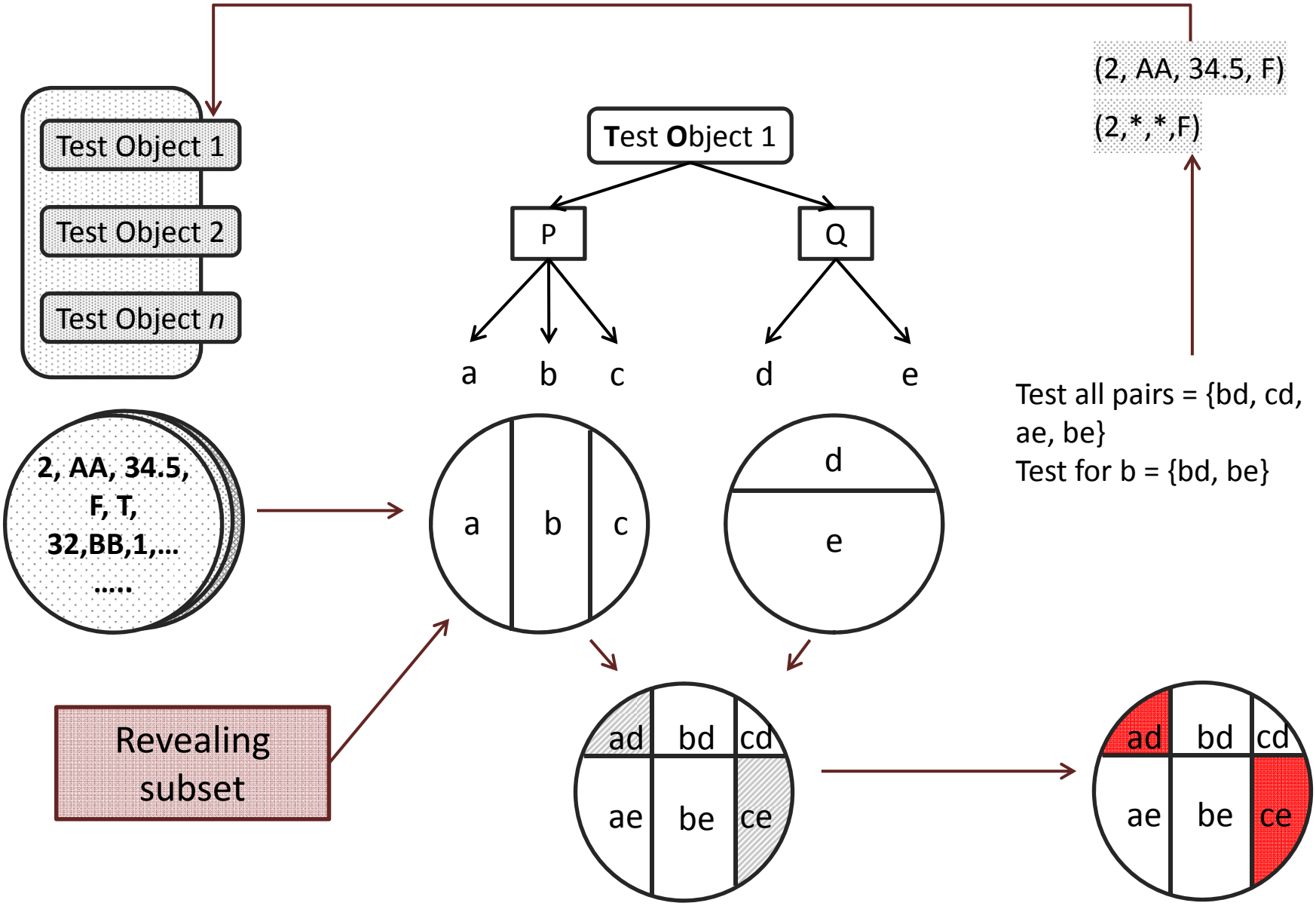
Integrating the classification tree method with combinatorial testing: test selection strategy

Eduardo Miranda, PhD.
CMU/MSE
MSE/NIST Seminar
June 7th, 2011
Pittsburgh, PA

The Classification Tree Method illustrated



Partition and boundary value analysis

- Two types of faults:
  - Computation faults: The wrong function is applied to some subdomain S_i in the implementation
  - Domain faults: The boundary between two subdomains S_i, S_j in the implementation is wrong
- Partition testing
  - Test inputs are designed to find computation faults
- Boundary value analysis
  - BVA aims to find domain faults by using test inputs near the boundaries

R. Hierons, Avoiding Coincidental Correctness in Boundary Value Analysis, 2006

How do we build confidence that a subset is revealing?

- If it is possible to establish an order relationship on the members of a given subset we can use boundary value analysis
  - Quantities
  - Lengths
  - Collection of elements
  - Positions
  - Distance
- Test catalogues
- Unless the subset has only one member, the absolute minimum number of values to test is 2

Simple test catalog

Boolean

[in/out] True

[in/out] False

Enumeration

[in/out] Each enumerated value

[in] Some value outside the enumerated set

Range *L* ...*U*

[in] *L* − 1 (the element immediately preceding the lower bound)

[in/out] *L* (the lower bound)

[in/out] A value between *L* and *U*

[in/out] *U* (the upper bound)

[in] *U* + 1 (the element immediately following the upper bound)

Numeric Constant *C*

[in/out] *C* (the constant value)

[in] *C* − 1 (the element immediately preceding the constant value)

[in] *C* + 1 (the element immediately following the constant value)

[in] Any other constant compatible with *C*

Non-Numeric Constant *C*

[in/out] *C* (the constant value)

[in] Any other constant compatible with *C*

[in] Some other compatible value

Sequence

[in/out] Empty

[in/out] A single element

[in/out] More than one element

[in/out] Maximum length (if bounded) or very long

[in] Longer than maximum length (if bounded)

[in] Incorrectly terminated

Scan with action on elements *P*

[in] *P* occurs at beginning of sequence

[in] *P* occurs in interior of sequence

[in] *P* occurs at end of sequence

[in] *PP* occurs contiguously

[in] *P* does not occur in sequence

[in] *pP* where *p* is a proper prefix of *P*

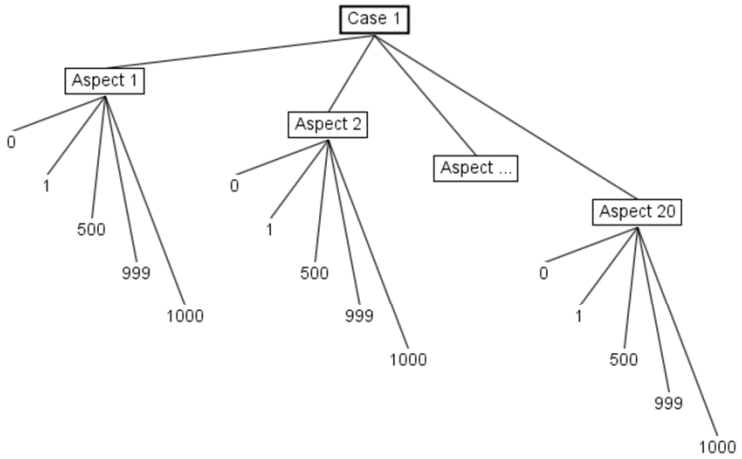
[in] Proper prefix *p* occurs at end of sequence

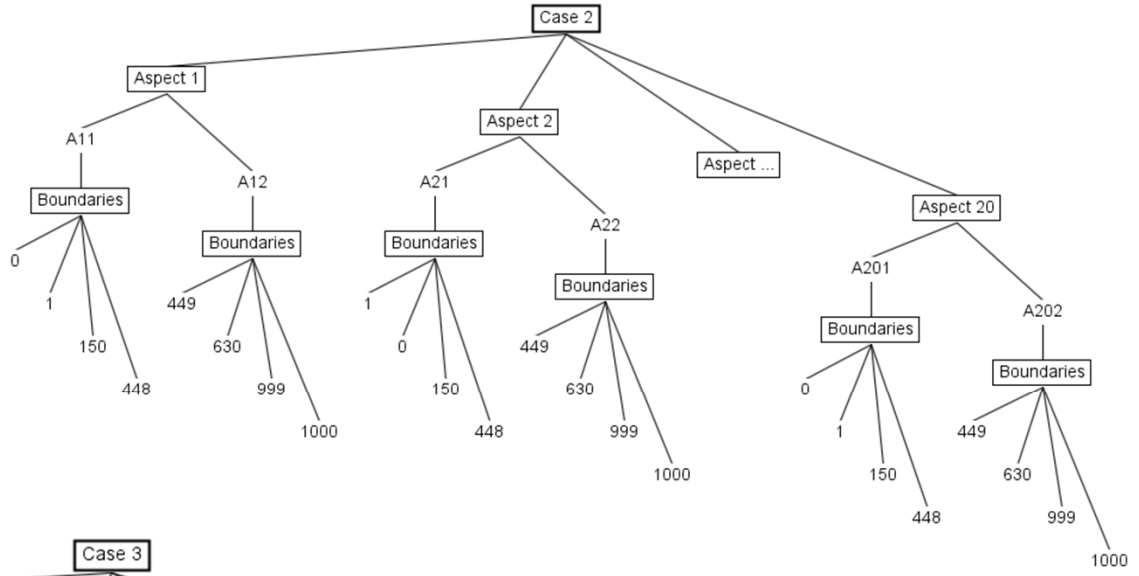
Software Testing and Analysis—Process, Principles, and Techniques, M. Pezzè & M. Young, 2008

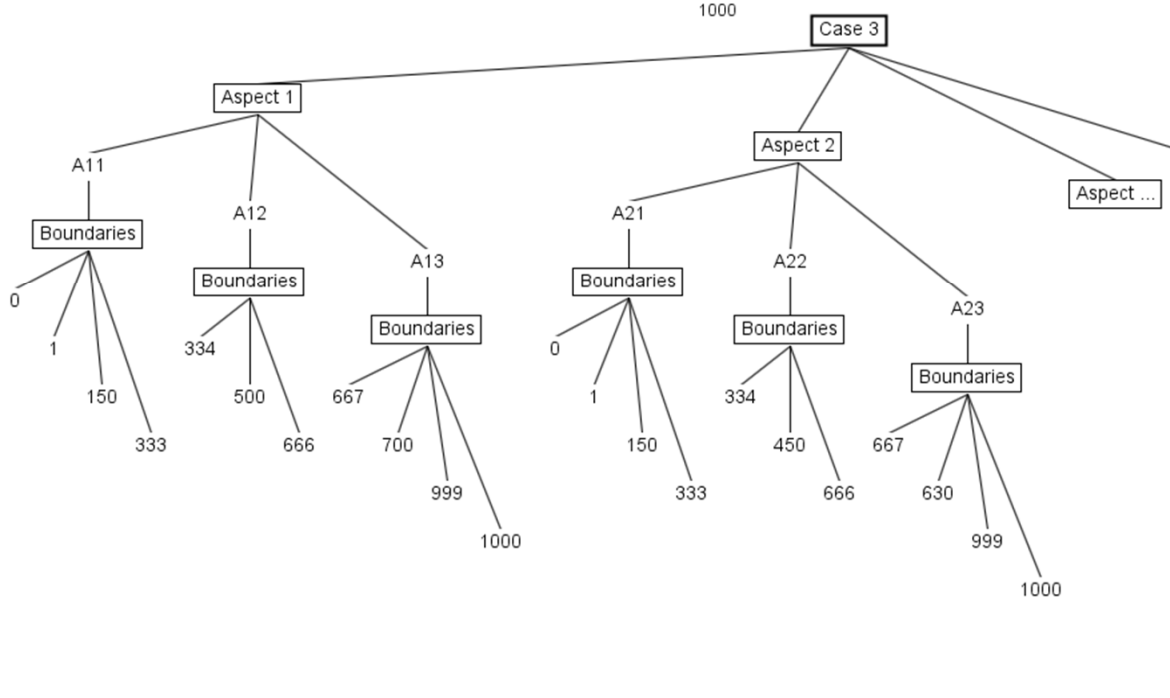
At what level should we employ interaction testing?

- If we are using BVA to establish confidence in a subdomain:
  - Should we use the BVA values (left-, left, middle, right, right +) in the interactions?
  - Should we first establish confidence on the equivalence class and then test for interactions?
  - What happens if we are not using BVA?
- What do we do with invalid values?

To answer the previous questions we will look at three scenarios

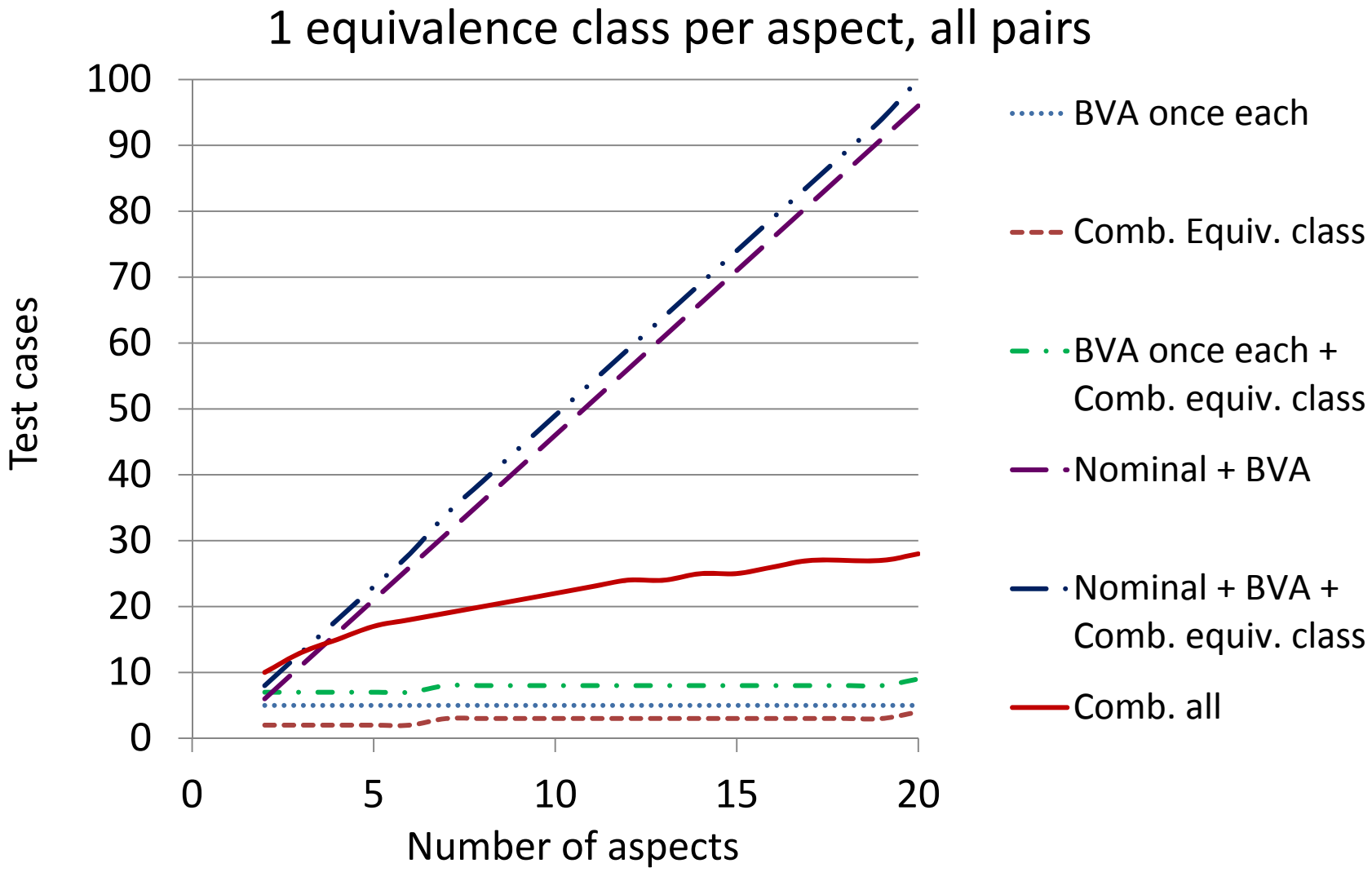




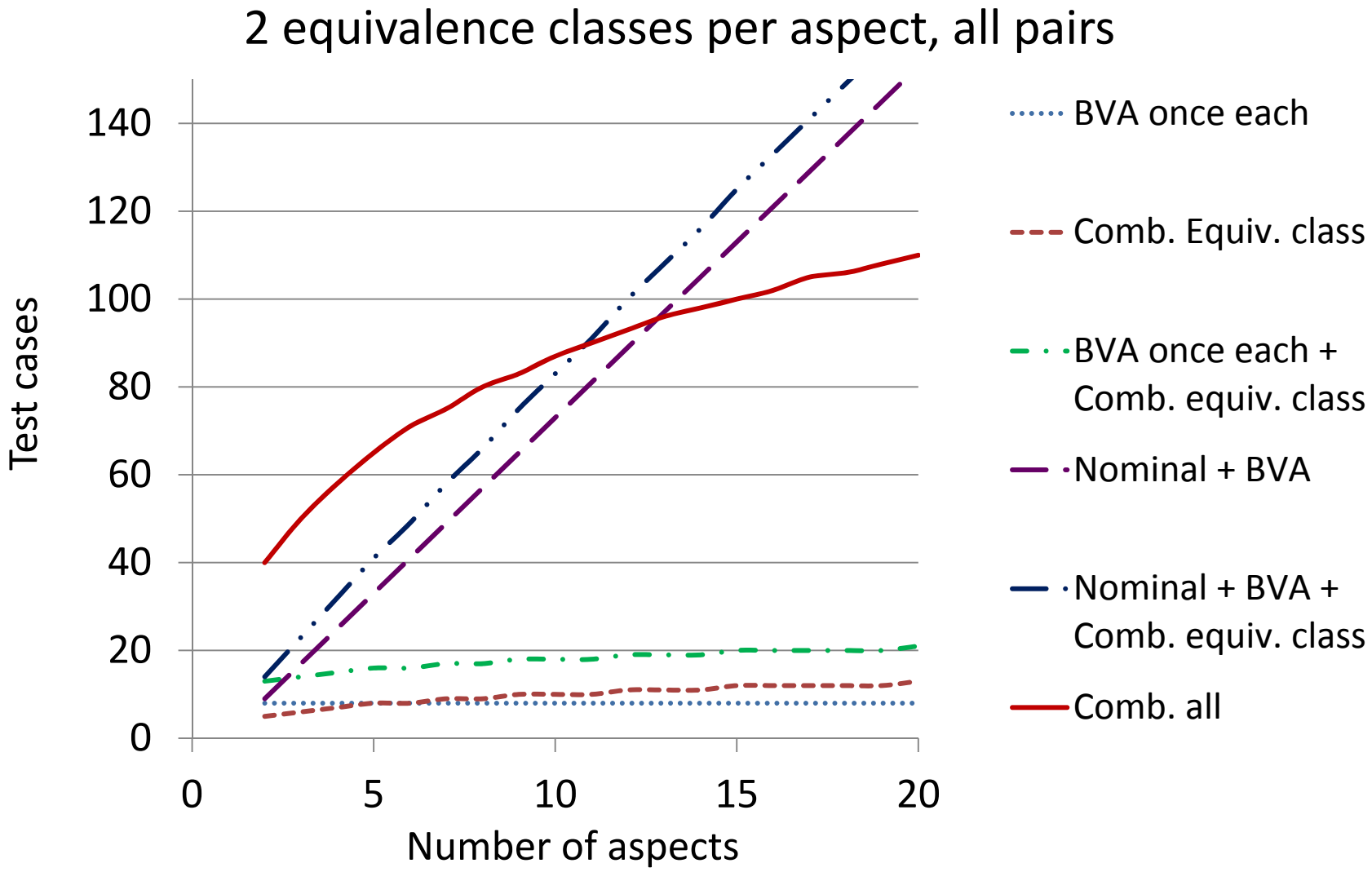


Range for each aspect [1,999]

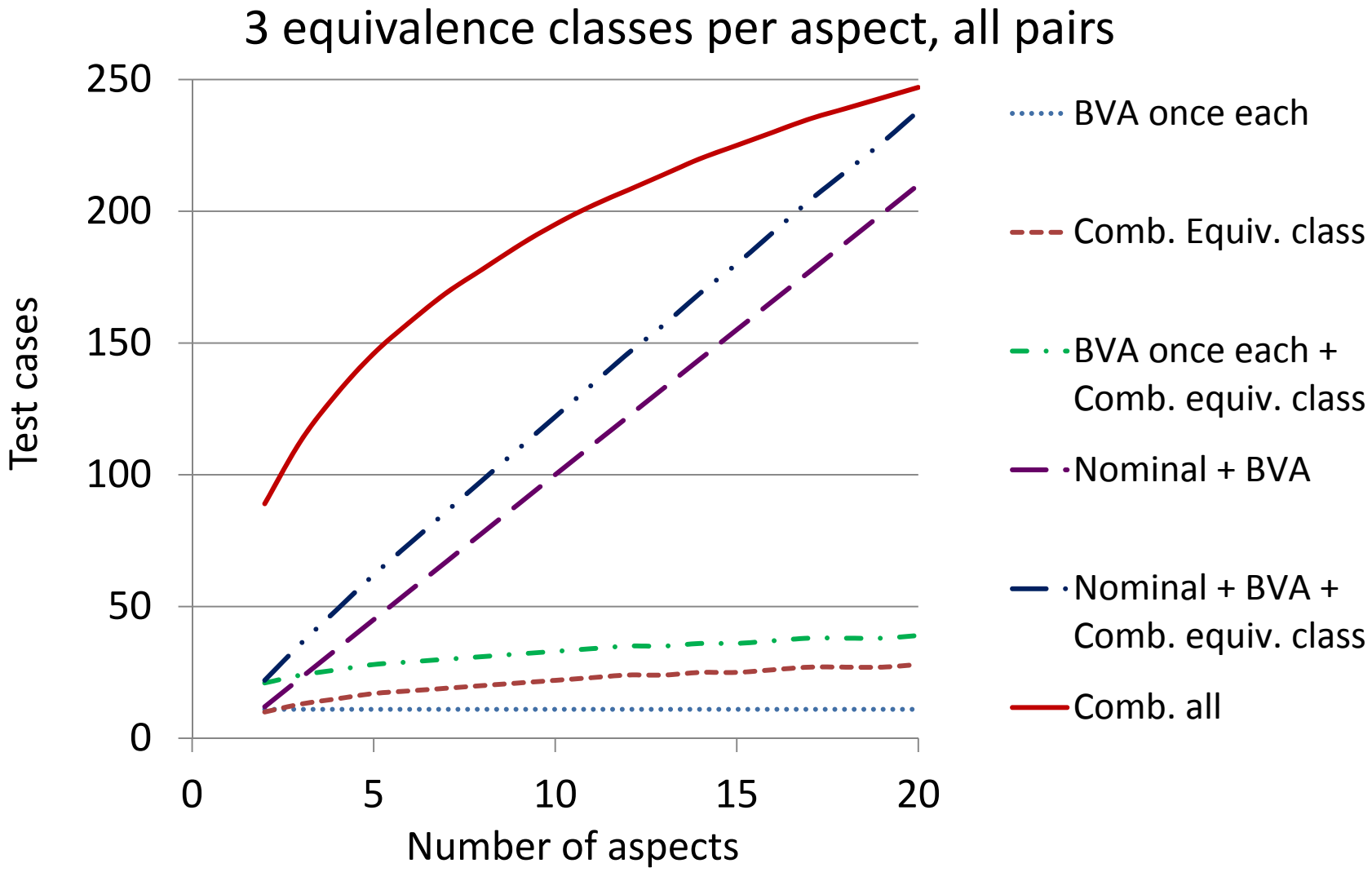
Number of test cases required by different strategies (1)



Number of test cases required by different strategies (2)



Number of test cases required by different strategies (3)



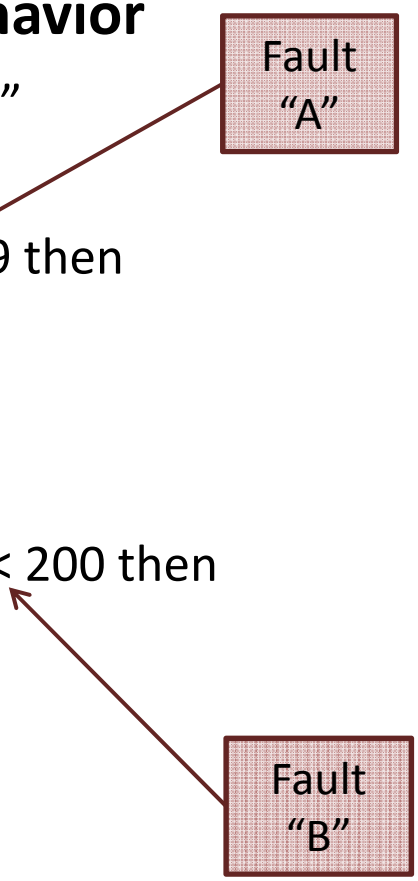
Comparison of different strategies on the basis of fault finding effectiveness (1)

intended behavior

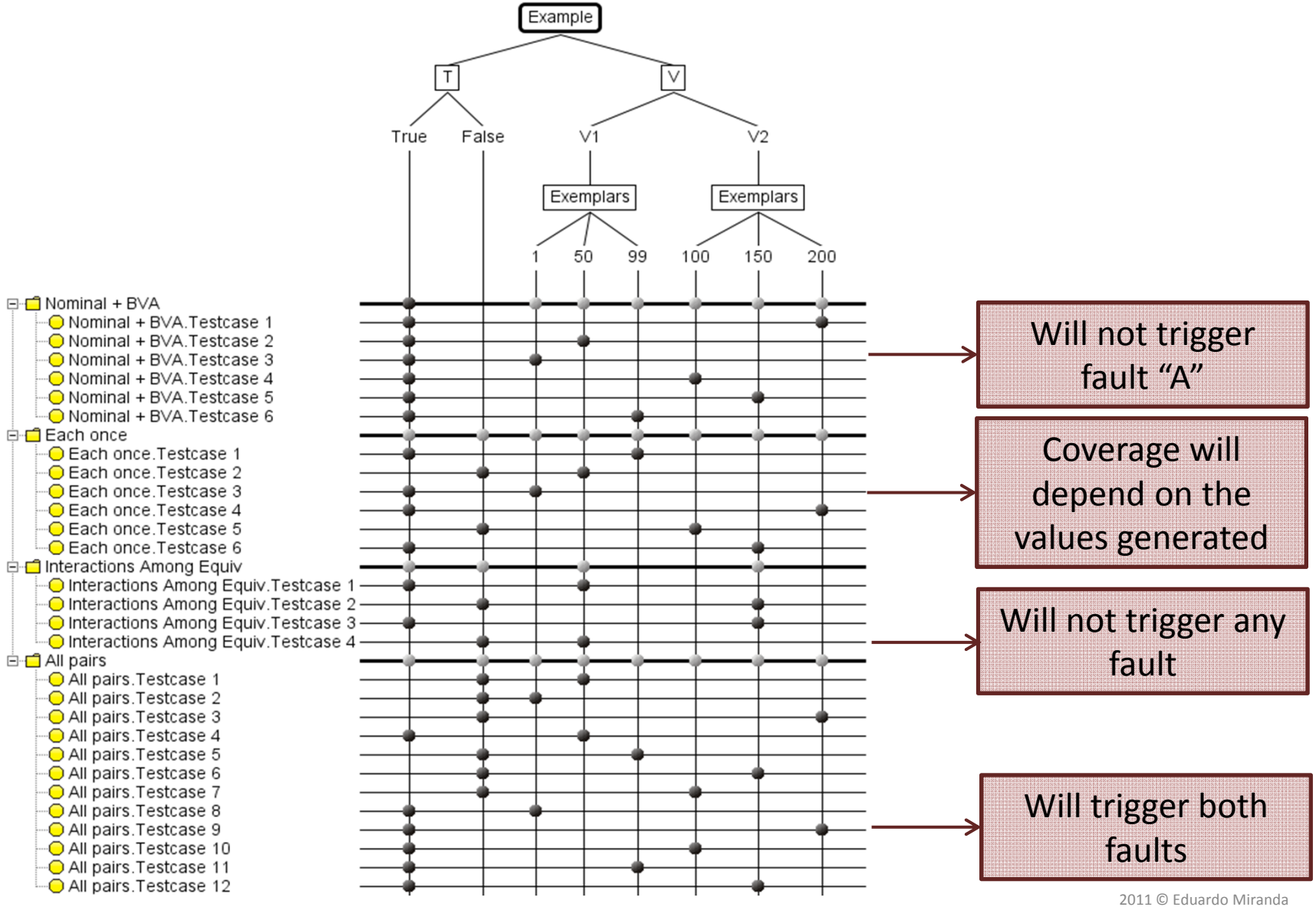
1. X = “Don’t know”
2. if T then
    1. if 1<= V <= 99 then
        1. x = “V1”
    2. endif
3. else
    1. If 100 <= V <= 200 then
        1. x = “V2”
    2. endif
4. endif
5. print (x)

programmed behavior

1. X = “Don’t know”
2. if T then
    1. if 1<= V < 99 then
        1. x = “V1”
    2. endif
3. else
    1. If 100 <= V < 200 then
        1. x = “V2”
    2. endif
4. endif
5. print (x)



Comparison of different strategies on the basis of fault finding effectiveness (2)



What about negative testing

- Should we mix positive and negative testing?
- Should we test for combinations of invalid values or it suffices with all singles testing?

Fault masking

All pairs array

	V	U	Z
1	V1	1	Z1
2	V2	2	Z1
3	V3	3	Z1
4	Invalid	4	Z1
5	V1	2	Z2
6	V2	1	Z2
7	V3	4	Z2
8	Invalid	3	Z2
9	V1	3	Z3
10	V2	4	Z3
11	V3	1	Z3
12	Invalid	2	Z3
13	V1	4	Z4
14	V2	3	Z4
15	V3	2	Z4
16	Invalid	1	Z4
17	V1	1	Z5
18	V2	2	Z5
19	V3	3	Z5
20	Invalid	4	Z5

if V = "Invalid" *then*

display message

exit

endif

.

.

if z = z1 *then*

x := 6 / (u - 4)

endif

The fault is masked by the invalid value in row 4

Exercising invalid combinations

Is V1 = invalid, V2 = invalid and V3 = invalid a good test case?

```
if  V1 is invalid then
    display Error1
    exit
endif
if  V2 is invalid then
    display Error2
    exit
endif
If  V3 is invalid then
    display Error3
    exit
endif
```

```
if  V1 is invalid then
    V1 = DefaultValue1
endif
if  V2 is invalid then
    V2 = DefaultValue2
endif
if  V3 is invalid then
    V3 = DefaultValue3
endif
```

Summary

- In testing you get what you pay for. Do not compare the cost of a method to the cost of not testing, because the latter will always require less test cases
- Combinatorial testing
  - Very good and efficient at finding computational faults
  - Very good, but expensive at finding domain problems. Still, what would be a better approach is an open issue
- Unless there is a fault model conjecture for doing something else:
  - Keep negative testing separate from positive one
  - Don not combine invalid values
- To verify equivalence within a class
  - Combinations might act as confounders
  - For 1 or 2 equivalence classes within each aspect, combinatorial testing is feasible and effective. For three or more, the number of test cases required becomes very large and all singles might be a good alternative
  - One at a time testing will eliminate confounding but interactions might mask faults and will require a large number of test cases

Questions?