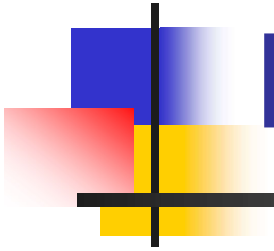
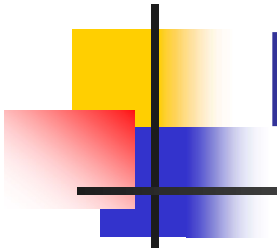


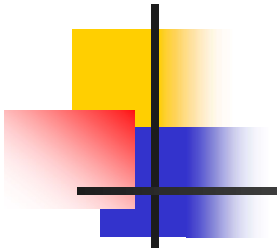
Black Box Testing



Role of Test cases

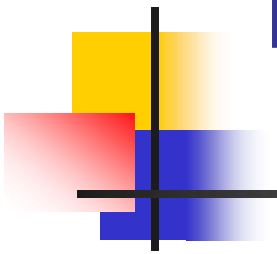
- Ideally would like the following for test cases
  - No failure implies "no defects" or "high quality"
  - If defects present, then some test case causes a failure
- Role of test cases is clearly very critical
- Only if test cases are "good", the confidence increases after testing

Test case design



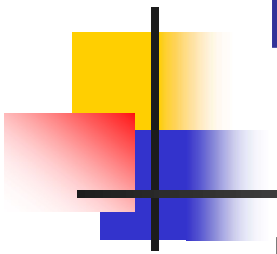
- During test planning, have to design a set of test cases that will detect defects present
- Some criteria needed to guide test case selection
- Two approaches to design test cases
  - functional or black box
  - structural or white box
- Both are complimentary

Black Box testing



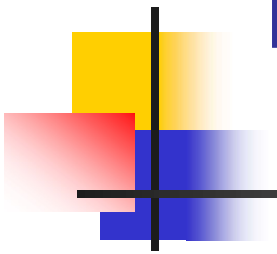
- Software tested to be treated as a block box
- Specification for the black box is given
- The expected behavior of the system is used to design test cases
- i.e test cases are determined solely from specification.
- Internal structure of code not used for test case design

Black box Testing…



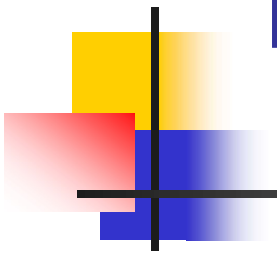
- Premise: Expected behavior is specified.
- Hence just test for specified expected behavior
- How it is implemented is not an issue.
- For modules,specification produced in design specify expected behavior
- For system testing, SRS specifies expected behavior

Black Box Testing…



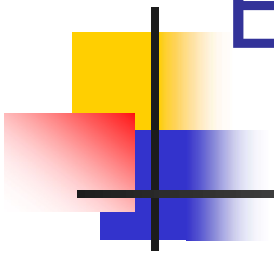
- Most thorough functional testing - exhaustive testing
  - Software is designed to work for an input space
  - Test the software with all elements in the input space
- Infeasible - too high a cost
- Need better method for selecting test cases
- Different approaches have been proposed

Equivalence Class partitioning



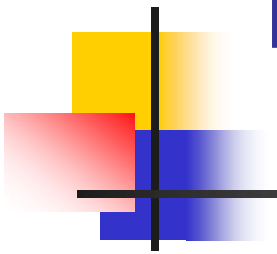
- Divide the input space into equivalent classes
- If the software works for a test case from a class then it is likely to work for all
- Can reduce the set of test cases if such equivalent classes can be identified
- Getting ideal equivalent classes is impossible
- Approximate it by identifying classes for which different behavior is specified

Equivalence class partitioning…



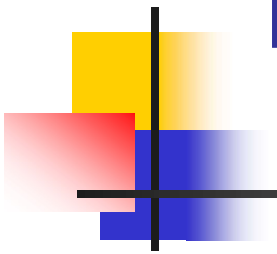
- Rationale: specification requires same behavior for elements in a class
- Software likely to be constructed such that it either fails for all or for none.
- E.g. if a function was not designed for negative numbers then it will fail for all the negative numbers
- For robustness, should form equivalent classes for invalid inputs also

Equivalent class partitioning..

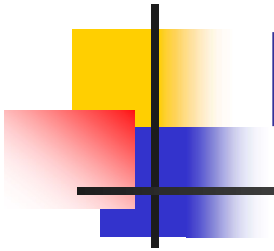


- Every condition specified as input is an equivalent class
- Define invalid equivalent classes also
- E.g. range 0< value<Max specified
  - one range is the valid class
  - input < 0 is an invalid class
  - input > max is an invalid class
- Whenever that entire range may not be treated uniformly - split into classes

Equivalent class partitioning..

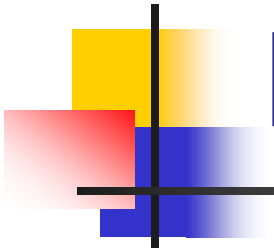


- Should consider eq. classes in outputs also and then give test cases for different classes
- E.g.: Compute rate of interest given loan amount, monthly installment, and number of months
  - Equivalent classes in output: + rate, rate = 0 ,-ve rate
  - Have test cases to get these outputs



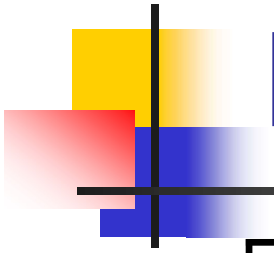
Equivalence class…

- Once eq classes selected for each of the inputs, test cases have to be selected
  - Select each test case covering as many valid eq classes as possible
  - Or, have a test case that covers at most one valid class for each input
  - Plus a separate test case for each invalid class



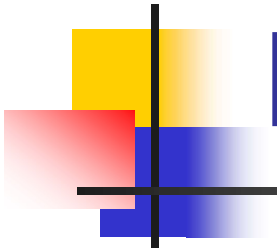
Example

- Consider a program that takes 2 inputs – a string s and an integer n
- Program determines n most frequent characters
- Tester believes that programmer may deal with diff types of chars separately
- A set of valid and invalid equivalence classes is given



Example..

Input	Valid Eq Class	Invalid Eq class
S	1: Contains numbers 2: Lower case letters 3: upper case letters 4: special chars 5: str len between 0-N(max)	1: non-ascii char 2: str len > N
N	6: Int in valid range	3: Int out of range



Example…

- Test cases (i.e. s , n) with first method
  - s : str of len < N with lower case, upper case, numbers, and special chars, and n=5
  - Plus test cases for each of the invalid eq classes
  - Total test cases: 1+3= 4
- With the second approach
  - A separate str for each type of char (i.e. a str of numbers, one of lower case, …) + invalid cases
  - Total test cases will be 5 + 2 = 7