Software Testing & Quality Assurance

Module 4: Equivalence Class Testing (ECT)

- Background and Motivation
- Weak Normal ECT
- Strong Normal ECT
- Weak Robust ECT
- Strong Robust ECT

Credits & Readings

- The material included in these slides are mostly adopted from the following books:
  - *Software Testing: A Craftsman's Approach*, by Paul Jorgensen, CRC PRESS, third edition, ISBN: 0-8493-7475-8
  - Cem Kaner, Jack Falk, Hung Q. Nguyen, *"Testing Computer Software"* Wiley (see also http://www.testingeducation.org/)
  - Cem Kaner, James Bach, Bret Pettichord, *"Lessons Learned in Software Testing",* Wiley
  - Paul Ammann and Jeff Offutt, *"Introduction to Software Testing"*, Cambridge University Press
  - Kent Beck, *"Test-driven Development by Example"* Addison-Wesley
  - Robert Binder, *"Testing Object-Oriented Systems: Models, Patterns, and Tools"* Addison-Wesley
  - Glen Myers, *"The Art of Software Testing"*

Motivation

- One weakness of Boundary Value Testing (BVT) is that it derives test cases with
  - Massive redundancy
    - For instance, <5,5,5>, <6,6,6> and <100,100,100> are all *redundant* test cases for the triangle problem since they are "treated the same" i.e. "traversing the same execution path"
  - Serious gaps (i.e. sense of incomplete testing)
- Equivalence Class Testing (ECT) attempts to alleviate these problems
- It echoes the two deciding factors of BVT:
  - Robustness (i.e. handling invalid inputs effectively)
  - Single/multiple fault assumption (Weak vs. Strong ECT)

Equivalence Class Testing

- Partition the set of all test cases into mutually *disjoint subsets* whose union is the entire set
- Choose *one test case from each subset*
- Two important advantages of ECT:
  - The fact that the entire set is represented provides a form of completeness
  - The disjoint-ness assures a form of non-redundancy

Equivalence Class Selection

- The key point in ECT is the choice of the *equivalence criteria (relation)* that determine the classes
  - If the equivalence classes are chosen wisely, the potential redundancy among test cases is greatly reduced
  - When you define equivalence classes on the input domain watch for inputs being "treated the same" (they should belong to the same class)
  - Also, attempt to define equivalence classes on the output range of the program
- We will discuss *four different types* of ECT
  - Weak Normal ECT **("Weak/Strong" refers to the single/multiple fault assumption)**
  - Strong Normal ECT
  - Weak Robust ECT **("Robust" relates to the consideration of invalid inputs)**
  - Strong Robust ECT

Applicability

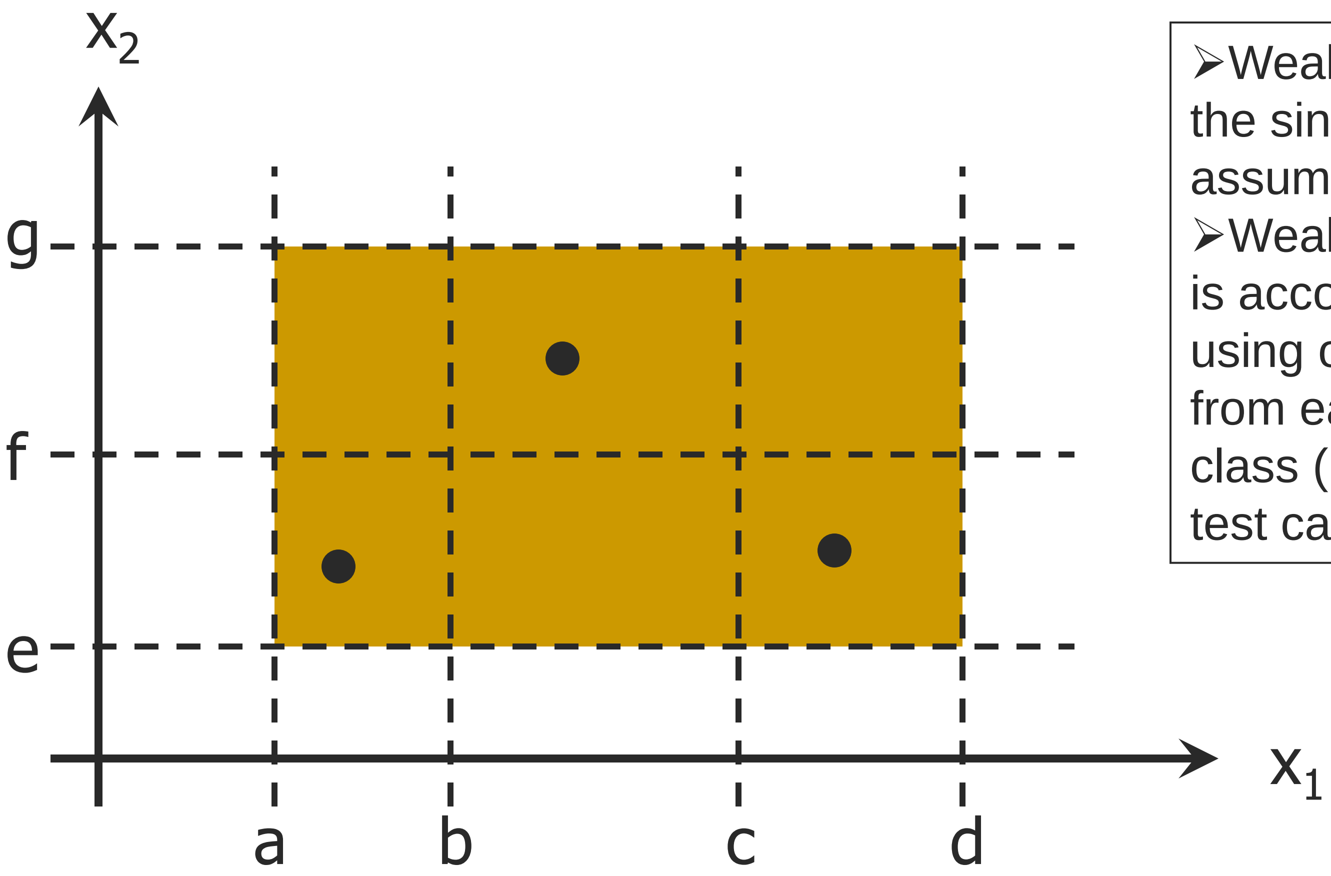
- ECT is appropriate when the system under test can be expressed as a function of one or more variables, whose domains have well defined *intervals* *(i.e. equivalence classes)*
- Example: A two-variable function F(x1,x2)

$a \le x_1 \le d$, with intervals [a,b), [b,c), [c,d]*

$e \le x_2 \le g$, with intervals [e,f), [f,g]

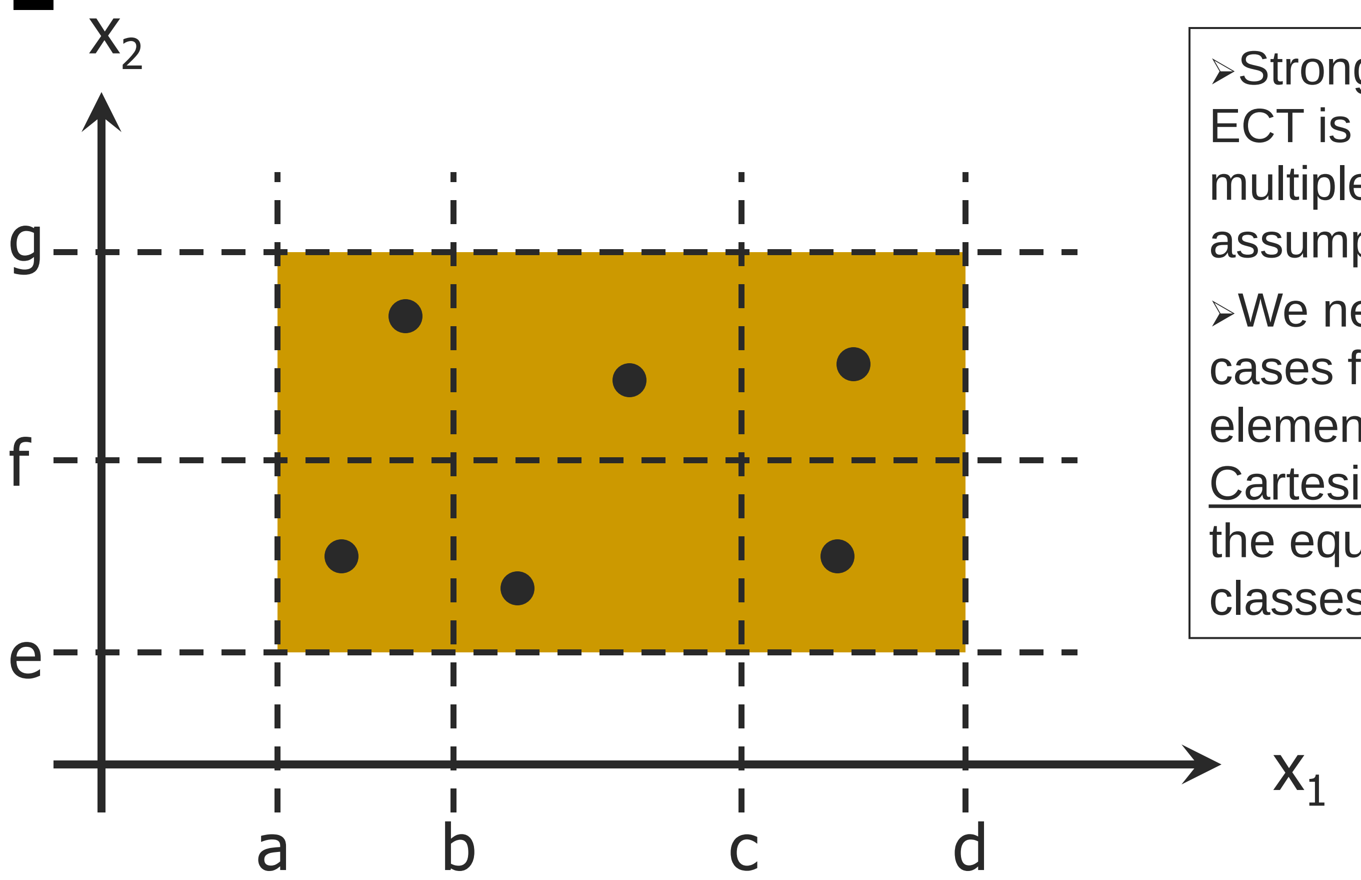
* Where [indicates a closed interval endpoint and) indicates an open interval endpoint.

Weak Normal ECT



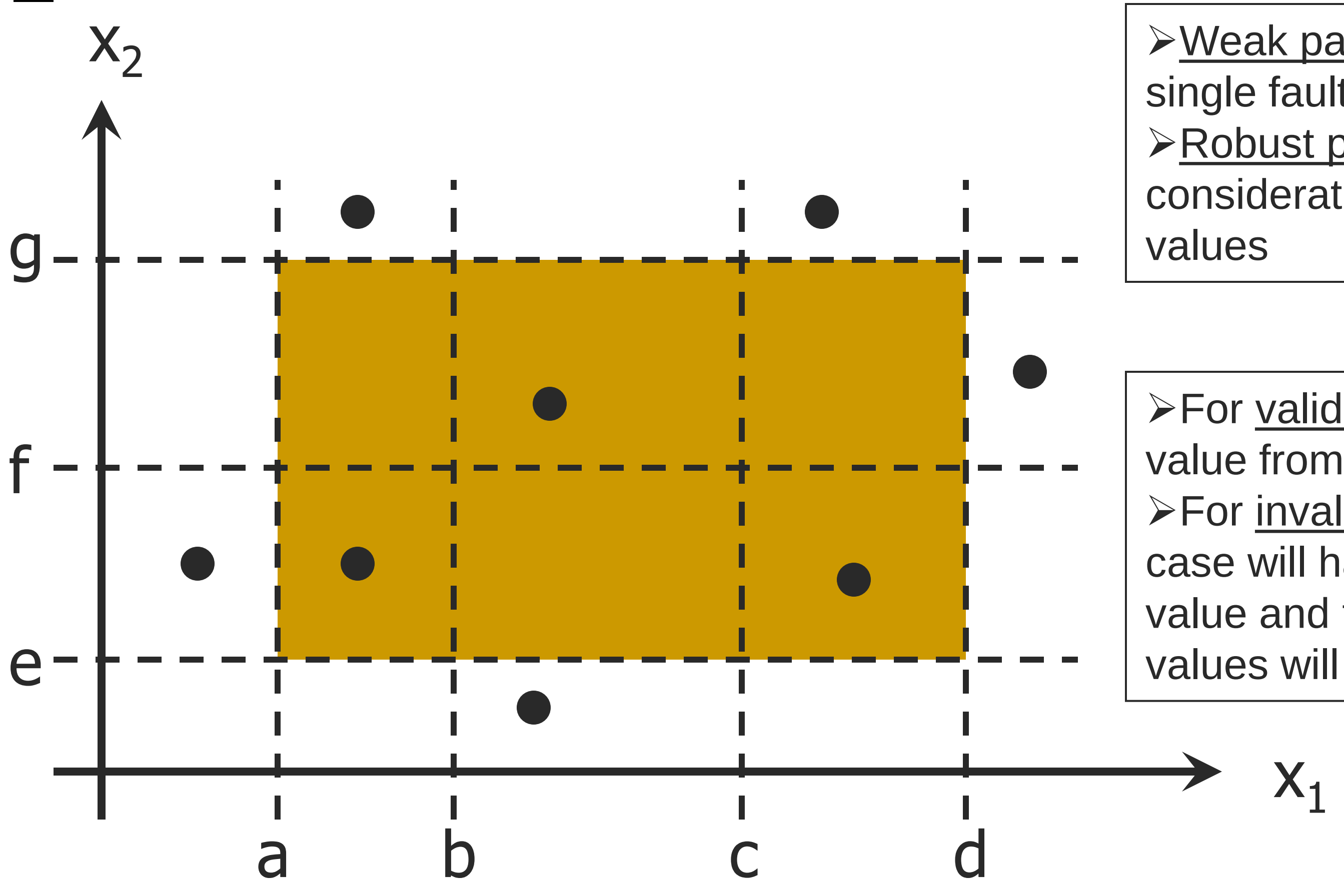
- ➢Weak part refers to the single fault assumption
- ➢Weak Normal ECT is accomplished by using one variable from each equivalence class (interval) in a test case

Strong Normal ECT



- Strong Normal ECT is based on the multiple fault assumptions
- We need test cases from each element of the Cartesian product of the equivalence classes

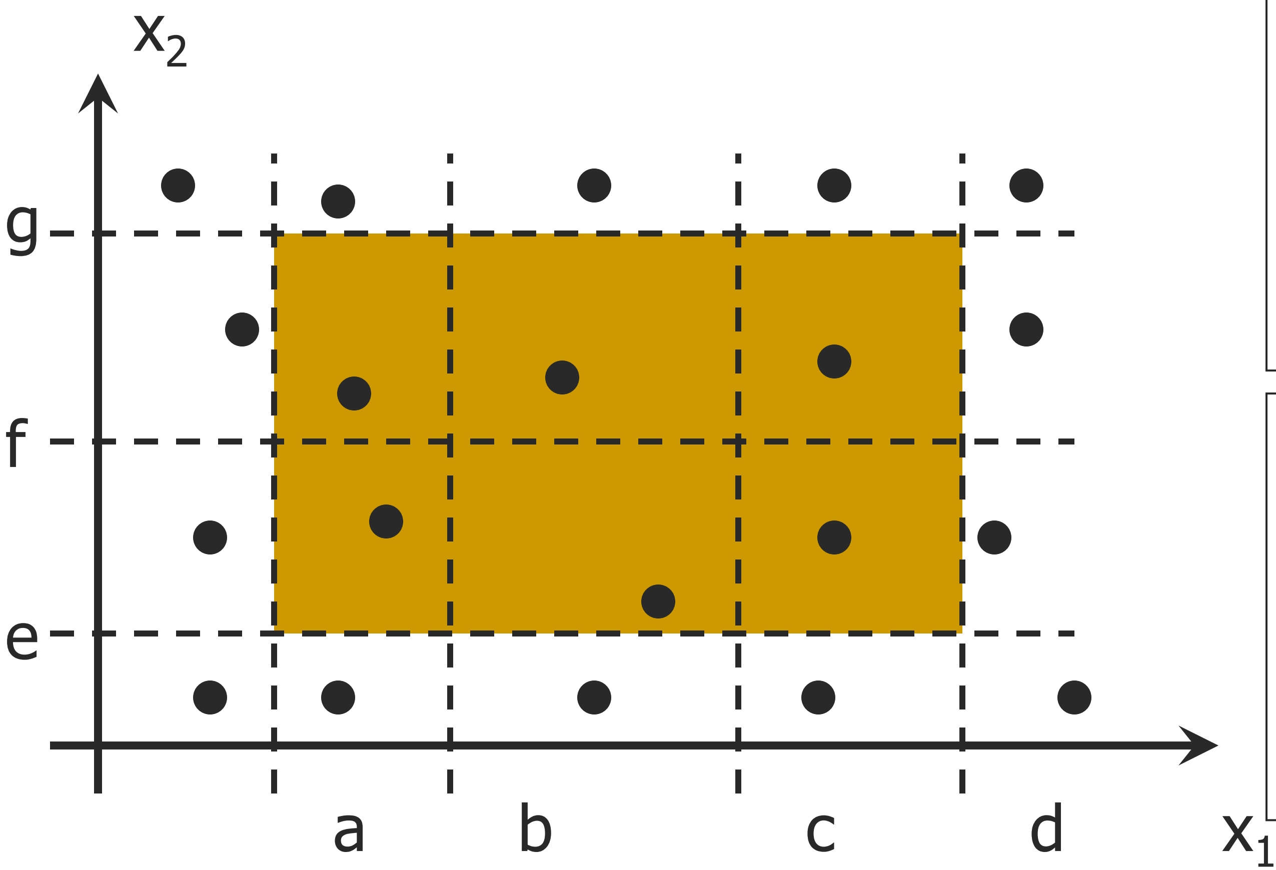
Weak Robust ECT



- <u>Weak part</u> refers to the single fault assumption
- <u>Robust part</u> comes from consideration of invalid values

- For <u>valid inputs</u>, use one value from each valid class
- For <u>invalid inputs</u>, a test case will have one invalid value and the remaining values will all be valid

Strong Robust ECT



- Robust part comes from consideration of invalid values
- Strong part refers to the multiple fault assumption

- Test cases are obtained from each element of the Cartesian product of all the equivalence classes as shown in the figure

Selection of ECT test cases

- The inputs are mechanically selected from the *approximate middle* of the corresponding equivalence class (interval)
- A mechanical selection of input values makes no consideration of the domain knowledge
  - This is a problem with “automatic” test case generation

Limitations of Robust ECT

- Often, the specifications do not define what the expected output for an invalid test case should be
  - One could argue that this is a deficiency of the specs
  - Testers spend a lot of time defining expected outputs for these cases
- Strongly typed languages eliminate the need for the consideration of invalid inputs
  - ECT was introduced during the time when languages such as FORTRAN and COBOL were dominant and this error was common

Triangle Problem: Output (Range) Equivalence Classes

- Four possible outputs:
  - Not a Triangle
  - Isosceles
  - Equilateral
  - Scalene

- We can use these to identify the following output (range) equivalence classes:

R1= { <a, b, c>: the triangle with sides a, b, c, is equilateral}

R2= { <a, b, c>: the triangle with sides a, b, c, is isosceles}

R3= {<a, b, c>: the triangle with sides a, b, c, is scalene}

R4= {<a, b, c>: sides a, b, c do not form a triangle}

Weak Normal Test Cases

Test Case	a	b	c	Expected Output
WN1	5	5	5	Equilateral
WN2	2	2	3	Isosceles
WN3	3	4	5	Scalene
WN4	4	1	2	Not a Triangle

Weak Normal ECT contains one test case from each equivalence class

Note: the Strong Normal ECT is identical with the Weak Normal ECT because no valid subintervals of variables a, b and c exist

Weak Robust Test Cases

Test Case	a	b	c	Expected Output
WR1	-1	5	5	a not in range
WR2	5	-1	5	b not in range
WR3	5	5	-1	c not in range
WR4	201	5	5	a not in range
WR5	5	201	5	b not in range
WR6	5	5	201	c not in range

- In addition to the weak part (previous WN1-4 test cases)
- The robust part considers some invalid values for variables a, b and c
- Each test case has one invalid value and the remaining values are all valid

Strong Robust Test Cases

Test Case	a	b	c	Expected Output
SR1	-1	5	5	a not in range
SR2	5	-1	5	b not in range
SR3	5	5	-1	c not in range
SR4	-1	-1	5	a, b not in range
SR5	5	-1	-1	b, c not in range
SR6	-1	5	-1	a, c not in range
SR6	-1	-1	-1	a, b, c not in range

- Only a subset (a “corner” of the cube (3D-space) of the additional strong robust equivalence class test cases are shown in the table

Triangle Problem:
Input (Domain) Equivalence Classes

A richer set of test cases can be obtained if we base equivalence classes on the input domain as follows (R1-R4 vs. D1-D11):

D1= $\{<a,b,c> \mid a = b = c\}$ — *all sides are equal*

D2= $\{<a,b,c> \mid a = b,\ a \neq c\}$ — *exactly one pair is equal*
D3= $\{<a,b,c> \mid a = c,\ a \neq b\}$
D4= $\{<a,b,c> \mid b = c,\ a \neq b\}$

D5= $\{<a,b,c> \mid a \neq b,\ a \neq c,\ b \neq c\}$ — *none is equal*

D6= $\{<a,b,c> \mid a > b+c\}$ — *values don't form a valid triangle*
D7= $\{<a,b,c> \mid a = b+c\}$
D8= $\{<a,b,c> \mid b > a+c\}$
D9= $\{<a,b,c> \mid b = a+c\}$
D10= $\{<a,b,c> \mid c > a+b\}$
D11= $\{<a,b,c> \mid c = a+b\}$

The *NextDate* Application: Input (Domain) Equivalence Classes

- The *NextDate* function returns the date of the day after the input date
- It uses 3 input parameters: *month, day, year* which have intervals

Invalid input:

M1= {month < 1}
M2= {month > 12}

D1= {day < 1 }
D2= {day > 31}

Y1= {year < 1812}
Y2= {year > 2012}

Useful criteria for choosing equivalence classes:

- If the input date is not at the end of the month the program will simply increment the day value, however
- If the input date is at the end of the month, it will force the program to change the day to 1 and increment the month
- If the input date is at the end of the year, it will force the program to reset both the day and the month to 1 and increment the year
- The leap year makes determining the last day of the month interesting

Weak Normal Test Cases

Test Case	Month	Day	Year	Expected Output
WN1	6	14	2000	6/15/2000
WN2	7	29	1996	7/30/1996
WN3	2	30	2002	Invalid input date
WN4	6	31	2000	Invalid input date

One test case from each equivalence class

<u>Equivalence classes</u>:

M1= {month | month has 30 days}
M2= {month | month has 31 days}
M3= {month | month is February}

D1= {day | $1 \le$ day ≤ 28}
D2= {day | day = 29}
D3= {day | day = 30}
D4= {day | day=31}

Y1= {year | year = 2000 special treatment}
Y2= {year | year is a leap year}
Y3= {year | year is a common year}

NextDate Discussion

- There are 36 strong normal test cases: (3x4x3) i.e. M1-M3 x D1-D4 x Y1-Y3 (see previous slide)
- Some redundancy creeps in
  - Testing February 30 and 31 for three different types of years seems unlikely to reveal errors
- There are 150 strong robust test cases (adding the invalid input – see previous slide) : M1-M5 x D1-D6 x Y1-Y5

ECT for the Commission Problem

- More typical of commercial computing
- Contains a mix of computation and decision making
  - See problem statement in chapter 2

Valid classes of the input variables are:

L1 = {locks : $1 \le$ locks ≤ 70}
L2 = {locks = -1}

S1 = {stocks : $1 \le$ stocks ≤ 80}

B1 = {barrels : $1 \le$ barrels ≤ 90}

Invalid classes of the input variables are:

L3 = {locks: locks=0 OR locks < -1}
L4 = {locks: locks > 70}

S2 = {stocks: stocks < 1}
S3 = {stocks: stocks > 80}

B2 = {barrels: barrels < 1 }
B3 = {barrels: barrels > 90 }

See chapter 6 for the detailed equivalence class test cases

Guidelines and Observations

- Equivalence Class Testing is appropriate when input data is defined in terms of intervals and sets of discrete values
- Equivalence Class Testing is strengthened when combined with Boundary Value Testing
- Complex functions, such as the NextDate program, are well-suited for Equivalence Class Testing
- Several tries may be required before the “right” equivalence relation is discovered
- Strong equivalence takes the presumption that variables are independent
  - If that is not the case, redundant test cases may be generated

In class activity

- For the triangle problem create
  - Weak Normal test cases
  - Strong Normal test cases
  - Weak Robust test cases
  - Strong Robust test cases