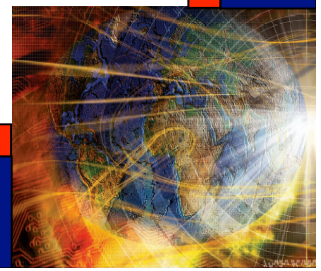
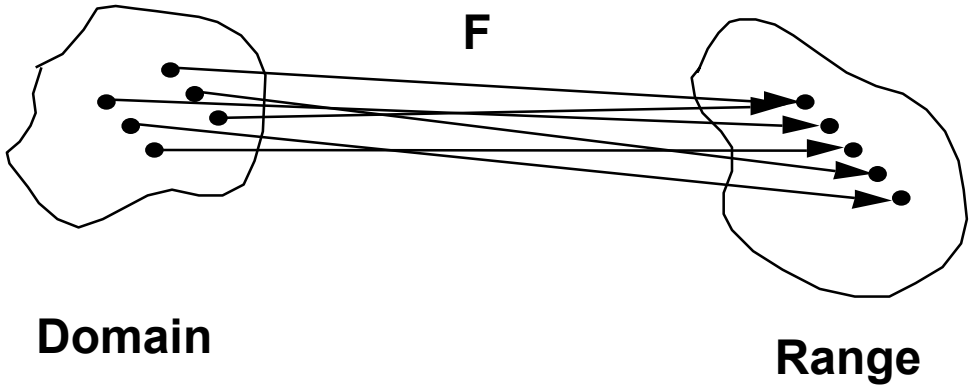
Chapter 6

Equivalence Class Testing



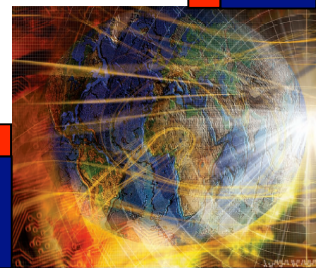
Equivalence Class Testing



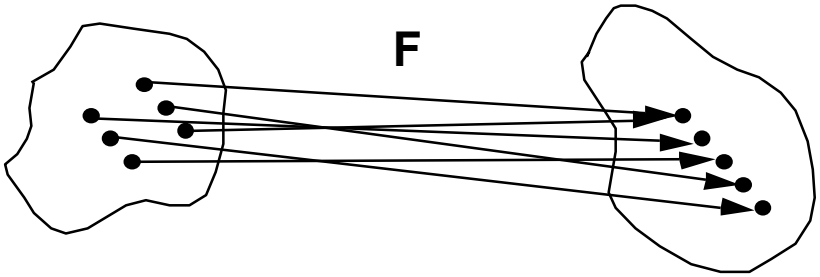
Equivalence class testing uses information about the functional mapping itself to identify test cases

Equivalence Relations

- Given a relation R defined on some set S, R is an equivalence relation if (and only if), for all, x, y, and z elements of S:
  - R is reflexive, i.e., xRx
  - R is symmetric, i.e., if xRy, then yRx
  - R is transitive, i.e., if xRy and yRz, then xRz
- An equivalence relation, R, induces a partition on the set S, where a partition is a set of subsets of S such that:
  - The intersection of any two subsets is empty, and
  - The union of all the subsets is the original set S
- Note that the intersection property assures no redundancy, and the union property assures no gaps.



Equivalence Partitioning

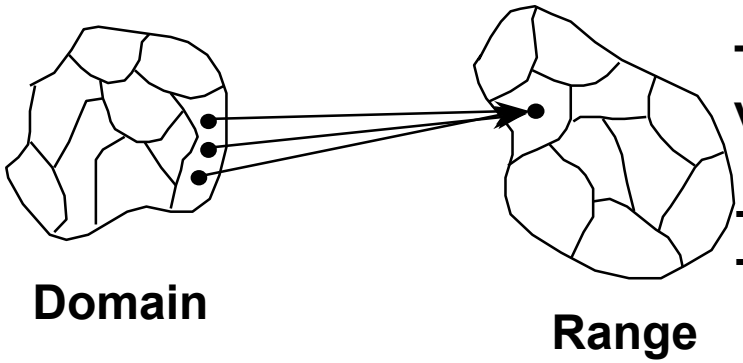


Define relation R as follows:

for x, y in domain, xRy iff F(x) = F(y).

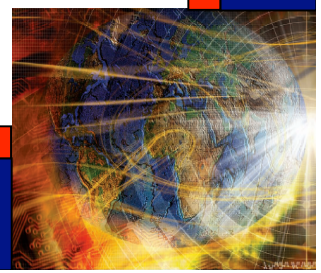
Facts:

1. R is an equivalence relation.
2. An equivalence relation induces a partition on a set.
3. Works best when F is many-to-one
4. (pre-image set)



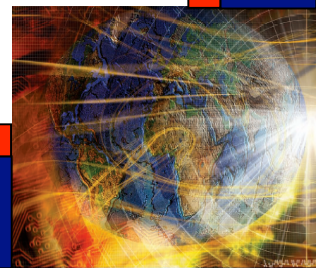
Test cases are formed by selecting one value from each equivalence class.

- reduces redundancy
- identifying the classes may be hard



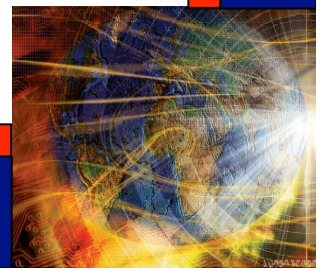
Forms of Equivalence Class Testing

- Normal: classes of valid values of inputs
- Robust: classes of valid and invalid values of inputs
- Weak: (single fault assumption) one from each class
- Strong: (multiple fault assumption) one from each class in Cartesian Product
- We compare these for a function of two variables, $F(x_1, x_2)$
- Extension to problems with 3 or more variables is “obvious”.

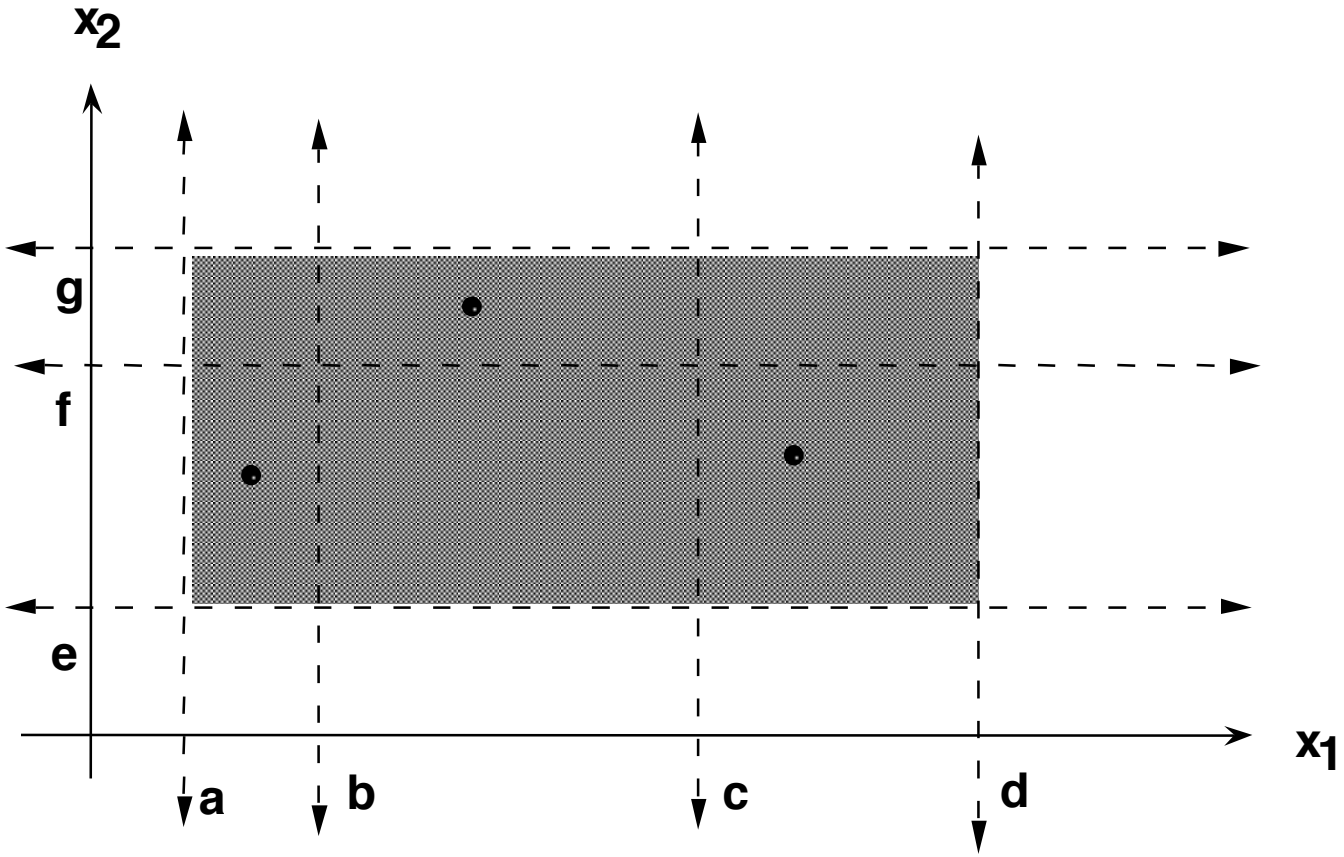


Weak Normal Equivalence Class Testing

- Identify equivalence classes of valid values.
- Test cases have all valid values.
- Detects faults due to calculations with valid values of a single variable.
- OK for regression testing.



Weak Normal Equivalence Class Test Cases



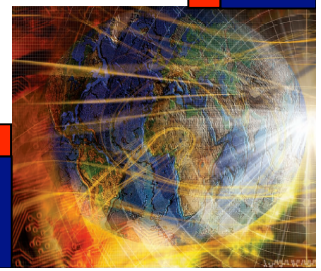
Equivalence classes (of valid values):

$\{a <= x_1 < b\}$, $\{b <= x_1 < c\}$, $\{c <= x_1 <= d\}$

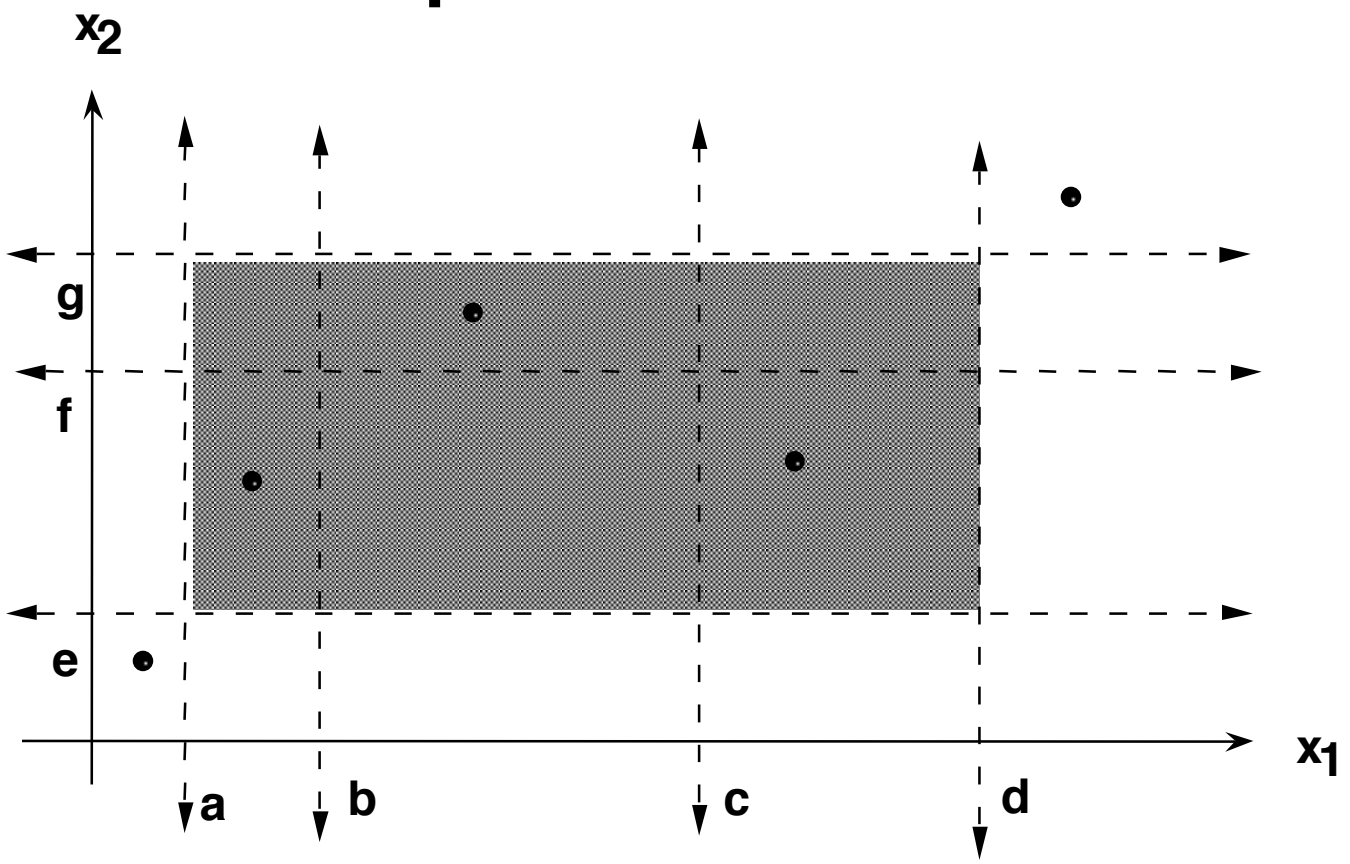
$\{e <= x_2 < f\}$, $\{f <= x_2 <= g\}$

Weak Robust Equivalence Class Testing

- Identify equivalence classes of valid and invalid values.
- Test cases have all valid values except one invalid value.
- Detects faults due to calculations with valid values of a single variable.
- Detects faults due to invalid values of a single variable.
- OK for regression testing.

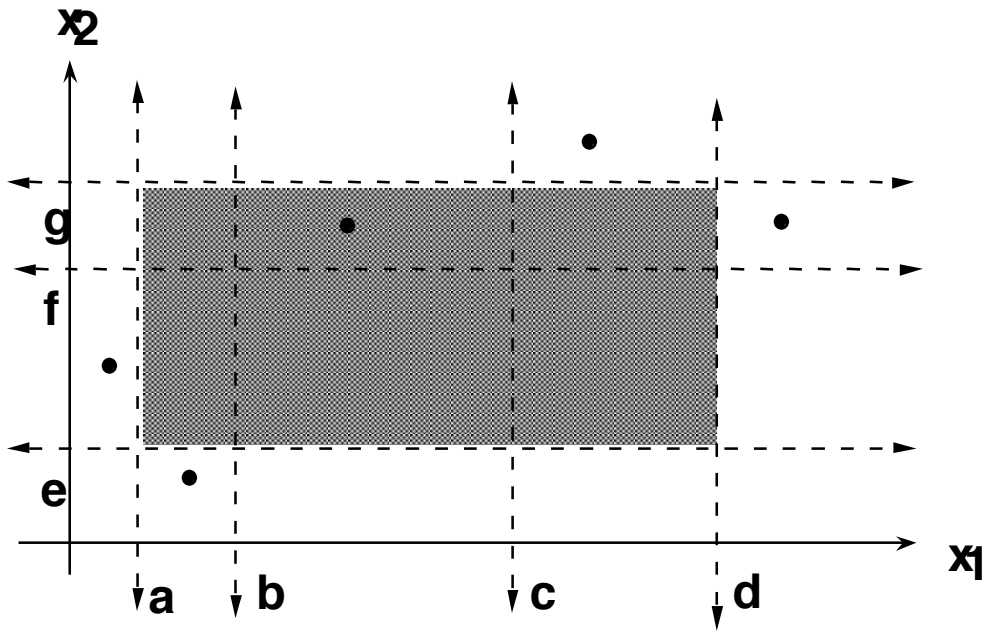


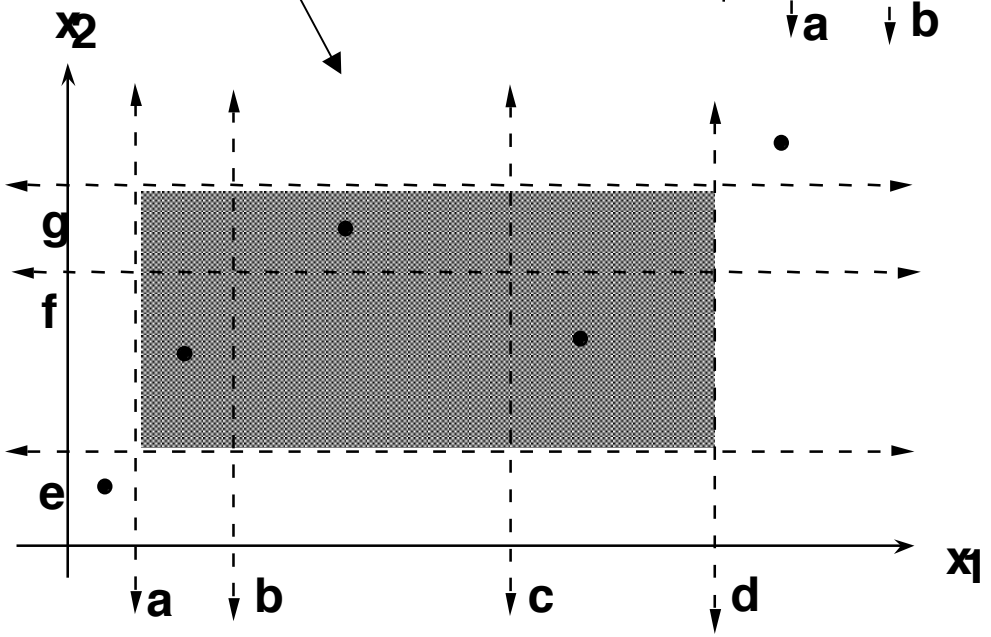
Weak Robust Equivalence Class Test Cases



Equivalence classes (of valid and invalid values):
{a <= x_1 < b}, {b <= x_1 < c}, {c <= x_1 <= d}, {e <= x_2 < f}, {f <= x_2 <= g}
Invalid classes: {x_1 < a}, {x_1 > d}, {x_2 < e}, {x_2 > g}

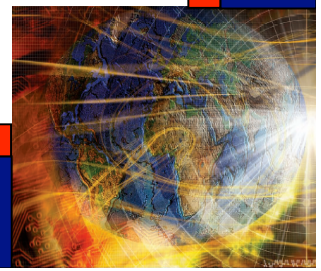
Is this preferable to this? Why?



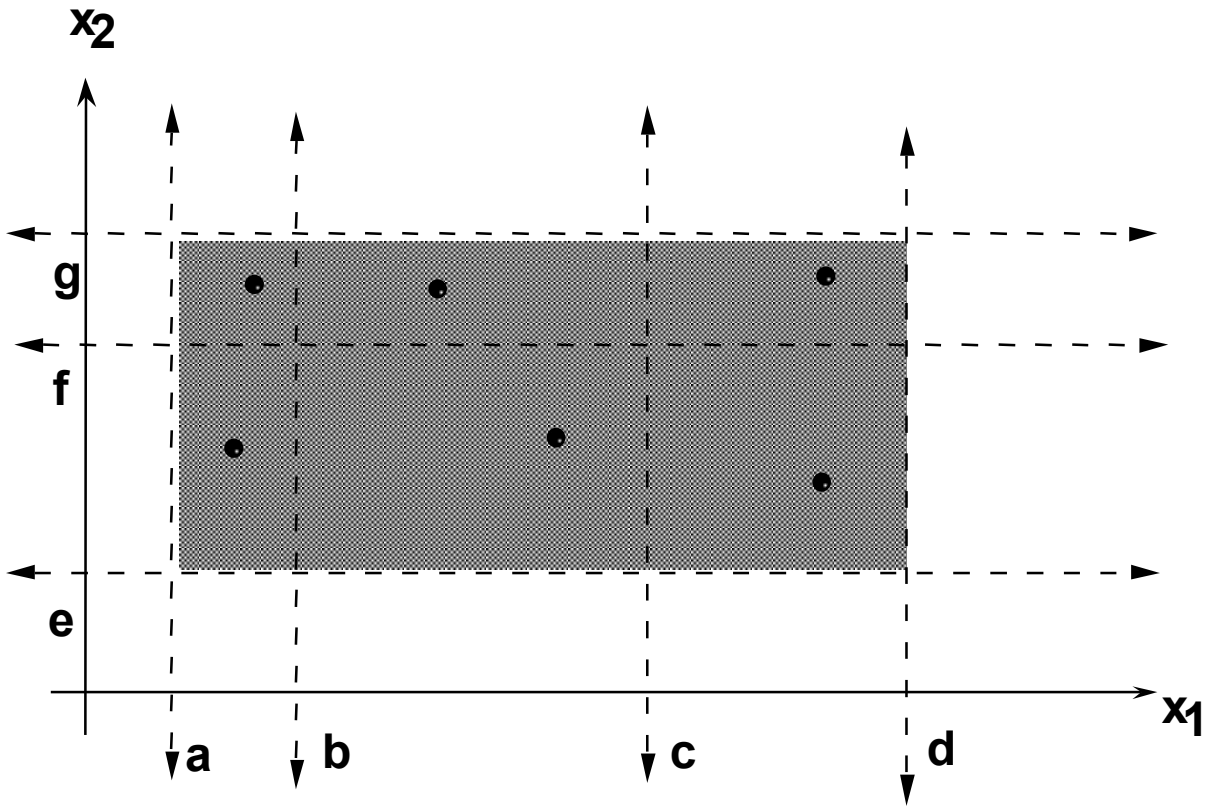


Strong Normal Equivalence Class Testing

- Identify equivalence classes of valid values.
- Test cases from Cartesian Product of valid values.
- Detects faults due to interactions with valid values of any number of variables.
- OK for regression testing, better for progression testing.



Strong Normal Equivalence Class Test Cases



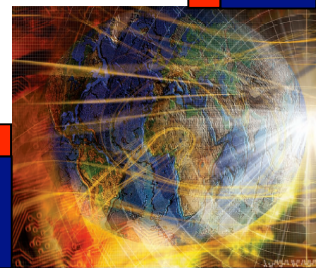
Equivalence classes (of valid values):

{a <= x_1 < b}, {b <= x_1 < c}, {c <= x_1 <= d}

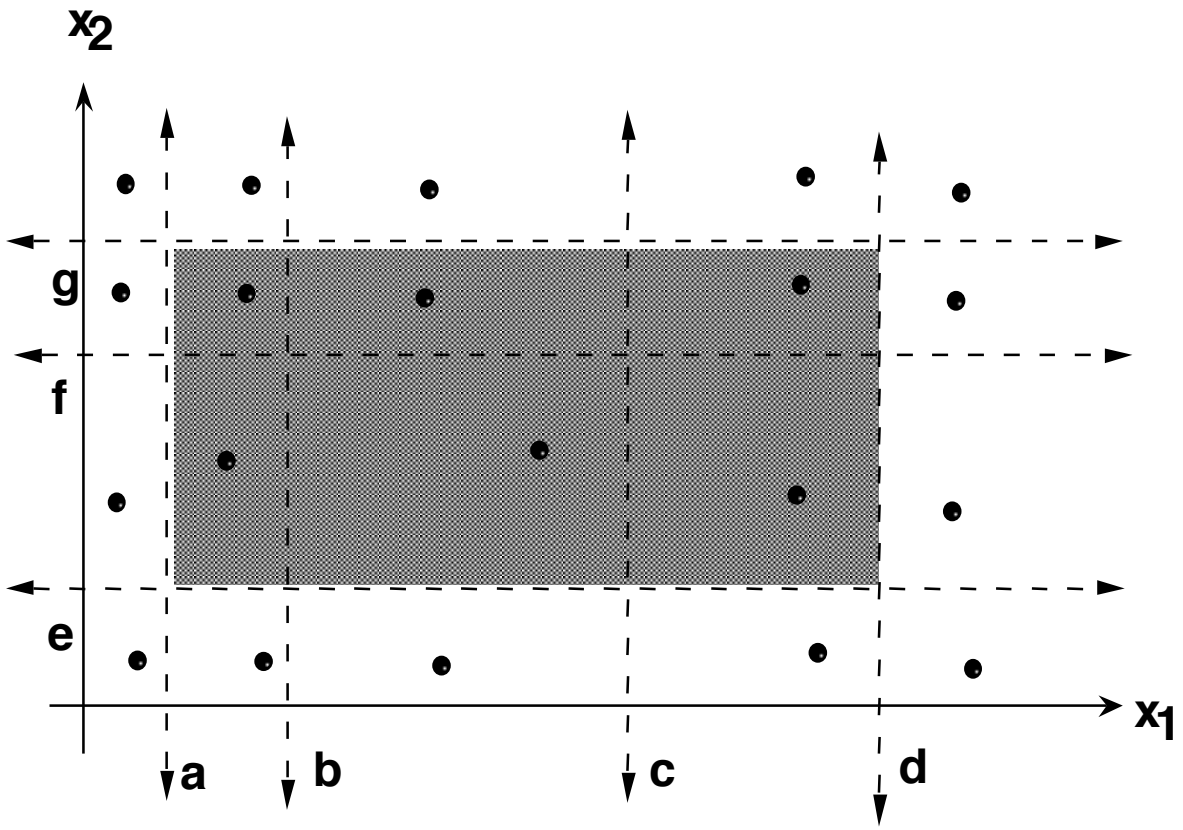
{e <= x_2 < f}, {f <= x_2 <= g}

Strong Robust Equivalence Class Testing

- Identify equivalence classes of valid and invalid values.
- Test cases from Cartesian Product of all classes.
- Detects faults due to interactions with any values of any number of variables.
- OK for regression testing, better for progression testing. (Most rigorous form of Equivalence Class testing, BUT, Jorgensen's First Law of Software Engineering applies.)
- Jorgensen's First Law of Software Engineering:
  - The product of two big numbers is a really big number.
  - (scaling up can be problematic)



Strong Robust Equivalence Class Test Cases



Equivalence classes (of valid and invalid values):

$\{a <= x_1 < b\}, \{b <= x_1 < c\}, \{c <= x_1 <= d\}, \{e <= x_2 < f\}, \{f <= x_2 <= g\}$

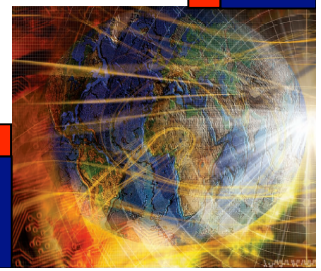
Invalid classes: $\{x_1 < a\}, \{x_1 > d\}, \{x_2 < e\}, \{x_2 > g\}$

Selecting an Equivalence Relation

There is no such thing as THE equivalence relation.
If x and y are days, some possibilities for Nextdate are:

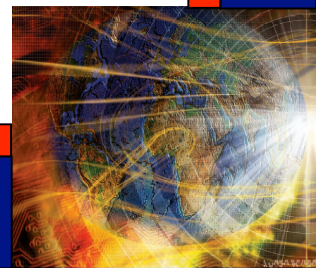
- x R y iff x and y are mapped onto the same year
- x R y iff x and y are mapped onto the same month
- x R y iff x and y are mapped onto the same date
- x R y iff x(day) and y(day) are "treated the same"
- x R y iff x(month) and y(month) are "treated the same"
- x R y iff x(year) and y(year) are "treated the same"

Best practice is to select an equivalence relation that reflects the behavior being tested.



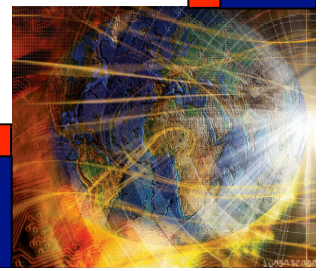
NextDate Equivalence Classes

- Month:
  - M1 = { month : month has 30 days}
  - M2 = { month : month has 31 days}
  - M3 = { month : month is February}
- Day
  - D1 = {day : 1 <= day <= 28}
  - D2 = {day : day = 29 }
  - D3 = {day : day = 30 }
  - D4 = {day : day = 31 }
- Year (are these disjoint?)
  - Y1 = {year : year = 2000}
  - Y2 = {year : 1812 <= year <= 2012 AND (year ≠ 0 Mod 100) and (year = 0 Mod 4)
  - Y3 = {year : (1812 <= year <= 2012 AND (year ≠ 0 Mod 4)



Not Quite Right

- A better set of equivalence classes for year is
  - Y1 = {century years divisible by 400} i.e., century leap years
  - Y2 = {century years not divisible by 400} i.e., century common years
  - Y3 = {non-century years divisible by 4} i.e., ordinary leap years
  - Y4 = {non-century years not divisible by 4} i.e., ordinary common years
- All years must be in range: 1812 <= year <= 2012
- Note that these equivalence classes are disjoint.

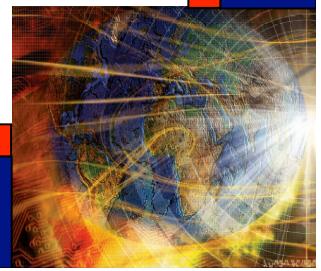


Weak Normal Equivalence Class Test Cases

Select test cases so that one element from each input domain equivalence class is used as a test input value.

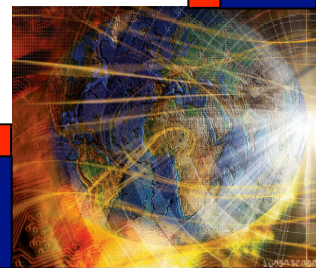
Test Case	Input Domain Equiv. Classes	Input Values	Expected Outputs
WN-1	M1, D1, Y1	April 1 2000	April 2 2000
WN-2	M2, D2, Y2	Jan. 29 1900	Jan. 30 1900
WN-3	M3, D3, Y3	Feb. 30 1812	impossible
WN-4	M1, D4, Y4	April 31 1901	impossible

Notice that all forms of equivalence class testing presume that the variables in the input domain are independent; logical dependencies are unrecognized.



Strong Normal Equivalence Class Test Cases

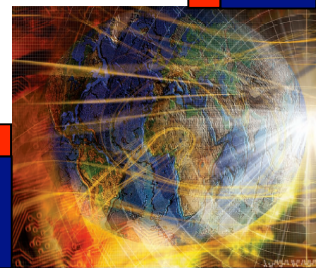
- With 4 day classes, 3 month classes, and 4 year classes, the Cartesian Product will have 48 equivalence class test cases. (Jorgensen's First Law of Software Engineering strikes again!)
- Note some judgment is required. Would it be better to have 5 day classes, 4 month classes and only 2 year classes? (40 test cases)
- Questions such as this can be resolved by considering Risk.



Revised NextDate Domain Equivalence Classes

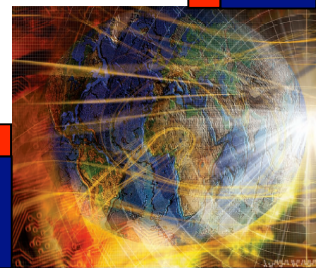
- Month:
  - M1 = { month : month has 30 days}
  - M2 = { month : month has 31 days except December}
  - M3 = { month : month is February}
  - M4 = {month : month is December}
- Day
  - D1 = {day : 1 <= day <= 27}
  - D2 = {day : day = 28 }
  - D3 = {day : day = 29 }
  - D4 = {day : day = 30 }
  - D5 = {day : day = 31 }
- Year (are these disjoint?)
  - Y1 = {year : year is a leap year}
  - Y2 = {year : year is a common year}

The Cartesian Product of these contains 40 elements.



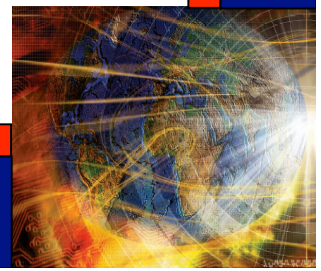
When to Use Equivalence Class Testing

- Variables represent logical (rather than physical) quantities.
- Variables "support" useful equivalence classes.
- Try to define equivalence classes for
  - The Triangle Problem
    - 0 < sideA < 200
    - 0 < sideB < 200
    - 0 < sideC < 200
  - The Commission Problem (exercise)



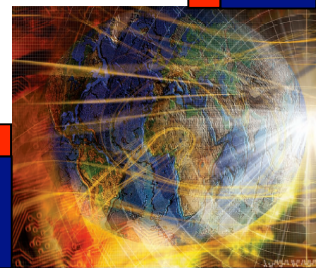
Another Equivalence Class Strategy

- "Work backwards" from output classes.
- For the Triangle Problem, could have
  - {x, y, z such that they form an Equilateral triangle}
  - {x, y, z such that they form an Isosceles triangle with x = y}
  - {x, y, z such that they form an Isosceles triangle with x = z}
  - {x, y, z such that they form an Isosceles triangle with y = z}
  - {x, y, z such that they form a Scalene triangle}
- How many equivalence classes will be needed for x,y,z Not a triangle?



In-Class Exercise

- Apply the "working backwards" approach to develop equivalence classes for the Commission Problem.
- Hint: use boundaries in the output space.



Assumption Matrix

	Valid Values	Valid and Invalid Values
Single fault	**Boundary Value** **Weak Normal Equiv. Class**	**Robust Boundary Value** **Weak Robust Equiv. Class**
Multiple fault	**Worst Case Boundary Value** **Strong Normal Equiv. Class**	**Robust Worst Case Boundary Value** **Strong Robust Equiv. Class**