:	Invalid	Other special

Practice: Answers 3

- User ID – eight characters at least two of which are not alphabetic (numeric, special).

Length

Valid: 8

Invalid: 2; 11

Number of numeric and special characters

Valid: 2

Invalid: 1; 10

Example	Result	Comment
1!abcDYZ	Valid	Length, number
2%	Invalid	Length <
0#?(cyzagq4	Invalid	Length >
abcptu6w	Invalid	Number <
“(/\,.123+	Invalid	Number >

Practice: Answers 4

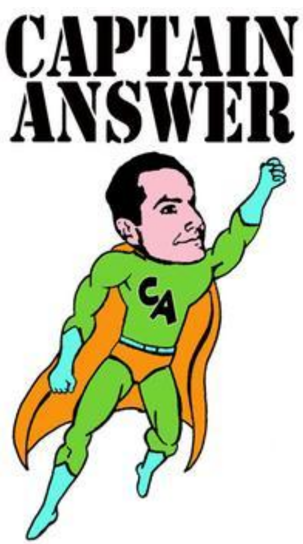
- Student ID – eight characters. The first two represent the student's home campus while the last six are a unique six-digit number. Valid home campus abbreviations are: AN, Annandale; LC, Las Cruces; RW, Riverside West; SM, San Mateo; TA, Talbot; WE, Weber; and WN, Wenatchee.

Length	Characters position	Campus	Unique
Valid: 8	Valid: first 2	Valid: in the list	Valid: unique
Invalid: 5; 10	Invalid: 3^{d} and 4th	Invalid: other	Invalid: not unique

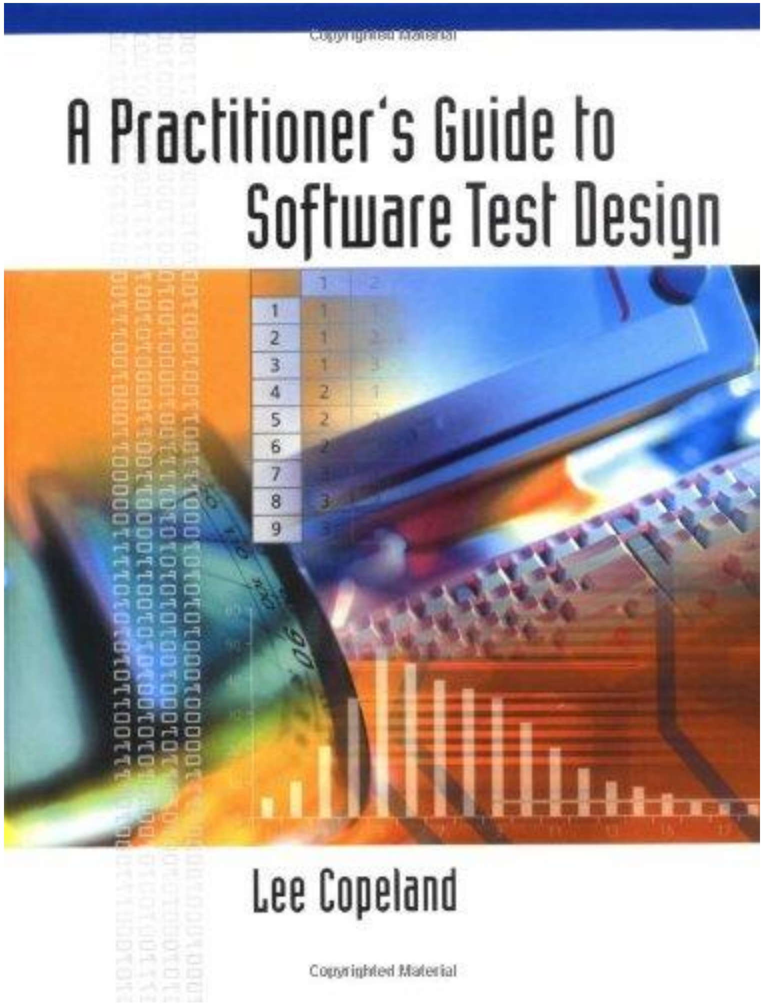
Example	Result	Comment
AN123409	Valid	Length, position, campus, unique
LC136	Invalid	Length <
TA98765432	Invalid	Length >
12SM4446	Invalid	Position
AC963201	Invalid	Campus
AN123409	Invalid	Not unique

Practice: Answers 5

N	ZIP Code	Last Name	User ID	Student ID	Result
1	12345	Co.- 1”	1!abcDYZ	AN123409	Valid
2	**AbcDZ**	KolirtypUedv	LDKW9456	LC874562	Invalid
3	**‘ –(:**	.– 2”34567890'.	Poief63t	RW456732	Invalid
4	**129**	....	&)ˆASGYK	SM687414	Invalid
5	**12345678901234567890**	------------	KOfd27,.	TA312458	Invalid
6	67890		!@#;$%:?	WE965874	Invalid
7	11111	**!@#;$%:**	12378964	WN221133	Invalid
8	23487	**ABCDEFghijklmnopqrs**	09876,=_	SM747498	Invalid
9	89453	'''	**2%**	TA321987	Invalid
10	09342	PODSAF	**0#?(cyzagq4**	WE126542	Invalid
11	34567	lju77 fsd 5	**abcptu6w**	WN369874	Invalid
12	09789	Lopwefdvc	**“(/\,.123+**	AN546887	Invalid
13	19823	se.rt3456	Ty_1236*	**LC136**	Invalid
14	73287	1594	;ldfskt8	**TA98765432**	Invalid
15	64785	43	3333UOPQ	**12SM4446**	Invalid
16	98883	R	pn7f1uN6	**AC963201**	Invalid
17	19823	yu	n8m!c~2{	**AN123409**	Invalid



References



Copyrighted Material
A Practitioner's Guide to
Software Test Design
Lee Copeland
Copyrighted Material


Q&A
You have
Questions
We have
Answers


THANK YOU


Have a nice day!