SQA

MANUAL TESTING

সফটওয়্যার টেস্টের সম্পর্কে আমরা জেনে এসেছি বিগত ক্লাসগুলো থেকে। এখন আমরা দেখে নিবো কীভাবে টেস্ট কেইস ডিজাইন করতে হয়।

Test Case Design

আপনাকে প্রব্লেম স্টেটমেন্ট এ যে সমস্যাটা সমাধান করতে বলেছে সেটাই মোট কতগুলো ইনপুট দিয়ে চেক করা হবে, সেটা হল টেস্ট কেস সংখ্যা। আমরা সাধারনত দুইভাবে এই টেস্ট কেস ডিজাইন করতে পারি –

1. Static test design technique
2. Dynamic test design technique

Static Test Design Technique

স্ট্যাটিক টেস্টিং টেকনিক পরীক্ষা করা কম্পোনেন্ট ও সিস্টেমের বিভিন্ন পদ্ধতিকে সংক্ষেপে গুছিয়ে আনে।

স্ট্যাটিক টেস্টের অন্তর্ভুক্ত হলো -

- রিভিউ (ম্যানুয়াল একটিভিটি)
- স্ট্যাটিক এনালাইসিস (বেশিরভাগ সময় টুল ভিত্তিক একটিভিটি) টোল হলো এমন কিছু যা আমাকে কোন কাজ করতে সহযোগিতা করে। আমরা এখানে স্ট্যাটিক এনালাইসিস করার সময় একটা টুল ব্যবহার করব যেটা হচ্ছে কন্ট্রোল ফ্লো গ্রাফ।

স্ট্যাটিক টেস্টের কম্পোনেন্ট ডায়ানামিক মেথড

- স্ট্যাটিক টেস্ট ফেইলিউর না বরং ডিফেক্ট খুঁজে বের করে
- কার্যকর কোডই না বরং কনসেপ্টগুলো ও ভালোভাবে পরীক্ষা করে দেখা হয়।
- কোডে ব্যবহার করার আগেই ডিফেক্ট? দেভিয়েশনগুলোকে খুঁজে বের করা হয়
- ডায়নামিক টেস্টে না পাওয়া ডিফেক্টগুলোও স্ট্যাটিক টেস্টের মাধ্যমে পাওয়া যেতে পারে।

হাই কোয়ালিটি ডকুমেন্ট থেকে হাই কোয়ালিটি প্রোডাক্ট পাওয়া যায়

- পরীক্ষা করে দেখা স্পেসিফিকেশনগুলোর মধ্যে কোন ভুল না থাকলেও স্পেসিফিকেশন গুলোর ইন্টারপ্রিটেশন ও ডিজাইন তৈরিতে ভুল থাকতে পারে।

Static Test Design Technique

স্ট্যাটিক এনালাইসিস এর সংজ্ঞাঃ

স্ট্যাটিক এনালাইসিস হলো একটি টেস্ট অবজেক্ট কে বিশ্লেষণ করে দেখা টেস্ট অবজেক্ট গুলোকে কার্যকর না করেই। যেমনঃ সোর্স কোড, স্ক্রিপ্ট, রিকোয়ারমেন্ট)

General Aspects/1

স্ট্যাটিক এনালাইসিস দ্বারা পরীক্ষা করা হয় সেগুলো হলোঃ

- প্রোগ্রামিং রুলস এবংস্ট্যান্ডার্ড
- প্রোগ্রাম ডিজাইন (কন্ট্রোল ফ্লো এনালাইসিস)
- ডাটার ব্যবহার (ডাটা ফ্লো এনালাইসিস)
- প্রোগ্রাম স্ট্রাকচারের জটিলতা। (ম্যাট্রিক্স যেমনঃ The Cycloramic Number)

সকল টেস্ট অবজেক্টের একটা ফরমাল স্ট্রাকচার থাকতে হবে

- এটি খুবই গুরুত্বপূর্ণ যখন টুলগুলো পরীক্ষা করা হয়
- অনেকসময়ে ডকুমেন্টগুলো ফরমালি তৈরি করা হয়ে থাকে না। ডকুমেন্টেশন যদি ফরমাল ভাবে তৈরি করা হতো তাহলে সেটা থেকেই একটা সফটওয়্যার তৈরি করা যেত।
- প্র্যাক্টিস মডেলিং প্রোগ্রামিং এবং ল্যাঙ্গুয়েজ স্ক্রিটিং নো রুল এবং কিছু ডায়াগ্রাম এর সাথে মানিয়ে চলে।

General Aspects/2

টুল ভিত্তিক এনালাইসিস তুলনামূলক সহজ রিভিউ করার জন্য। আমরা কোনো ডকুমেন্টেশন রিভিউ করবো কীনা তা স্ট্যাটিক এনালাইসিস থেকে বুঝা যায়। কম্পাইলার ও এনালাইজিং টুল হিসেবে যেসব টুল ব্যবহৃত হতে পারে।

কম্পাইলার

প্রোগ্রাম সোর্স কোডের ভিতরে সিনটেক্স এরোর খুজে বের করে।

- প্রোগ্রামের রেফারেন্স ডাটা তৈরী করে। যেমনঃ প্রোগ্রামের **Cross reference list, Call hierarchy, Symbol table** ইত্যাদি।
- ভ্যারিয়েবলের মধ্যে কন্সিস্টেন্সি যাচাই করে
- আনডিক্লারড ভ্যারিয়েবল ও ডেড কোড খুজে বের করে।
- এনালাইজার আর যেসব বিষয় তুলে ধরে -
    - ✓ **Convention and standards**
    - ✓ **Metrics of complexity**
    - ✓ **Object coupling**

Ostad

Control Flow Analysis /1

Aim

প্রোগ্রাম কোডের ভুলিভাবে তৈরী হওয়ার কারনে যেসব ডিফেক্ট হয় তা খুঁজে বের করা।

Method

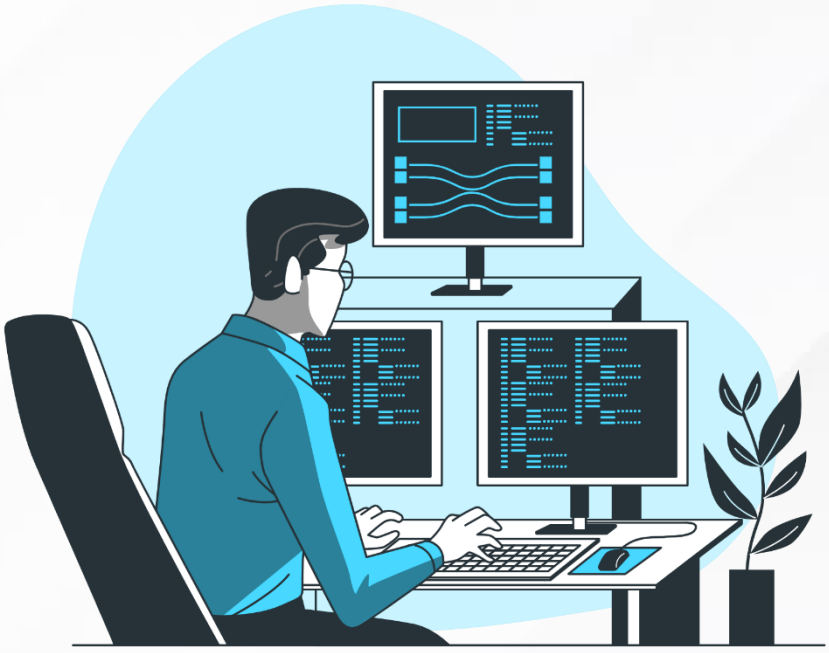
1. কোড স্ট্রাকচার টা **Control Flow Graph** এর মাধ্যমে দেখানো হয়।
2. **Directed Graph**
    a. **Statement** এর **Sequence** গুলো **Nodes** দিয়ে বোঝায়।
    b. **Decision** ও **Loop** এর মধ্যে **Control flow transfer** বোঝানো হয় **Edge** দিয়ে।
    c. **Tool** ভিত্তিক **Construction**

Control Flow Analysis /2

Result

- সহজে বোঝা যায় এমন প্রোগ্রাম কোডের অভারভিউ।
- যেকোনো অসঙ্গতি সহজেই শনাক্ত করা যায়। ডিফেক্টগুলো যেমনঃ
    - **Loops exited by jumps**
    - **Dead branches**
    - **Multiple returns**
- **Control flow graph** হলো **Flow chart** এর সহজ রুপ।

Metrics And Their Computation



- ম্যাট্রিক্স দিয়ে প্রোগ্রামের কোয়ালিটির একটি নির্দিষ্ট ধরন যাচাই করা যায়।
    - ম্যাট্রিক্সটি শুধুমাত্র সেই ধরনের সাথেই সামঞ্জস্যপূর্ণ হয়ে থাকে।

- প্রোগ্রামটির স্ট্যাটিক কম্পলেক্সিটি পরিমাপ করা যায় ম্যাট্রিক্স দিয়ে।
    - আপাতত ১০০ টার বেশি ভিন্ন ভিন্ন ম্যাট্রিক্স রয়েছে।

- বিভিন্ন ম্যাট্রিক্স বিভিন্ন ধরনের প্রোগ্রাম কম্পলেক্সিটি তুলে ধরে।
    - Program size (eg. Lines of Code - LOC)
    - Program control structure (eg. Cyclomatic complexity)
    - Data control structure (eg. Halstead complexity)

- প্রোগ্রামের একই এট্রিবিউট কে তুলে ধরলেও বিভিন্ন ম্যাট্রিক্সের মধ্যে তুলনা করা কঠিন।

Metrics And Their Implementation

Cyclomatic Number v(G)

এটি একটি পরিমাপক পদ্ধতি যা একটি প্রোগ্রামের সোর্সকোড দ্বারা লিনিয়ারভাবে স্বতন্ত্র পথের সংখ্যা পরিমাপ করে। এটি ১৯৭৬ সালে থমাস জে. ম্যাককেব, সিনিয়র দ্বারা উদ্ভাবিত হয়েছিল।

- **Control flow graph** এর উপর ভিত্তি করে প্রোগ্রামের স্ট্যাটিক কম্পলেক্সিটি মাপার ম্যাট্রিক

- **Linear independent program path** পরিমাপ করা হয় **Testability** ও **maintainability** এর নির্দেশক হিসেবে।

- **Cyclomatic number** তৈরী হয়ে থাকে -
    - **Number of edge - e**
    - **Number of nodes - n**
    - **Number of inspected independent program parts - p (mostly 1)**

- **v(G)** ১০ পর্যন্ত মান গ্রহনযোগ্য। এর উপরে হলে কোডটিকে ইম্প্রুভ করতে বা পুনরায় কাজ করতে হবে। **(best practice, McCabe)**

Find the cyclomatic number using the formula:
v(G) = e-n+2p

Cyclomatic Complexity :

1 program part, p = 1

15 nodes, n = 15

20 edges, e = 20

The number to a cyclomatic complexity

v(G) = e-n+2p

v(G) = 7

Cyclomatic Complexity (by Thomas J. mcCabe) - Implication

কোড রিভিউ করার জন্য **cyclomatic complexity** ব্যবহৃত হতে পারে।

Independent decision এর সংখ্যা হিসেবেও **cyclomatic complexity** হিসেব করা হতে পারে। যদি দুই জায়গার হিসেব ভিন্ন হয়। তার কারন হতে পারে -

- কোনো অপ্রয়োজনীয় ব্রাঞ্চ
- কোনো অনুপস্থিত ব্রাঞ্চ

Cyclomatic complexity প্রোজনীয় টেস্ট কেসের সংখ্যার নির্দেশ ও দিয়ে থাকে। (সিদ্ধান্ত পুর্নাঙ্গ করতে)

Static Analysis

- কম্পাইলার, এনালাইজার টুল ব্যবহার করে স্ট্যাটিক এনালাইসিস করা যায় প্রোগ্রাম চালু না করেই।
- টুল ব্যবহার করে ইন্সপেকশন করা থেকে সহজে ও কম কষ্টে প্রোগ্রামের স্ট্যাটিক এনালাইসিস করা যায়।

Analysis result

- প্রোগ্রামে থাকা **Dead branch** ও **Unreachable codes** গুলো **Control flow diagram** দেখিয়ে দেয়।
- অসঙ্গত ডাটা পাওয়া যায় **Data flow** এনালাইসিস করে।
- স্ট্রাকচারাল কম্পলেক্সিটি পেতে ম্যাট্রিক্স ব্যবহার করা যায়, যা কতটুকু টেস্ট করা লাগবে তা অনুমান করতে সাহায্য করে।

Test Design Technique

টেস্ট কেস ডিজাইন পদ্ধতিগুলি সফটওয়্যার অ্যাপ্লিকেশনের জন্য টেস্ট পরিকল্পনা, ডিজাইন এবং পরীক্ষার করার মূল বিষয়। এই পদ্ধতিগুলি বিভিন্ন ধাপ থাকে করে, যা সফটওয়্যার প্রোগ্রামে বাগ বা অন্যান্য ত্রুটিগুলি খুলতে টেস্ট কেসগুলির কার্যক্ষমতা নিশ্চিত করতে লক্ষ্য করে।

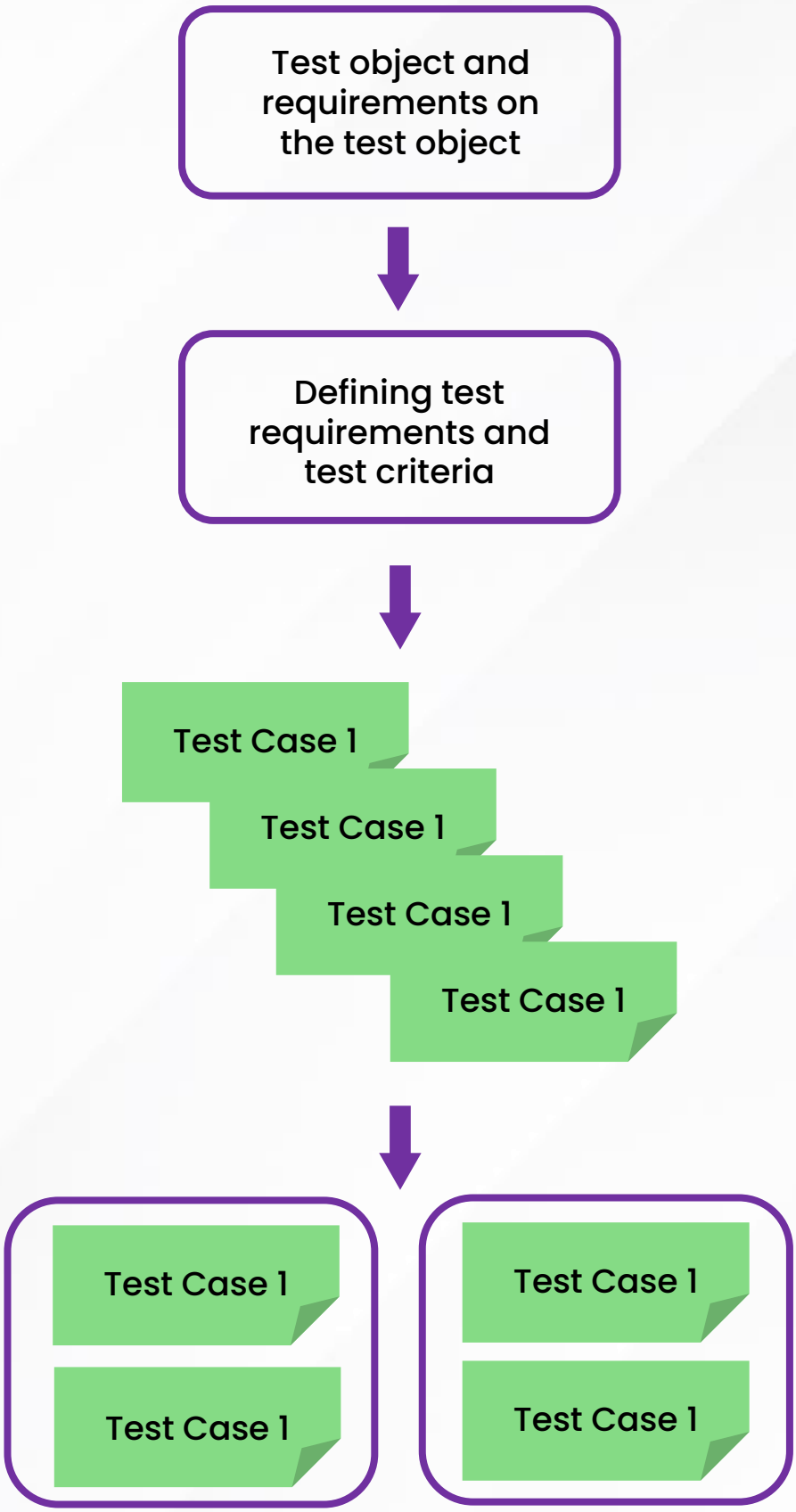
Designing the test

রিকোয়ারমেন্ট থেকে টেস্ট কেস তৈরী করা।

টেস্ট কেস বানানোর একটা নিয়ন্ত্রিত প্রক্রিয়া থাকা লাগবে।

প্রজেক্টের সীমাবদ্ধতা ও ব্যবহৃত প্রক্রিয়ার ম্যাচিউরিটির উপর নির্ভর করে টেস্ট কেস ফরমাল বা ইনফরমাল ভাবে বানানো যেতে পারে।

টেস্ট কেস সবসময় সন্ধানযোগ্য হতে হবে।


Test object and requirements on the test object
Defining test requirements and test criteria
Test Case 1
Test Case 1
Test Case 1
Test Case 1
Test Case 1
Test Case 1
Test Case 1
Test Case 1

Deriving Test Case From Requirement

Test Object :

The subject to be examined : যেকোনো ডকুমেন্ট বা কম্পাইল করা কোনো সফটওয়্যার এর অংশ।

Test Condition:

An item or an event : কোনো ফাংশন, ট্রাঞ্জেকশন বা সিস্টেমের কোনো এলিমেন্ট

Test Criteria :

টেস্ট অবজেক্ট কে টেস্ট নিশ্চিত করতে হবে।

Black Box Technique

টেস্টার টেস্ট অবজেক্ট কে একটি ব্ল্যাক বক্স হিসেবে দেখে। টেস্ট অবজেক্টের ভিতরের গঠন টেস্টারের কাছে অপ্রাসঙ্গিক বা অজানা। অবজেক্টর ইনপুট-আউটপুট এর কার্যক্রম দেখে থাকে।

ব্ল্যাক বক্স টেস্টিং কে **functional** বা **specification oriented** টেস্টিং ও বলা হয়।

White Box Technique

টেস্টার প্রোগ্রামের ভিতরের গঠন সম্পর্কে ও কোড জানে। যেমনঃ **Component hierarchy, control flow, data flow.** টেস্ট কেস ঠিক করা হয় প্রোগ্রামের ভিতরের কোড বা গঠন দেখে।

হোয়াইট বক্স টেস্টিং কে **structure based** বা **control flow based** টেস্টিং ও বলা হয়।

Categories of Test Design Methods

স্পেসিফিকেশন ভিত্তিক পদ্ধতি - **Black Box**

গঠন ভিত্তিক পদ্ধতি - **White Box**

অভিজ্ঞতা ভিত্তিক পদ্ধতি - **Black Box**

Equivalence Class (EC) Partitioning

সংজ্ঞায়িত মানগুলোকে **Equivalence class** এর গ্রুপে ভাগ করে দেয়া হয়। যার জন্য নিম্নের নিয়মগুলো মানা হয়ে থাকে -

- যেসকল মানের জন্য প্রোগ্রামের একটি সাধারন আচরন করে সেসকল মানকে একটি **Equivalence class** এ গ্রুপ করা হয়।
- **Equivalence class** এর মধ্যে কোনো ওভারল্যাপ বা গ্যাপ থাকবে না।
- **Equivalence class** এ একটি রেঞ্জের মান যেমনঃ 0<x<10 বা একটি মান যেমনঃ x = 'yes'এমনও থাকতে পারে।

ভ্যালিড EC

ইনভ্যালিড EC

Age limit = Between 25 to 60
25<=Age<=60

Invalid	Valid	Invalid
1, 2, ..., 24	25,....., 60	61, 62, ...

Equivalence class গুলো ভ্যালিড আর ইনভ্যালিড স্টাএর জন্য ঠিক করা হয়।

x এর মান যদি $0 \leq x \leq 100$ দ্বারা প্রকাশ করা হয় তাহলে Equivalence ক্লাস গুলো হবে –

- $x<0$ (invalid input values)
- $0 \leq x \leq 100$ (valid inputs)
- $x>100$ (invalid input values)

এছাড়াও ইনভ্যালিড Equivalence class সংজ্ঞায়িত করা যায় যাতে নিম্নোক্ত জিনিসে সীমাবদ্ধ না থাকলে এসব অন্তর্ভুক্ত থাকতে পারে –

- Non numerical inputs
- Numbers too big or too small
- Non supported format for numbers

<0	0-100	>100

Equivalence Class Partitioning - Example

Problem:

A program expected a percentage value according to the following requirements:

- Only integer values are allowed
- 0 is the valid lower boundary of the range
- 100 is the valid upper boundary of the range

Solution:

- Valid are all numbers from 0 to 100,
- Invalid are all negative numbers,
  - all numbers greater than 100,
  - all decimal numbers and
  - all non numerical values (e.g. "rashed")

- One valid equivalence class: $0 \leq x \leq 100$
- 1st invalid equivalence class: $x < 0$
- 2nd invalid equivalence class: $0 > 100$
- 3rd invalid equivalence class: x= no integer
- 4th invalid equivalence class: x= not numeric (e.g. "abc")

Equivalence Class Partitioning - Example

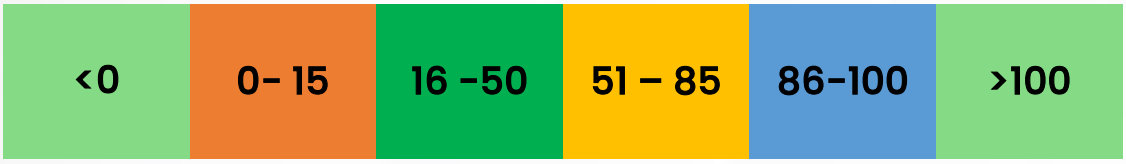
Additional Requirement:

The percentage value will now be displayed in a bar chart. The following additional requirements apply (both values included):

- values between 0 and 15 : Orange bar,
- values between 16 and 50: Green bar,
- values between 51 and 85: Yellow bar,
- values between 86 and 100: Blue bar,

Additional Solution for valid EC:

- Now there are four instead of one valid equivalence classes:
  - 1st valid equivalence class: $0 \leq x \leq 15$
  - 2nd valid equivalence class: $16 \leq x \leq 50$
  - 3rd valid equivalence class: $51 \leq x \leq 85$
  - 4th valid equivalence class: $86 \leq x \leq 100$



Ostad

EC Partitioning - Picking Representatives

Variable	Status	EC	Representative
Percentage Value	Valid	EC 1: 0<= X <=15	+10
		EC 2: 16<= X <=50	+20
		EC 3: 51<= X <=85	+80
		EC 4: 86<= X <=100	+90
	Invalid	EC 5: X<0	-10
		EC 6: X>100	+200
		EC 7: X not integer	1.5
		EC 8: X non number	fred

Equivalence Class Partitioning - Example 2 /1

Analyzing the specification

A piece of code computes the price of a product, based on its value, a discount in % and shipping costs (BDT 19, 29, 49 depending on shipping mode)

Variable	Equivalence class	Status	Representatives
Values of goods	EC_{11}: x >= 0	Valid	1000
	EC_{12}: x < 0	invalid	-1000
	EC_{13}: x non-numerical value	Invalid	Rashed
Discount	EC_{21}: 0% ≤ x ≤ 100%	valid	10%
	EC_{22}: x < 0%	Invalid	-10%
	EC_{23}: > 100	Invalid	200%
	EC_{24}: x non numeric value	Invalid	Karim
Shipping costs	EC_{31}: x = 19	valid	19
	EC_{32}: x = 29	valid	29
	EC_{33}: x = 49	valid	49
	EC_{34}: x ≠ {19, 29, 49}	Invalid	30
	EC_{35}: x non numeric value	invalid	Student

Ostad

Equivalence Class Partitioning - Example 2 /2

Test cases for valid EC:

Valid equivalence classes provide the following combinations or test cases: T1, T2 and T3

Variable	Equivalence class	Status	Representatives	T1	T2	T3
Values of goods	EC_{11}: x >= 0	Valid	1000	*	*	*
	EC_{12}: x < 0	invalid	-1000			
	EC_{13}: x non-numerical value	Invalid	Rashed			
Discount	EC_{21}: 0% ≤ x ≤ 100%	valid	10%	*	*	*
	EC_{22}: x < 0%	Invalid	-10%			
	EC_{23}: > 100	Invalid	200%			
	EC_{24}: x non numeric value	Invalid	Karim			
Shipping costs	EC_{31}: x = 19	valid	19	*		
	EC_{32}: x = 29	valid	29		*	
	EC_{33}: x = 49	valid	49			*
	EC_{34}: x ≠ {19, 29, 49}	Invalid	30			
	EC_{35}: x non numeric value	invalid	Student			

Equivalence Class Partitioning - Example 2 /3

Test cases for invalid EC:

The following test cases were created using the invalid EC, each in combination with valid ECs of other elements:

Variable	Equivalence class	Status	Representatives	T4	T5	T6	T7	T8	T9	T10
Values of goods	EC_{11}: x >= 0	Valid	1000			*	*	*	*	*
	EC_{12}: x < 0	invalid	-1000	*						
	EC_{13}: x non-numerical value	Invalid	Rashed		*					
Discount	EC_{21}: 0% ≤ x ≤ 100%	valid	10%	*	*				*	*
	EC_{22}: x < 0%	Invalid	-10%			*				
	EC_{23}: > 100	Invalid	200%				*			
	EC_{24}: x non numeric value	Invalid	Karim					*		
Shipping costs	EC_{31}: x = 19	valid	19	*	*	*	*	*		
	EC_{32}: x = 29	valid	29							
	EC_{33}: x = 49	valid	49							
	EC_{34}: x ≠ {19, 29, 49}	Invalid	30						*	
	EC_{35}: x non numeric value	invalid	Student							*

Equivalence Class Partitioning - Example 2 /4

Values of goods	EC_{11}: x >= 0	Valid	1000	*	*	*			*	*	*	*	*
	EC_{12}: x < 0	invalid	-1000				*						
	EC_{13}: x non-numerical value	Invalid	Rashed					*					
Discount	EC_{21}: 0% ≤ x ≤ 100%	valid	10%	*	*	*	*	*				*	*
	EC_{22}: x < 0%	Invalid	-10%						*				
	EC_{23}: > 100	Invalid	200%							*			
	EC_{24}: x non numeric value	Invalid	Karim								*		
Shipping costs	EC_{31}: x = 19	valid	19	*			*	*	*	*	*		
	EC_{32}: x = 29	valid	29		*								
	EC_{33}: x = 49	valid	49			*							
	EC_{34}: x ≠ {19, 29, 49}	Invalid	30									*	
	EC_{35}: x non numeric value	invalid	Student										*

Equivalence Class Partitioning - Coverage

Equivalence class coverage can be used as exit criteria to end testing activities

$$\text{EC Coverage} = \frac{\text{Number of EC tested}}{\text{Number of EC defined}} \times 100\%$$