Software Quality Assurance and Testing Lecture - 04

ABDUS SATTER
LECTURER
INSTITUTE OF INFORMATION TECHNOLOGY
UNIVERSITY OF DHAKA

Dynamic Testing

BLACK BOX TESTING TECHNIQUES

Equivalence Class Testing

- We know that the input domain for testing is too large to test every input. So we can divide or partition the input domain based on a common feature or a class of data. Equivalence partitioning is a method for deriving test cases where in classes of input conditions called equivalence classes are identified such that each member of the class causes the same kind of processing and output to occur.

Equivalence Class Testing

- Equivalence partitioning method for designing test cases has the following goals:
  - **Completeness:** Without executing all the test cases, we strive to touch the completeness of testing domain.
  - **Non-redundancy:** When the test cases are executed having inputs from the same class, then there is redundancy in executing the test cases. Time and resources are wasted in executing these redundant test cases, as they explore the same type of bug. Thus, the goal of equivalence partitioning method is to reduce these redundant test cases.

Identification Of Equivalent Classes

- Two types of classes can always be identified :
  - **Valid equivalence classes:** These classes consider valid inputs to the program.
  - **Invalid equivalence classes:** One must not be restricted to valid inputs only. We should also consider invalid inputs that will generate error conditions or unexpected behavior of the program

Guidelines for Forming Equivalence Classes

- If there is no reason to believe that the entire range of an input will be treated in the same manner, then the range should be split into two or more equivalence classes.
- If a program handles each valid input differently, then define one valid equivalence class per valid input.
- Boundary value analysis can help in identifying the classes. For example, for an input condition, say $0 \leq a \leq 100$, one valid equivalent class can be formed from the valid range of a. And with BVA, two invalid classes that cross the minimum and maximum values can be identified, i.e. $a < 0$ and $a > 100$.

Guidelines for Forming Equivalence Classes

- If an input variable can identify more than one category, then for each category, we can make equivalent classes. For example, if the input is a character, then it can be an alphabet, a number, or a special character. So we can make three valid classes for this input and one invalid class.
- If an input condition specifies a ‘must be’ situation (e.g., ‘first character of the identifier must be a letter’), identify a valid equivalence class (it is a letter) and an invalid equivalence class (it is not a letter).

Guidelines for Forming Equivalence Classes

- Equivalence classes can be of the output desired in the program. For an output equivalence class, the goal is to generate test cases such that the output for that test case lies in the output equivalence class. Determining test cases for output classes may be more difficult, but output classes have been found to reveal errors that are not revealed by just considering the input classes.

Guidelines for Forming Equivalence Classes

- Look for membership of an input condition in a set or group and identify valid (within the set) and invalid (outside the set) classes. For example, if the requirements state that a valid province code is ON, QU, and NB, then identify: the valid class (code is one of ON, QU, NB) and the invalid class (code is not one of ON, QU, NB).

Guidelines for Forming Equivalence Classes

- If the requirements state that a particular input item match a set of values and each case will be dealt with differently, identify a valid equivalence class for each element and only one invalid class for values outside the set. For example, if a discount code must be input as P for a preferred customer, R for a standard reduced rate, or N for none, and if each case is treated differently, identify: the valid class code = P, the valid class code = R, the valid class code = N, the invalid class code is not one of P, R, N.

Guidelines for Forming Equivalence Classes

- If an element of an equivalence class will be handled differently than the others, divide the equivalence class to create an equivalence class with only these elements and an equivalence class with none of these elements. For example, a bank account balance may be from 0 to Rs 10 lakh and balances of Rs 1,000 or more are not subject to service charges. Identify: the valid class: (0 ≤balance <Rs 1,000), i.e. balance is between 0 and Rs 1,000 – not including Rs 1,000; the valid class: (Rs 1,000 ≤ balance ≤Rs 10 lakh, i.e. balance is between Rs 1,000 and Rs 10 lakh inclusive the invalid class: (balance <0) the invalid class: (balance >Rs 10 lakh).

Example

- A program reads three numbers, A, B, and C, with a range [1, 50] and prints the largest number. Design test cases for this program using equivalence class testing technique.

$$I_1 = \{<A,B,C> : 1 \le A \le 50\}$$
$$I_2 = \{<A,B,C> : 1 \le B \le 50\}$$
$$I_3 = \{<A,B,C> : 1 \le C \le 50\}$$
$$I_4 = \{<A,B,C> : A < 1\}$$
$$I_5 = \{<A,B,C> : A > 50\}$$
$$I_6 = \{<A,B,C> : B < 1\}$$
$$I_7 = \{<A,B,C> : B > 50\}$$
$$I_8 = \{<A,B,C> : C < 1\}$$
$$I_9 = \{<A,B,C> : C > 50\}$$

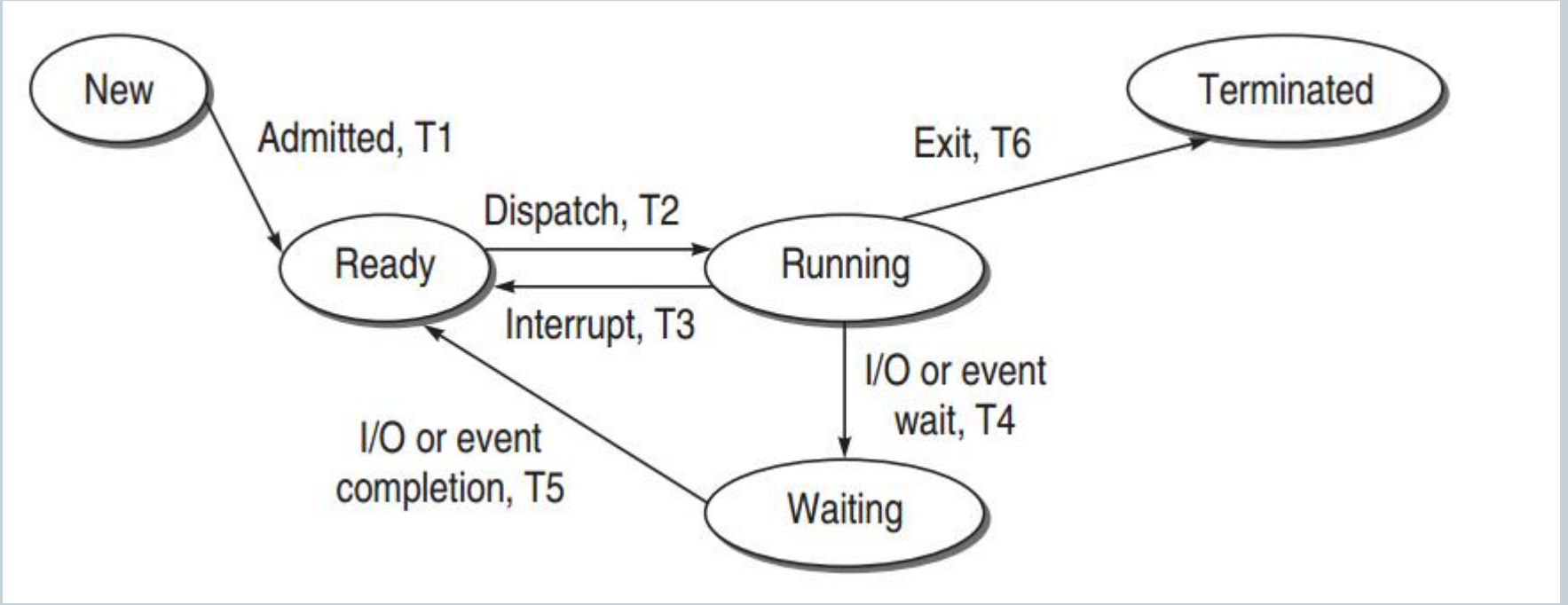
Example

Test case ID	A	B	C	Expected result	Classes covered by the test case
1	13	25	36	C is greatest	I_1, I_2, I_3
2	0	13	45	Invalid input	I_4
3	51	34	17	Invalid input	I_5
4	29	0	18	Invalid input	I_6
5	36	53	32	Invalid input	I_7
6	27	42	0	Invalid input	I_8
7	33	21	51	Invalid input	I_9

State Table-based Testing

- A system or its components may have a number of states depending on its input and time. For example, a task in an operating system can have the following states:
  - New State: When a task is newly created.
  - Ready: When the task is waiting in the ready queue for its turn.
  - Running: When instructions of the task are being executed by CPU.
  - Waiting: When the task is waiting for an I/O event or reception of a signal.
  - Terminated: The task has finished execution.

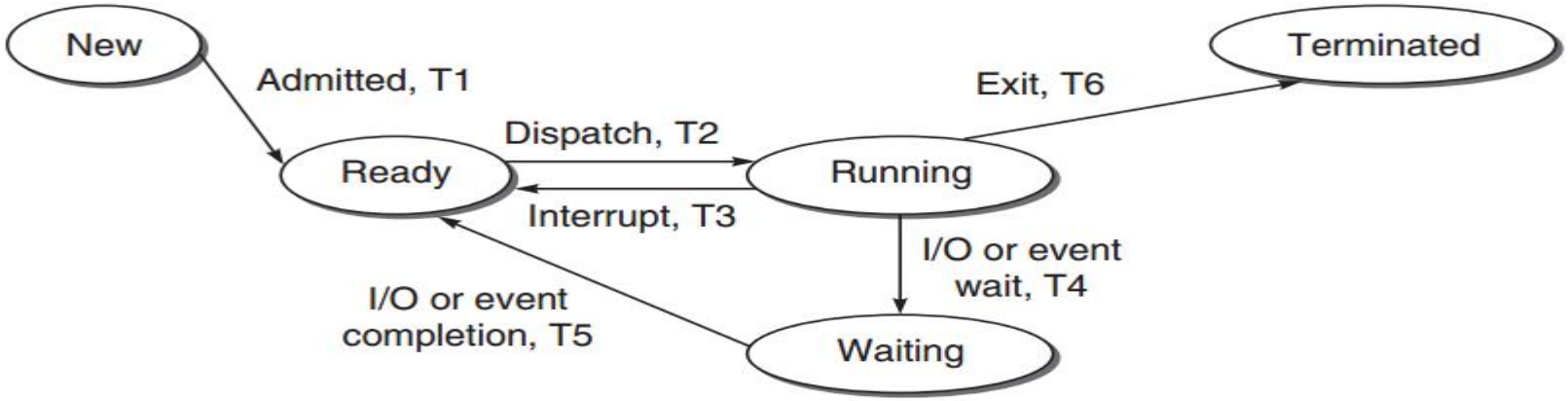
State Graph


New
Admitted, T1
Terminated
Exit, T6
Dispatch, T2
Ready
Running
Interrupt, T3
I/O or event
wait, T4
I/O or event
completion, T5
Waiting

The Resulting Output From A State

- T0 = Task is in new state and waiting for admission to ready queue
- T1 = A new task admitted to ready queue
- T2 = A ready task has started running
- T3 = Running task has been interrupted
- T4 = Running task is waiting for I/O or event
- T5 = Wait period of waiting task is over
- T6 = Task has completed execution

State Table



State\Input Event	Admit	Dispatch	Interrupt	I/O or Event Wait	I/O or Event Wait Over	Exit
New	**Ready/ T1**	New / T0	New / T0	New / T0	New / T0	New / T0
Ready	Ready/ T1	**Running/ T2**	Ready / T1	Ready / T1	Ready / T1	Ready / T1
Running	Running/T2	Running/ T2	**Ready / T3**	**Waiting/ T4**	Running/ T2	**Terminated/T6**
Waiting	Waiting/T4	Waiting / T4	Waiting/T4	Waiting / T4	**Ready / T5**	Waiting / T4

State Table-based Testing

- Identify the states
- Prepare state transition diagram after understanding transitions between states
- Convert the state graph into the state table as discussed earlier
- Analyze the state table for its completeness
- Create the corresponding test cases from the state table

State Table-based Testing

Test Case ID	Test Source	Input		Expected Results	
		Current State	Event	Output	Next State
TC1	Cell 1	New	Admit	T1	Ready
TC2	Cell 2	New	Dispatch	T0	New
TC3	Cell 3	New	Interrupt	T0	New
TC4	Cell 4	New	I/O wait	T0	New
TC5	Cell 5	New	I/O wait over	T0	New
TC6	Cell 6	New	exit	T0	New
TC7	Cell 7	Ready	Admit	T1	Ready
TC8	Cell 8	Ready	Dispatch	T2	Running
TC9	Cell 9	Ready	Interrupt	T1	Ready
TC10	Cell 10	Ready	I/O wait	T1	Ready
TC11	Cell 11	Ready	I/O wait	T1	Ready
TC12	Cell 12	Ready	Exit	T1	Ready
TC13	Cell 13	Running	Admit	T2	Running
TC14	Cell 14	Running	Dispatch	T2	Running
TC15	Cell 15	Running	Interrupt	T3	Ready
TC16	Cell 16	Running	I/O wait	T4	Waiting
TC17	Cell 17	Running	I/O wait over	T2	Running
TC18	Cell 18	Running	Exit	T6	Terminated
TC19	Cell 19	Waiting	Admit	T4	Waiting
TC20	Cell 20	Waiting	Dispatch	T4	Waiting
TC21	Cell 21	Waiting	Interrupt	T4	Waiting
TC22	Cell 22	Waiting	I/O wait	T4	Waiting
TC23	Cell 23	Waiting	I/O wait over	T5	Ready
TC24	Cell 24	Waiting	Exit	T4	Waiting

Thank You

END OF CHAPTER