Announcements

- Friday, Sept. 29th: No face-to-face class on Friday **BUT** I will post a video on the website and you must watch it before coming to class on Monday!
- Monday, Oct. 2nd :
  - SWE Industrial dinner (check the following slides for more details)
  - Talk from CrowdStrike, don't be late! (https://www.crowdstrike.com/)

    CrowdStrike Falcon Host is a two-component security product. One component is a "sensor", which is an agent installed on client machines that observes system activity and recognizes malicious behavior, then provides on-box prevention capability and remote telemetry to the Falcon Host cloud. The cloud component aggregates sensor telemetry for each customer's network, can correlate malicious behavior across multiple machines, and presents our customers' operations teams with a prioritized summary of the threats detected in their environments.


WASHINGTON STATE UNIVERISTY SOCIETY OF WOMEN ENGINEERS
INVITES YOU TO THE 19TH ANNUAL
EVENING
With
INDUSTRY
Buy your ticket on Eventbrite today!
wsu.swe@wsu.edu
2nd OCTOBER 2017
SEL EVENT CENTER
SOCIAL HOUR
5:30-6:30 pm
DINNER TO FOLLOW
All Engineering Disciplines Welcome
swe
Society of Women Engineers
Washington State University

swe
Society of Women Engineers
Washington State University

Details

- ALL engineering students are welcome!
- Twenty companies will be represented, from all disciplines of engineering.
- Dress code: business casual
- Social hour begins at 5:30pm, dinner and dessert will be served
- Three drawings for raffle prizes

Tickets

- Buy your ticket today at:

  https://www.eventbrite.com/e/evening-with-industry-tickets-37713551247

  [Eventbrite -> Evening with Industry]

- Feel free to drop by the EME bridge on the 2nd floor of Sloan for tickets/more information

- Ticket price: $15

Cpt S 422: Software Engineering Principles II

Black-box testing – Part 1

Dr. Venera Arnaoudova

WASHINGTON STATE UNIVERSITY

Basic Principles

- Based on the definition of what a program's specification, as opposed to its structure
- The notion of complete coverage can also be applied to functional testing
- Rigorous specifications have another benefit, they help functional testing, (e.g., categorize inputs, derive expected outputs) and thus they help generating test cases and oracles

Black-box testing methods

- Equivalence Class Partitioning
- Boundary-Value Analysis
- Category-Partition
- Decision tables
- Cause-Effect Graphs
- Logic Functions

Equivalence Class Partitioning Testing

Equivalence Class Partitioning Testing

- Motivation: we would like to have a sense of complete testing and we would hope to avoid redundancy
- Equivalence classes: partition of the input set
- Entire input set is covered: completeness
- Disjoint classes: avoid redundancy
- Test cases: one element of each equivalence class
- But equivalence classes have to be chosen wisely …
- Guessing the likely underlying system behavior …

Weak/Strong Equ. Class Testing

- Three input variables of domains: A, B, C
  - $A = A1 \cup A2 \cup A3$ where $a_n \in An$
  - $B = B1 \cup B2 \cup B3 \cup B4$ where $b_n \in Bn$
  - $C = C1 \cup C2$ where $c_n \in Cn$

- **Weak Equivalence Class Testing (WECT)**: choose one variable value form each equivalence class
  - max(|A|, |B|, |C|) test cases

- **Strong Equivalence Class Testing (SECT)**: based on the Cartesian product of the partition subsets ($A \times B \times C$), i.e., test all class interactions
  - |A| x |B| x |C| test cases

WECT Test Frames

	A	B	B
WECT1	A1	B1	C1
WECT2	A2	B2	C2
WECT3	A3	B3	C1
WECT4	A1	B4	C2

SECT Test Frames

	A	B	C
SECT1	A1	B1	C1
SECT2	A1	B1	C2
SECT3	A1	B2	C1
SECT4	A1	B2	C2
SECT5	A1	B3	C1
SECT6	A1	B3	C2
SECT7	A1	B4	C1
SECT8	A1	B4	C2
SECT9	A2	B1	C1
SECT10	A2	B1	C2
SECT11	A2	B2	C1
SECT12	A2	B2	C2
SECT13	A2	B3	C1
SECT14	A2	B3	C2
SECT15	A2	B4	C1
SECT16	A2	B4	C2
SECT17	A3	B1	C1
SECT18	A3	B1	C2
SECT19	A3	B2	C1
SECT20	A3	B2	C2
SECT21	A3	B3	C1
SECT22	A3	B3	C2
SECT23	A3	B4	C1
SECT24	A3	B4	C2

`NextDate` Example

- `NextDate` is a function with three variables: `month, day, year`. It returns the date of the day after the input date. Limitation: 1700-2017

- Treatment Summary:
  - if it is not the last day of the month, the next date function will simply increment the value of `day`.
  - At the end of a month, the next day is 1 and `month` is incremented.
  - At the end of the year, both `day` and `month` are reset to 1, and the year incremented.
  - The problem of leap year makes determining the last day of a month interesting.

`NextDate` Equivalence Classes

- M1 = {month: month has 30 days} // e.g., month = 6
- M2 = {month: month has 31 days} // e.g., month = 7
- M3 = {month: month is February} // i.e., month = 2
- D1 = {day: 1<= day <= 28} // e.g., day = 14
- D2 = {day: day = 29}
- D3 = {day: day = 30}
- D4 = {day: day = 31}
- Y1 = {year: century leap year} // i.e., year = 2000; 1700 <= year <= 2017
- Y2 = {year: century common year} // e.g., year = 1900; 1700 <= year <= 2017
- Y3 = {year: non-century leap year} // e.g., year = 2012; 1700 <= year <= 2017
- Y4 = {year: year: non-century common year} // e.g., year = 2007; 1700 <= year <= 2017

NextDate Tests: WECT

- WECT: 4 test cases- maximum partition (D and Y)

  - **Test Frames:**

ID:	Month	Day	Year
WECT1	M1	D1	Y1
WECT2	M2	D2	Y2
WECT3	M3	D3	Y3
WECT4	M1	D4	Y4

  - **Test Cases:**

ID:	Month	Day	Year	Output
WECT1	6	14	2000	6/15/2000
WECT2	7	29	1900	7/30/1900
WECT3	2	30	1912	Invalid Input
WECT4	6	31	2007	Invalid Input

NextDate Tests: SECT (48 tests)

	Month	Day	Year	Expected Output
SECT1	6	14	2000	6/15/2000
SECT2	6	14	1900	6/15/1900
SECT3	6	14	1912	6/15/1912
SECT4	6	14	2007	6/15/2007
SECT5	6	29	2000	6/30/2000
SECT6	6	29	1900	6/30/1900
SECT7	6	29	1912	6/30/1912
SECT8	6	29	2007	6/30/2007
SECT9	6	30	2000	7/1/2000
SECT10	6	30	1900	7/1/1900
SECT11	6	30	1912	7/1/1912
SECT12	6	30	2007	7/1/2007
SECT13	6	31	1913	ERROR
SECT14	6	31	1900	ERROR
SECT15	6	31	1912	ERROR
SECT16	6	31	1913	ERROR
SECT17	7	14	2000	7/15/2000
SECT18	7	14	1900	7/15/1900
SECT19	7	14	1912	7/15/1912
SECT20	7	14	2007	7/15/2007
SECT21	7	29	2000	7/30/2000
SECT22	7	29	1900	7/30/1900
SECT23	7	29	1912	7/30/1912
SECT24	7	29	2007	7/30/2007
SECT25	7	30	2000	7/31/2000
SECT26	7	30	1900	7/31/1900
SECT27	7	30	1912	7/31/1912
SECT28	7	30	2007	7/31/2007
SECT29	7	31	1913	8/1/2000
SECT30	7	31	1900	8/1/1900
SECT31	7	31	1912	8/1/1912
SECT32	7	31	1913	8/1/2007
SECT33	2	14	2000	2/15/2000
SECT34	2	14	1900	2/15/1900
SECT35	2	14	1912	2/15/1912
SECT36	2	14	2007	2/15/2007
SECT37	2	29	2000	3/1/2000
SECT38	2	29	1900	ERROR
SECT39	2	29	1912	3/1/1912
SECT40	2	29	2007	ERROR
SECT41	2	30	2000	ERROR
SECT42	2	30	1900	ERROR
SECT43	2	30	1912	ERROR
SECT44	2	30	2007	ERROR
SECT45	2	31	1913	ERROR
SECT46	2	31	1900	ERROR
SECT47	2	31	1912	ERROR
SECT48	2	31	1913	ERROR

Discussion

- If error conditions are a high priority, we should extend strong equivalence class testing to include invalid classes:
  - **Weak Robust Equivalence Class Testing (WRECT)**
  - **Strong Robust Equivalence Class Testing (SRECT)**
- ECT is appropriate when input data defined in terms of ranges and sets of discrete values
- SECT makes the assumption that the variables are independent – dependencies will generate "error" test cases (possibly too many of them …)
- See the category-partition and decision table techniques next to address this issue

Black-box testing methods

- ✓ Equivalence Class Partitioning
- ❑ Boundary-Value Analysis
- ❑ Category-Partition
- ❑ Decision tables
- ❑ Cause-Effect Graphs
- ❑ Logic Functions

Boundary Value Analysis Testing

Motivations

- We have partitioned input domains into suitable classes, on the assumption that the behavior of the program is “similar”
- Some typical programming errors happen to be at the boundary between different classes
- This is what boundary value testing focuses on
- Simpler but complementary to previous techniques

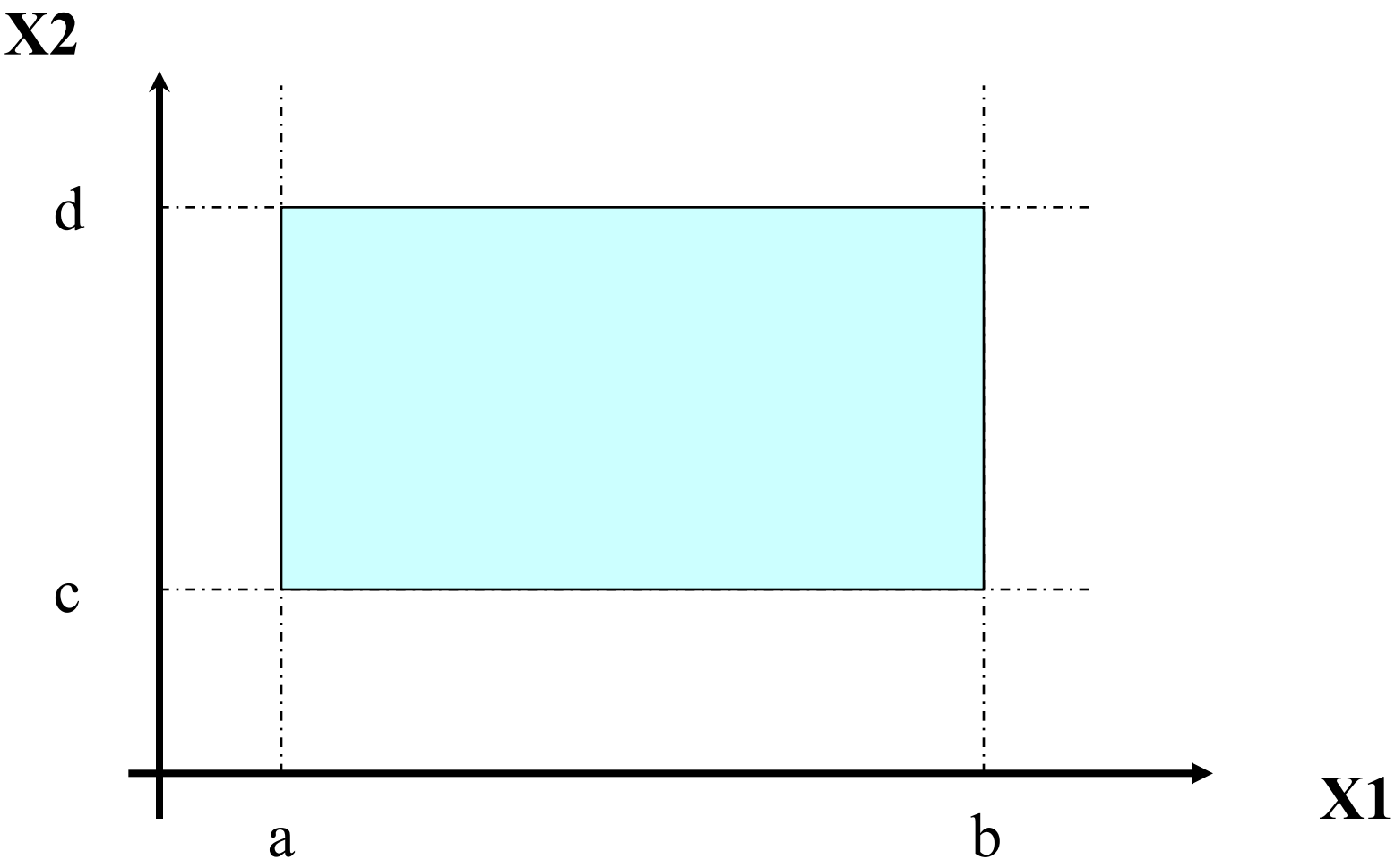
Boundary Value Analysis

- Assume a function F, with two variables x1 and x2
- (Possibly unstated) boundaries: a <= x1 <= b, c <= x2 <= d
- In some programming language, strong typing allows the specification of such intervals (e.g., Ada and Pascal)
- Focus on the boundary of the input space for identifying test cases
- The rationale is that errors tend to occur near extreme values of input variables – supported by some studies

Basic Ideas

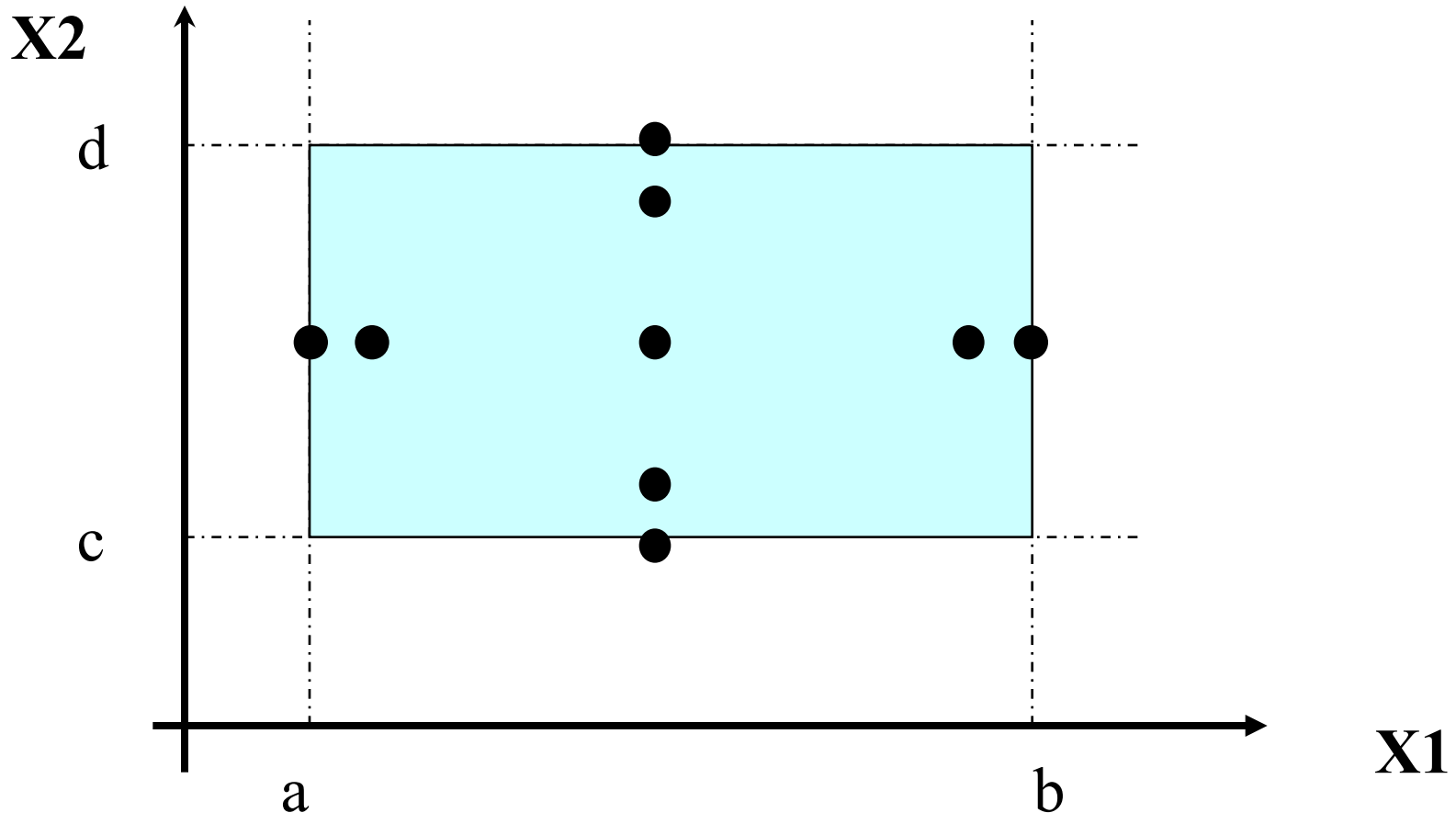
- Input variable values at their minimum, just above the minimum, a nominal value, just below their maximum, and at their maximum.
- Convention: min, min+, nom, max-, max
- Hold the values of all but one variable at their nominal values, letting one variable assume its extreme value

Input Domain of Function F



Boundary Analysis Tests

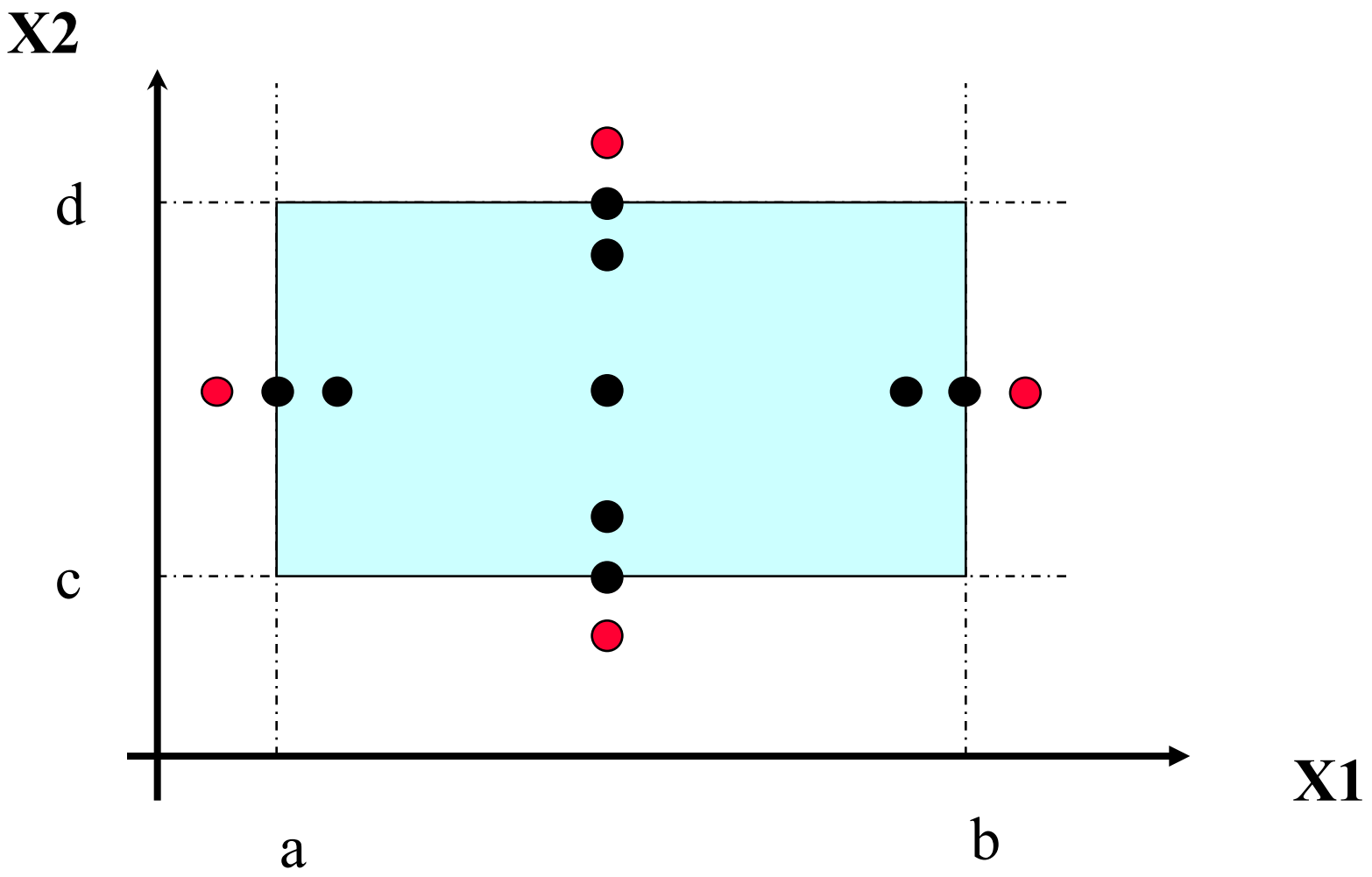
- Test set ={<$x1_{nom}$, $x2_{min}$>, <$x1_{nom}$, $x2_{min+}$>, <$x1_{nom}$, $x2_{nom}$>, <$x1_{nom}$, $x2_{max-}$>, <$x1_{nom}$, $x2_{max}$>, <$x1_{min}$, $x2_{nom}$,>, <$x1_{min+}$, $x2_{nom}$,>, <$x1_{max-}$, $x2_{nom}$>, <$x1_{max}$, $x2_{nom}$>}



General Case and Limitations

- A function with n variables will require **4n + 1** test cases
- Works well with variables that represent bounded physical quantities
- No consideration of the nature of the function and the meaning of variables
- Rudimentary technique that is amenable to robustness testing

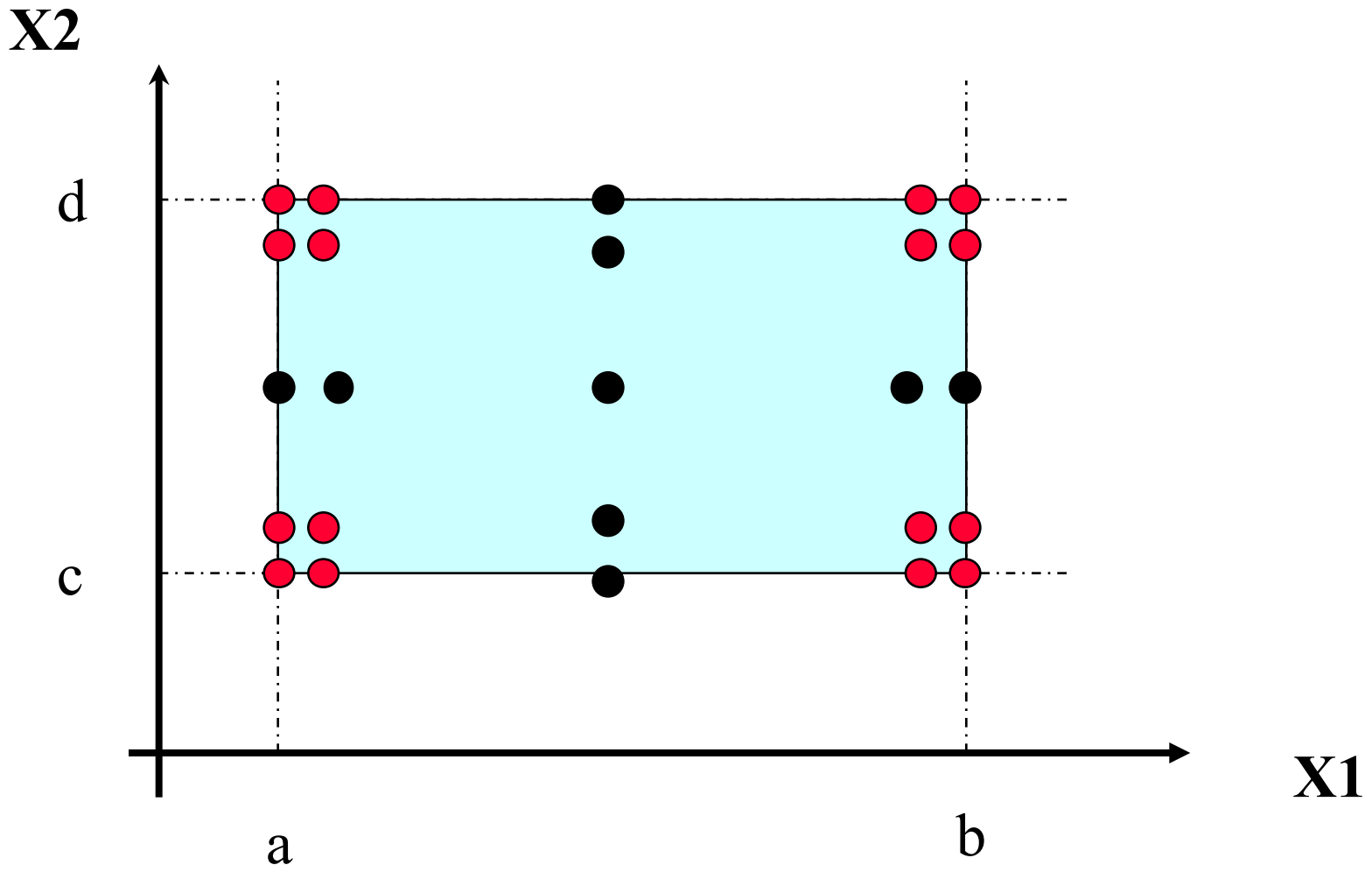
Robustness Testing



Worst Case Testing (WCT)

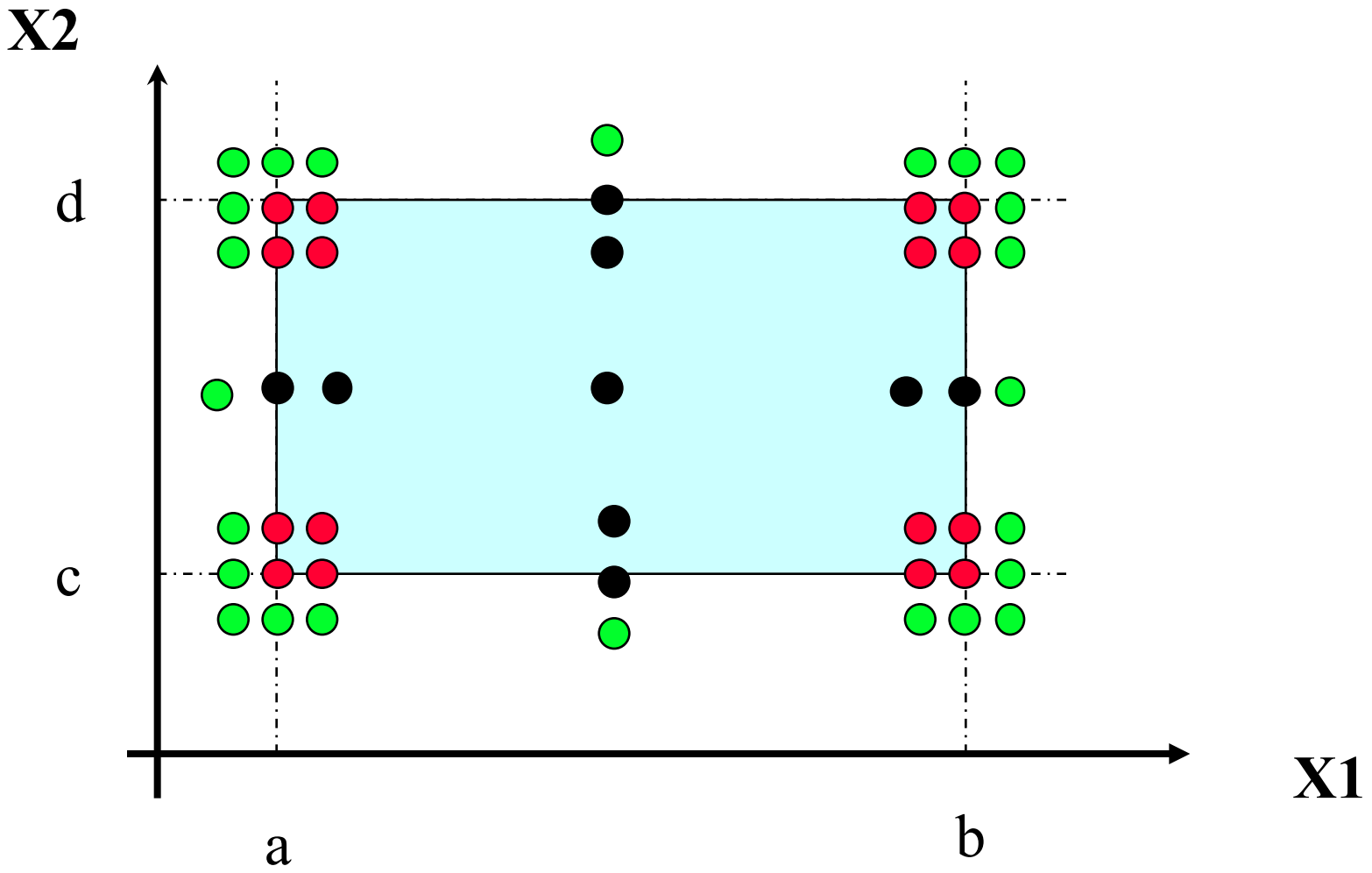
- Boundary value makes the common assumption that failures, most of the time, originate from one fault
- What happens when more than one variable has an extreme value?
- Cartesian product of {min, min+, nom, max-, max}
- Clearly more thorough than boundary value analysis, but much more effort: **5^n test cases**
- Good strategy when physical variables have numerous interactions, and where failure is costly

WCT for 2 variables



Robust WCT for 2 variables

- Even further: Robust Worst Case Testing



Tasks for today

1. Implement the `nextDay` method in the Calendar example

2. Test it using

   a) Equivalence Class Partitioning

   b) Boundary-Value Analysis

3. Evaluate the quality of the tests