Software Quality Assurance & Testing

TOPIC-5: TEST DESIGN TECHNIQUES ,

Prepared By:
Md. Minhaj Hosen, Lecturer, DIU

Outlines

1. Test Development Process
2. Test Design Techniques

1].Test Development Process

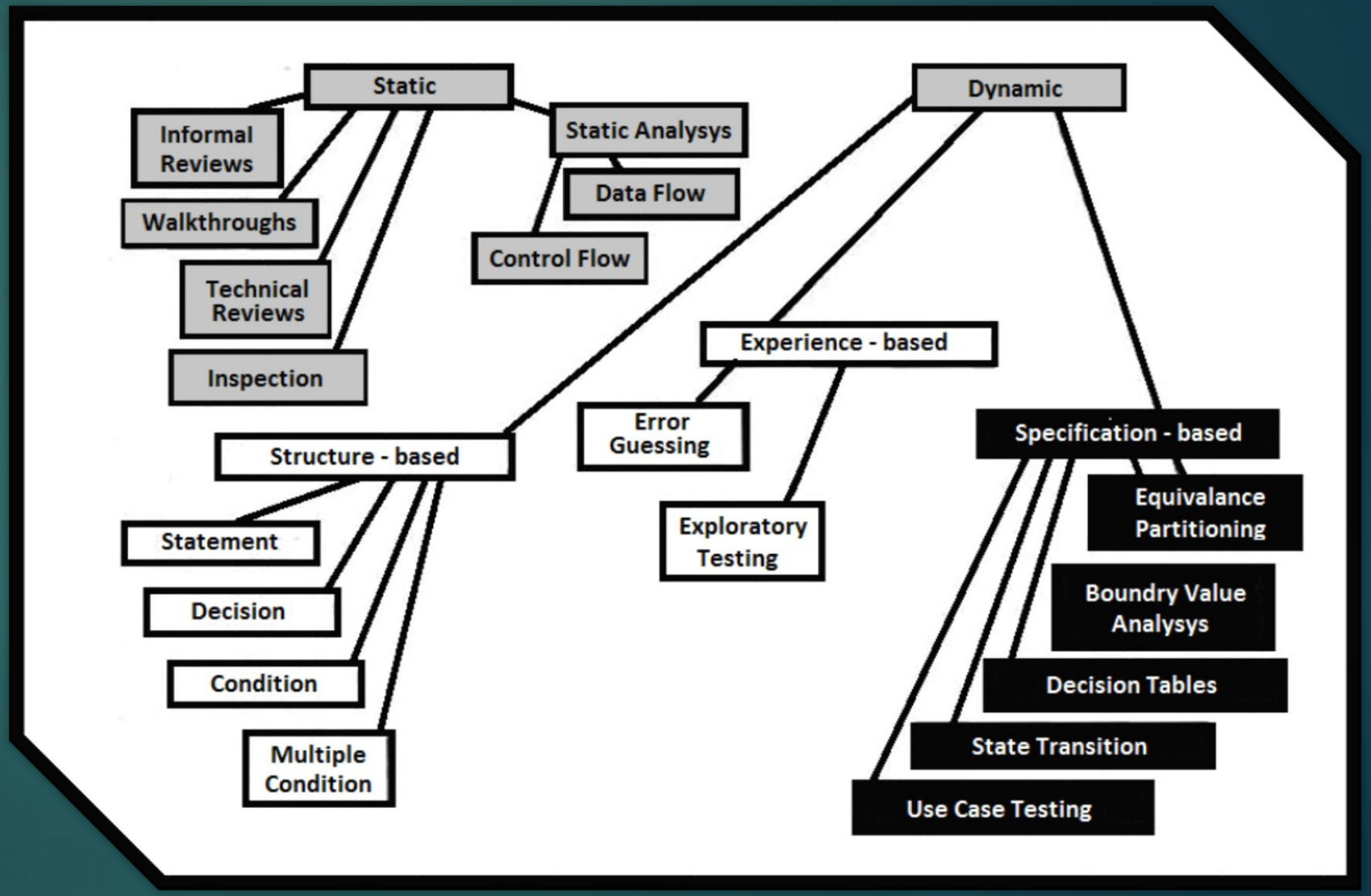
- **a) Test Analysis:** identifying test conditions
  - **Test Design Specification -** A document specifying the test conditions (coverage items) for a test item, the detailed test approach and associated high level test cases.
  - Template - Test design specification identifier, Features to be tested, Approach refinements, Test identifications, Feature pass/fail criteria
- **b) Test Design:** specifying test cases
  - **Test Case Specification -** A document specifying a set of test cases (objective, inputs, test actions, executed results, and execution preconditions) for a test item.
  - Template - Test case specification identifier, Test items, Input Specifications, Output specifications, Environmental needs, Special procedural requirements, Inter case dependencies

1].Test Development Process

- **c) Test Implementation:** specifying test procedure or scripts
  - **Test Procedure Specification -** A document specifying a sequence of actions for the execution of a test, also known as a test script or manual test script.
  - Template - Test case procedure specification identifier, Purpose, Special requirements, Procedure steps,
  - Include steps to be taken for a test including set up, logging, environment and measurement.

2. Test Design Techniques

- **Static**
- **Dynamic**



Categories of test design methods:

Specification based methods:
(Black Box)

Structure based methods:
(White Box)

Experienced based methods:
(Black Box)

2.1 Specification-based/Black-box technique

- A procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.
- Also know as Black Box Testing.
- Specification-based/Black-box technique:
  a) **Equivalence partitioning**
  b) **Boundary value analysis**
  c) **Decision table testing**
  d) **State transition testing**
  e) **Use case testing**

2.2 Structure-based/White-box technique

- A procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.
- Also know as White box technique
- Structure-based/White-box techniques:
  a) **Statement (Node)coverage**
  b) **Decision (Brunch) coverage**
  c) **Condition coverage**
  d) **Multiple condition coverage**

2.3 Experience-based techniques

- They depend on an individual's personal view rather than on a documented record of what the system should do.
- Experience based techniques - Error guessing, Exploratory testing
- Experience-based techniques:
  a) **Error guessing**
  b) **Exploratory testing**

Example: EC

Equivalence class (EC) partitioning

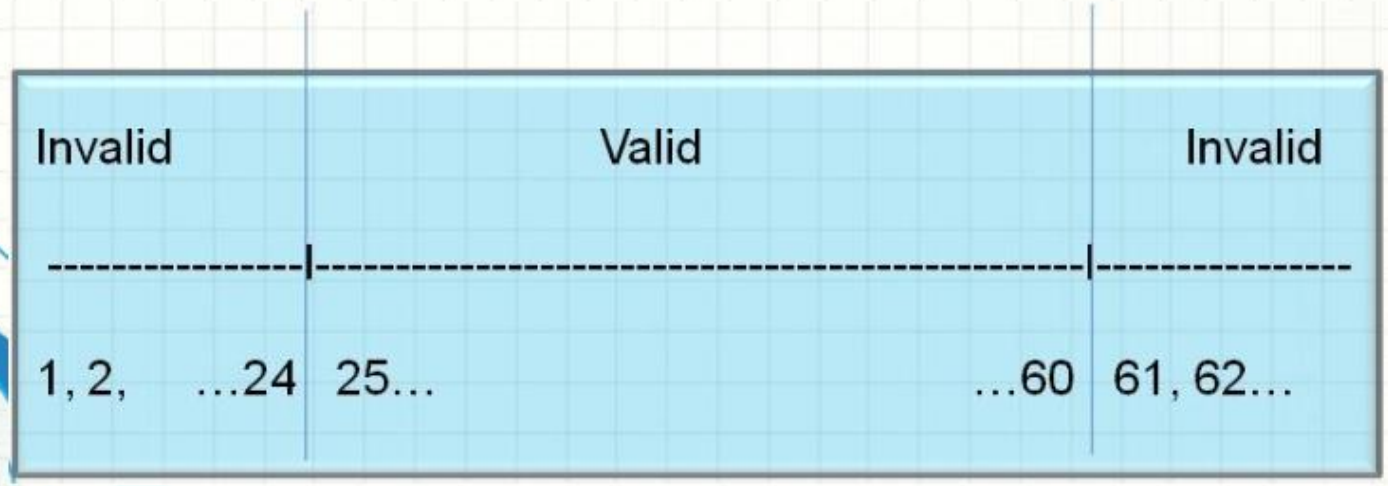
- The range of defined values is grouped into **equivalence classes**, for which the following **rules** apply:
  - All values, for which a **common behavior** of the program is **expected**, are grouped together in one equivalence class
  - Equivalence class may **not overlap** and may **not contain any gaps**
  - Equivalence class may contain a **range** of values (e.g. 0<x<10) or a **single** value (e.g. x="Yes")

-invalid EC

- valid EC:

Age limit = between 25 and 60

25<=Age=<60

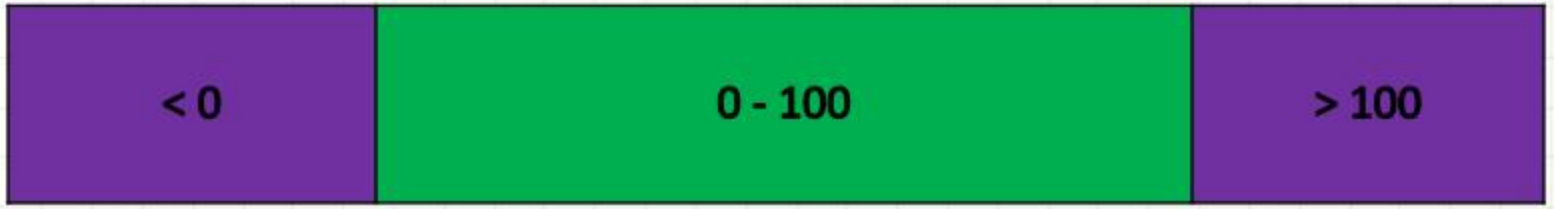


Equivalence class partitioning – example

- Equivalence classes are chosen for valid and invalid inputs
  - if a value x is defined as **0≤ x ≤ 100**, then when we can initially identify three equivalence classes:

1.	**x<0**	(invalid input values)
2.	**0 ≤ x≤100**	(valid input)
3.	**x> 100**	

- Further invalid EC can be defined, containing, but not limited to:
  non-numerical inputs,
  numbers to big or to small,
  non-supported format for numbers



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. “fred”)

- 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: $15 \leq x \leq 50$
  - 3rd valid equivalence class: $51 \leq x \leq 85$
  - 4th valid equivalence class: $86 \leq x \leq 100$

EC Partitioning – Picking Representatives

Variable	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: 85<= X <=100	+90
Percentage Value (Invalid)	EC 5: X<0	-10
	EC 6: X>100	+200
	EC 7: X not integer	1.5
	EC 8: X non number	fred

Boundary Value Analysis:

Boundary analysis /1

- Boundary analysis extends equivalence class partitioning by introducing a **rule** for the **choice of representatives**
- The **edge values** of the **equivalence class** are to be tested **intensively**
- Why put more attention to the edges?
    - Often, the boundaries of values ranges are **not well defined** or lead to different interpretations
    - Checking if the boundaries were **programmed correctly**

- Please note:

  Experience shows that **error** occur **very frequently** on the **boundaries** of value ranges!

Defining boundary values

- If the **EC** is defined as **single numerical value**, for example **x= 5**, the neighboring values will be used as well
  - the representatives (of the class and its neighboring values) are: **4,5 and 6**

Boundary analysis example 3a

- **Example 3a:**
  - value range for a discount in % : $0{,}00 \leq x \leq 100{,}00$
  - **Definition of EC**

    3 classes:

    1. EC: $x < 0$
    2. EC: $0{,}00 \leq x \leq 100{,}00$
    3. EC: $x > 100$
  - **Boundary analysis**

    extends the representatives to:

    2.EC: -0.01; 0.00; 0.01; 99.99; 100.00; 100.01

  - **Please note:**

    Instead of one representative for the valid EC, there are now six representatives (four valid and two invalid)

Boundary analysis example 3b

- **Basic scheme:** choose **three values** to be tested
  the exact boundary and the two neighboring values (within and outside the EC)

- **Alternative point of view**: since the boundary value belongs to the EC, only **two values** are needed for testing: **one within and one outside the EC**

- Example 3b:

- value range for a discount in %: $0.00 \leq x \leq 100.00$
- **valid EC**: $0.00 \leq x \leq 100.00$
- **boundary analysis**

  additional representatives are: **-0.01; 0.00; 100.00; 100.01**

  0.01 – same behavior as 0.00

  99.99 - same behavior as 100.00

- A programming error caused by a wrong comparison operator will be found with the two boundary values

Decision Table

Decision table testing

- Equivalence class partitioning and boundary analysis deal with **isolated input** conditions.
- However, an input condition may have an effect only in **combination** with other input conditions.
- All previously described methods do not take into account the effects of **dependencies and combinations.**
- Using the **full set of combinations** of all input equivalence classes usually leads to a very **high number of test cases** (test case explosion).
- With the help of **cause-and-effect graphs** and the decision tables derived from them, **the amount of possible combinations** can be **reduced** systematically to a subset.

State Transition

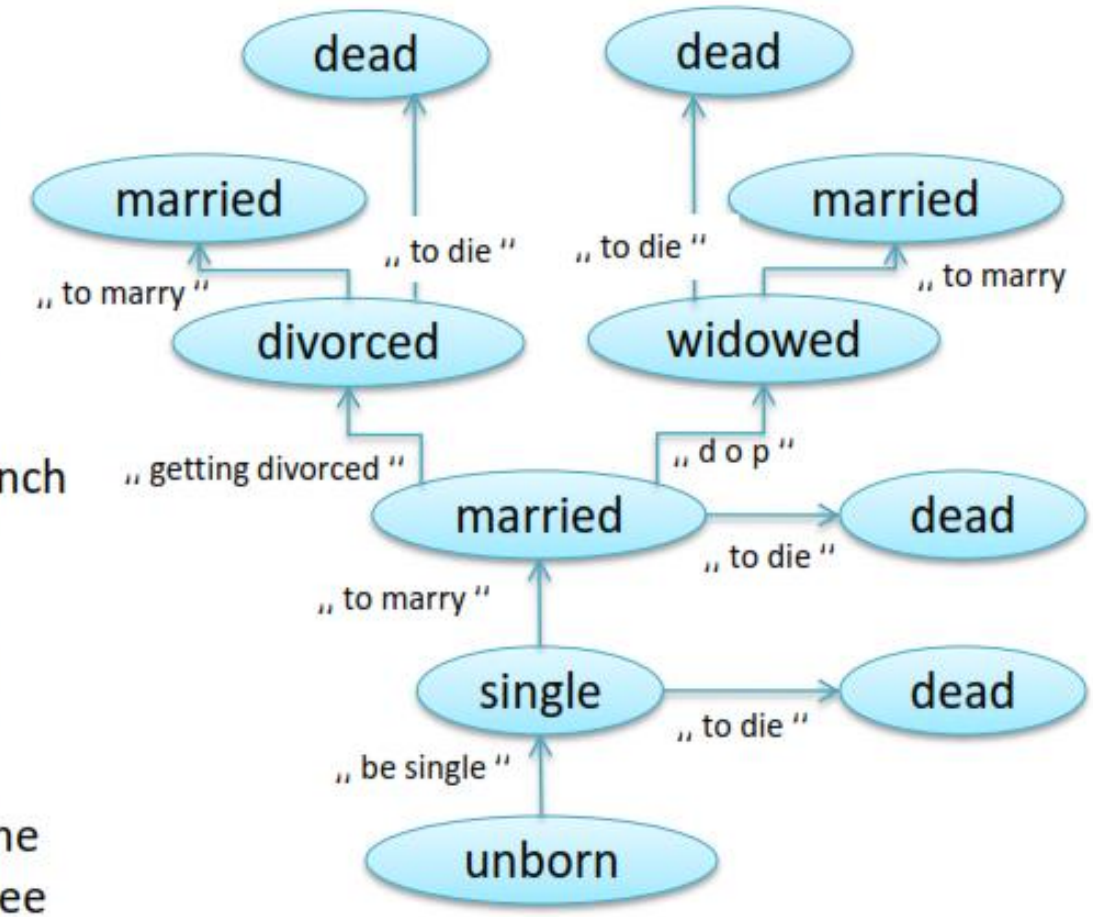
State transition testing

- Many methods only take into account the system behavior in terms of **input data** and **output data**
- Different states that a test object might take on are not taken into account
  - for example, results of actions that **happened in the past** actions, that caused the test object to be in a certain internal state

- The different states that a test object can take on are modeled using **state transition diagrams**
- **State transition analysis** is used to define state transition based test cases

State transition testing

- To determine test cases using a state transition diagram a **transition tree** is constructed:
- The initial state is the **root** of the tree
- For every state that may be reached from the initial state, A node is created which is connected to the root by a branch
- This operation is repeated and comes to an end if
  - the state of the node is an end state (a leaf of the tree) Or
  - the same node with the same state is already part of the tree

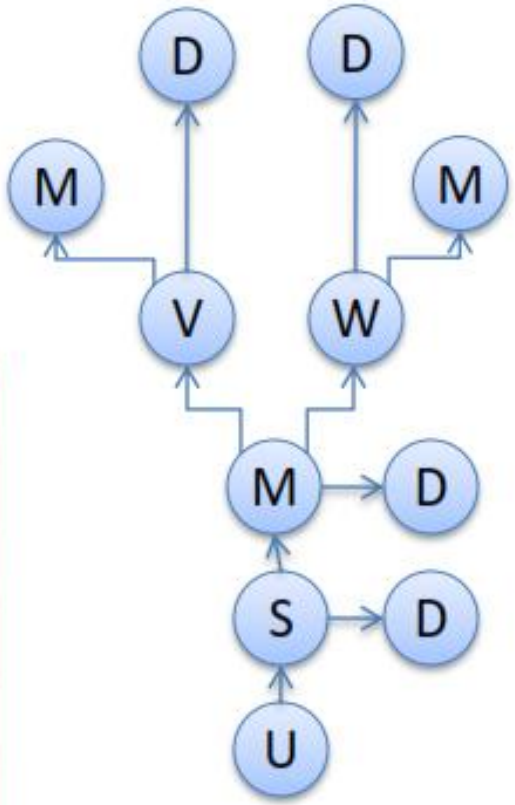


Event. „ d o p " : death of partner

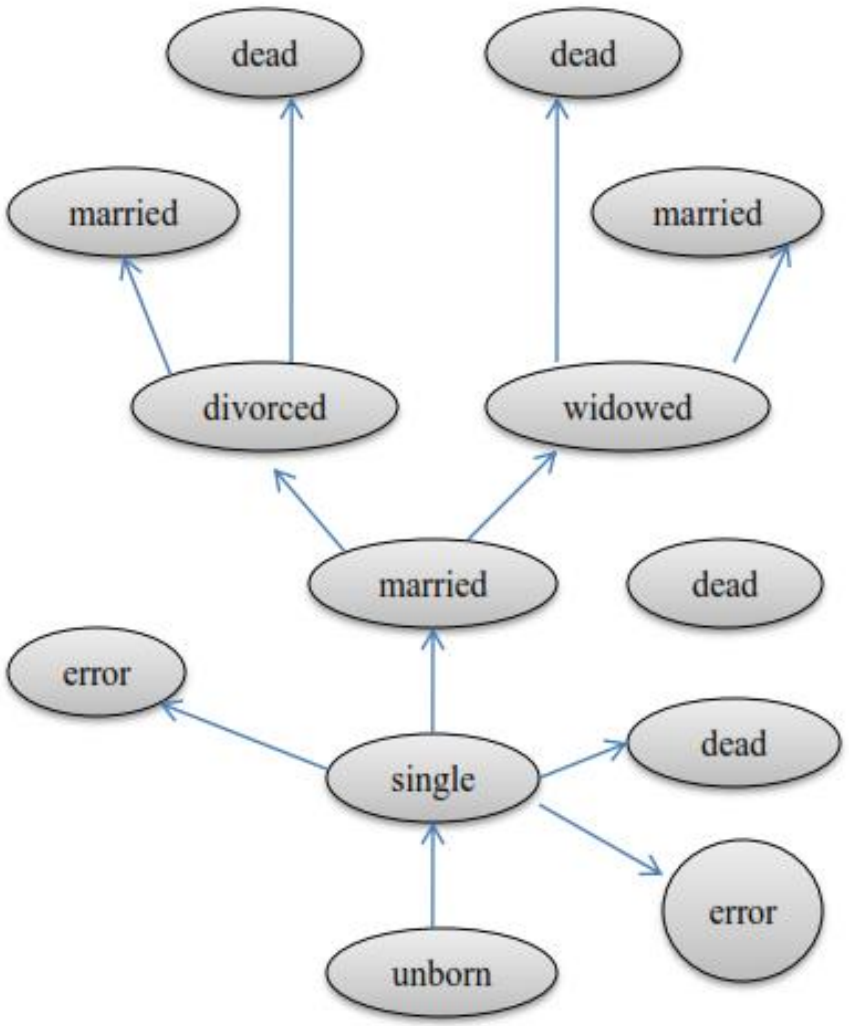
State transition testing

Every path from the root to a leaf then represents a test case for state transition testing

The state transition tree for this example leads to the following six test cases:



state1	state2	state3	state4	State5	End State
unborn	single	Dead			Dead
Unborn	Single	married	dead		Dead
Unborn	Single	married	widowed	dead	Dead
Unborn	single	married	widowed	Married	Married
Unborn	Single	married	divorced	Dead	Dead
Unborn	Single	married	divorced	married	married



-The transition tree of our example may now be extended using invalid transition (negative test cases robustness testing)

- Example: two possible **invalid** transitions- there are more
- **impossible** transitions between states can not be tested

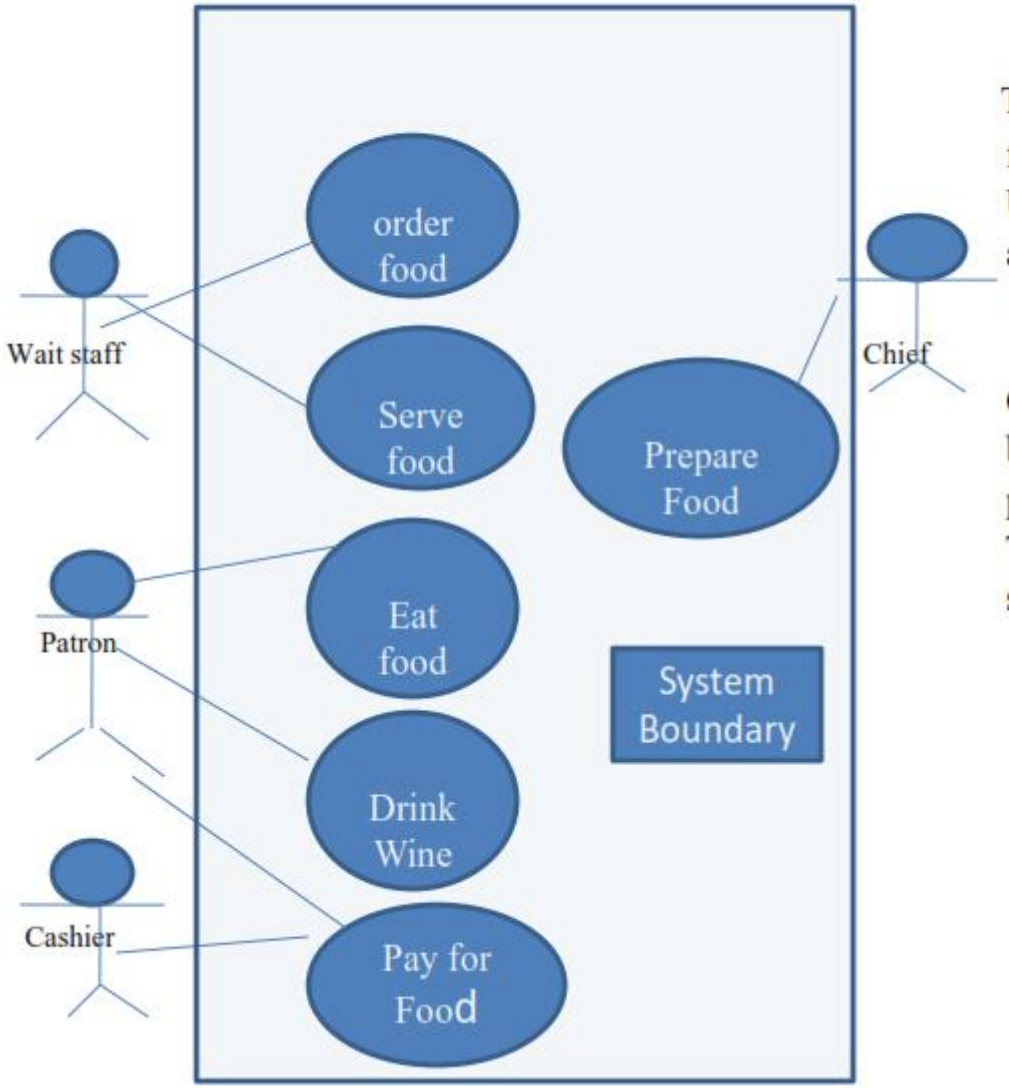
Use case based testing

- **Test cases** are derived directly from the **use cases** of the test object
  - The test object is seen as a **system** reacting with **actors**
  - A use case describes the **interaction** of **all involved actors** leading to an **end result** of the system
  - every use case has **pre-conditions** that have to be met the in order to execute the use case (the test case) successfully
  - Every use case have **post-conditions** describing the system after execution of the use case (the test case)
- Use cases are **elements** of the unified modeling language **UML***
  - **Use case diagram** are one of 13 different types of diagrams used by UML
  - A use case diagram is a diagram is a diagram describing a **behavior**, it does not describe the sequence events
  - It shows the system reaction from the **viewpoint of user**

UML is a non-proprietory specification language for object modeling

Use case based testing

- Example of a simple use case diagram (source: Wikipedia)



The diagram on the left describes the functionality of a simple restaurant system
Use case are represented by **ovals** and the actors are represented by **stick** figures.
The patron actor can eat food. Pay for food or drink wine
Only the chief actor can prepare food. Note that both the patron and the cashier are involved in the pay for food use case
The box defines the boundaries of the restaurant system being modeled, the actors are not

Use case based testing

- Every use case describes a certain task (user-system-interaction)
- **Use case** descriptions include, but are not limited to:
    - Pre conditions
    - Expected results/system behavior
    - Post: conditions
- These descriptive elements are also used to define the corresponding **test cases**
- Every **use case** may be used as the basis for a **test case**
- **Every alternative** within the diagram corresponds to a **separate test case**
- Typically, information provided with a use case has **not enough detail** to define the **test case** directly, Additional data is needed (input data, expected results) to make up a test case