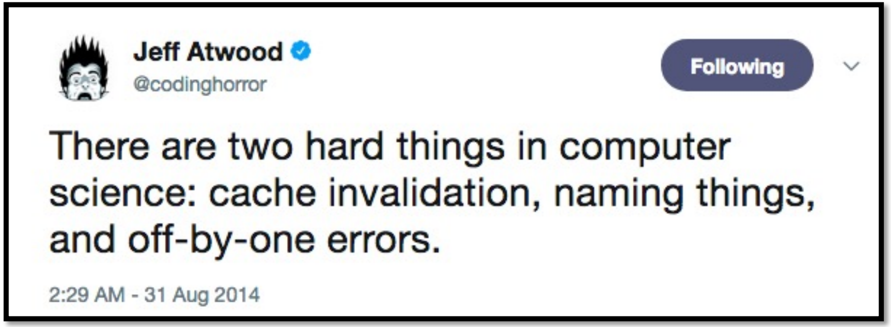
SPEC-BASED TESTING

Incorporating Boundary Values



Jeff Atwood
@codinghorror

There are two hard things in computer science: cache invalidation, naming things, and off-by-one errors.

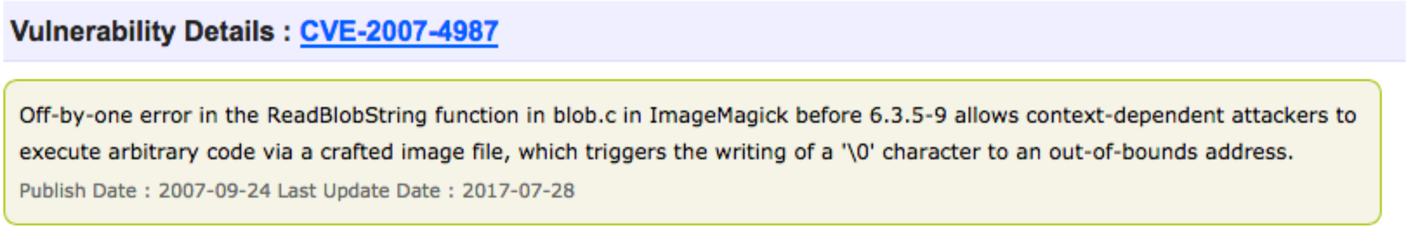
2:29 AM - 31 Aug 2014

An “off-by-one error” is caused by a program not properly handling boundary values

“An off-by-one error is a logic error that involves a numerical value incorrectly bigger or smaller by one.”

Generalization: **boundry-value fault**

A real-life example:

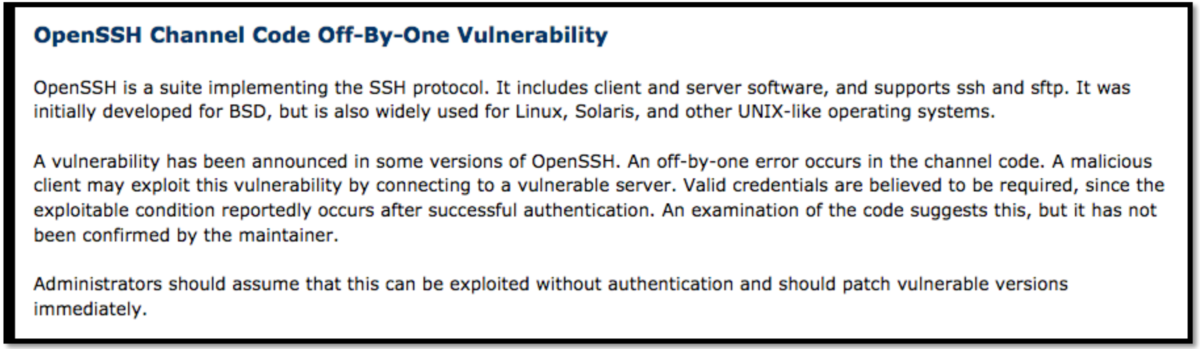


Vulnerability Details : CVE-2007-4987

Off-by-one error in the ReadBlobString function in blob.c in ImageMagick before 6.3.5-9 allows context-dependent attackers to execute arbitrary code via a crafted image file, which triggers the writing of a '\0' character to an out-of-bounds address.

Publish Date : 2007-09-24 Last Update Date : 2017-07-28

CVE Vulnerability Database



OpenSSH Channel Code Off-By-One Vulnerability

OpenSSH is a suite implementing the SSH protocol. It includes client and server software, and supports ssh and sftp. It was initially developed for BSD, but is also widely used for Linux, Solaris, and other UNIX-like operating systems.

A vulnerability has been announced in some versions of OpenSSH. An off-by-one error occurs in the channel code. A malicious client may exploit this vulnerability by connecting to a vulnerable server. Valid credentials are believed to be required, since the exploitable condition reportedly occurs after successful authentication. An examination of the code suggests this, but it has not been confirmed by the maintainer.

Administrators should assume that this can be exploited without authentication and should patch vulnerable versions immediately.

https://www.securityfocus.com/bid/4241/discuss

boundary value:

a value that is near **the extreme points** of an *ordinal* input domain or **at or around some special location**

What is ordinal?

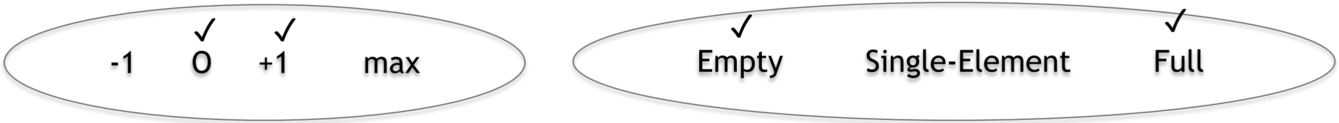
Rationale for testing boundary values

- Evidence
  - *Industrial, commercial, and defense software all note that faults seem to be more prevalent when variables have values at or near their boundaries*

- Application in input space modeling
  - *In partitioning a characteristic: boundary values may be represented as separate characteristics, or additional blocks in existing characteristics*
  - *In combining blocks: boundary blocks may be combined with each other or not*

If boundary values are important…

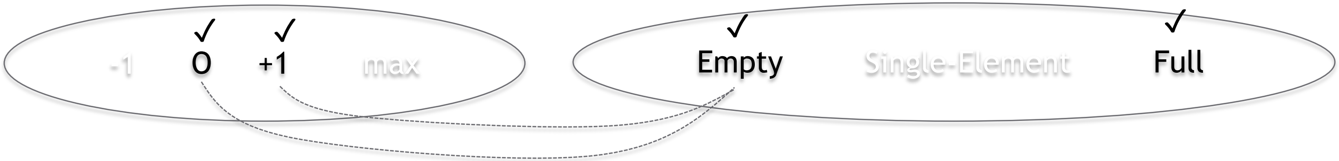
- Decide which ones are important in each characteristic, and include them as separate blocks in the input space model



-1 ✓ 0 ✓ +1 max

✓ Empty Single-Element ✓ Full

- Decide which combinations of boundary values from different characteristics are relevant

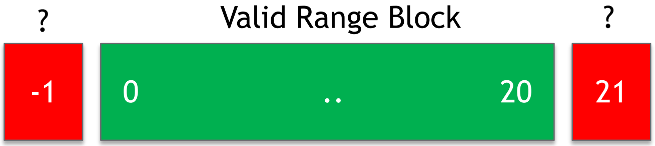


-1 ✓ 0 ✓ +1 max

✓ Empty Single-Element ✓ Full

Considerations for selecting boundary values

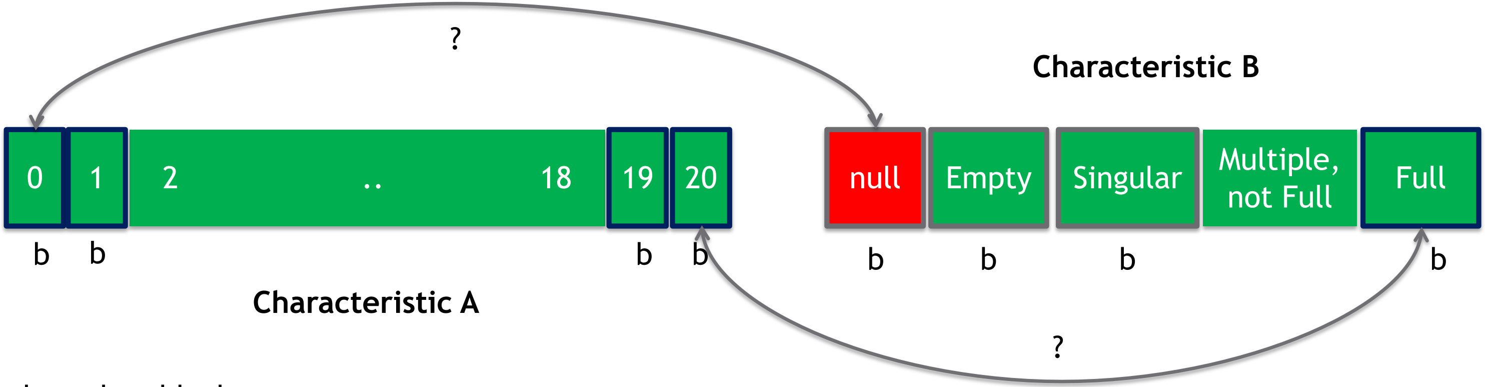
1. Invalid blocks: whether blocks just outside the valid range (beyond boundary blocks) are possible



- If blocks beyond the Valid Range are selected, we say "we **stress** the upper (lower) boundary" – this is called *robust* testing!
- Sometimes invalid blocks are not possible to convert to test cases because specifying input values for them is prohibited by the programming language, API, or UI

Considerations for selecting boundary values

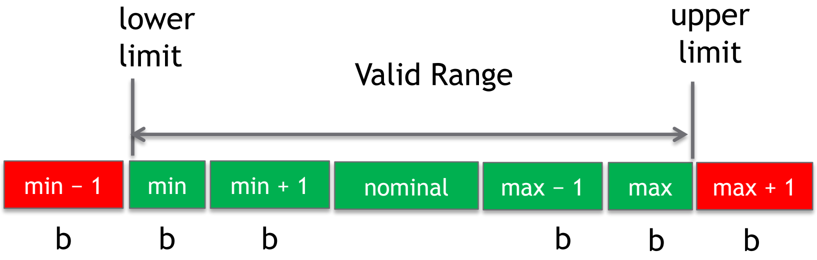
2. Interactions: whether faults can happen when boundary blocks are combined



b: boundary block

Which blocks represent boundary values?

Normally, for numeric domains

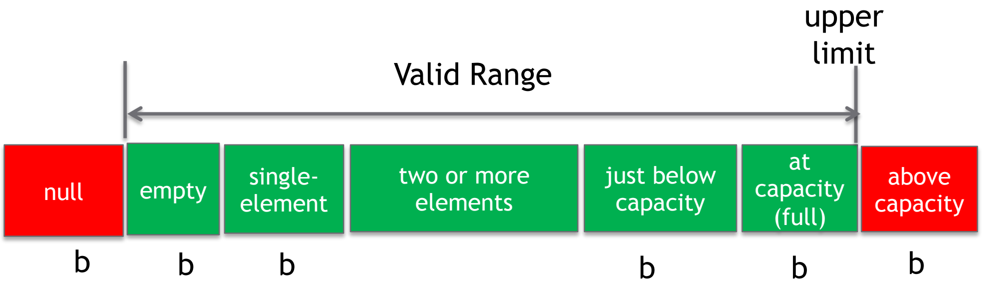


b: boundary block

“nominal”: used in the same sense as “ordinary”

Which blocks represent boundary values?

For non-numeric domains: objects, strings, and collections



upper limit

Valid Range

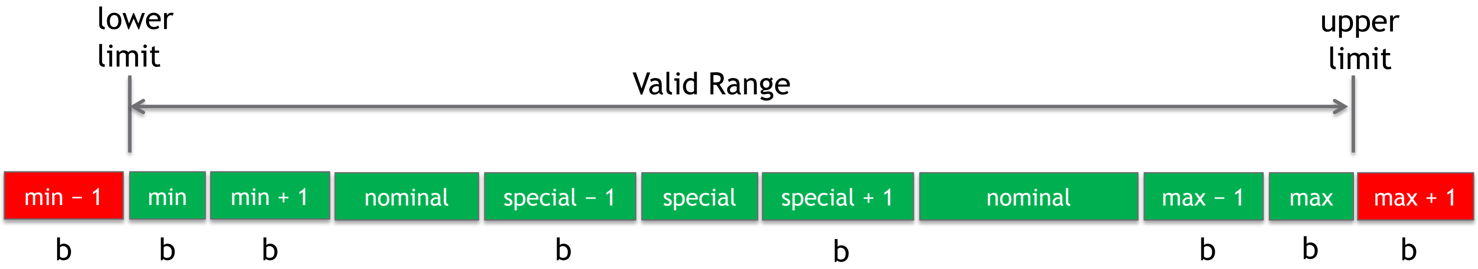
null | empty | single-element | two or more elements | just below capacity | at capacity (full) | above capacity

b b b b b b

b: boundary value

Which blocks represent boundary values?

We may also have some **special** values and values around them…



min − 1	min	min + 1	nominal	special − 1	special	special + 1	nominal	max − 1	max	max + 1
b	b	b		b		b		b	b	b

Special value example:

-1	0	+1

Decide which ones matter and should be included as a block?
And whether we can or want to stress *the boundary values…*

Boundary-value testing for NextDate: considerations

NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012. It returns the date of the next day.

- Questions
  - are boundary values relevant?
    - year? Yes
    - month? Yes
    - day? Yes
  - is there any good reasons for considering boundary value interactions?
    - Yes: some boundary values interact
      - day and month: last day of month
      - day, month, and year: last day of year (last month of year)
  - is there any good reasons for stressing boundaries?
    - Yes: invalid values are possible inputs

Interesting test cases for NextDate (without oracles)

Month	*Day*	*Year*	*Reason*
2	28	2012	Feb. 28 in a leap year
2	28	2013	Feb. 28 in a common year
2	29	2012	Leap day in a leap year
2	29	2000	Leap day in a century year
2	28	1900	Feb. 28 in a century year
12	31	2011	End of year
10	31	2012	End of 31-day month
11	30	2012	End of 30-day month
11	31	2012	Invalid date
2	29	1900	Invalid date (due to non-leap century year)

All boundary-value cases, with interactions

APPENDIX

Boundary Value Selection and Combinatorial Strategies

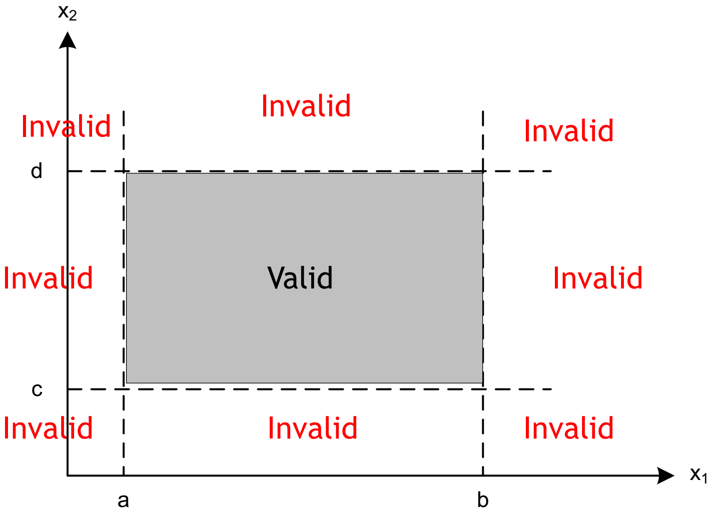
Considerations for boundary-value selection strategies

- Two considerations apply to boundary value choices
  - are invalid values an issue? are they possible? *Y/N*
  - do we accept “single-fault assumption” of reliability theory? *Y/N*

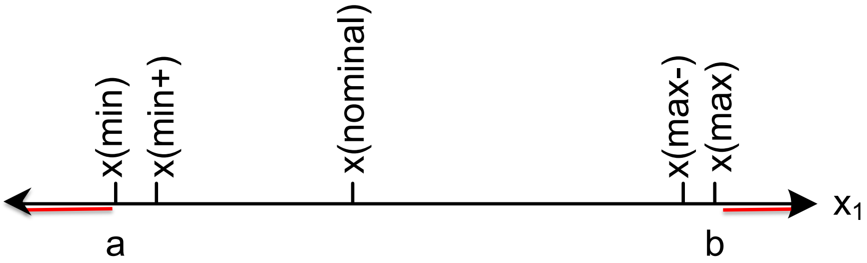
    “failures are only rarely the result of the simultaneous occurrence of two (or more) faults”
- Consequences
  - invalid values require the **robust** choice (testing outside valid ranges)
  - possibility of fault interaction requires **worst-case** testing (combining boundary values)
- Taken together, these yield four strategies (from weakest to strongest)
  - **Normal** boundary values (single-fault, valid values)
  - **Robust** boundary values (single-fault, invalid + valid values)
  - **Worst-case** boundary values (fault interaction, valid values)
  - **Robust worst-case** boundary values (fault interaction, invalid + valid values)

Input space of $F(x_1, x_2)$

where $a \leq x_1 \leq b$ and $c \leq x_2 \leq d$

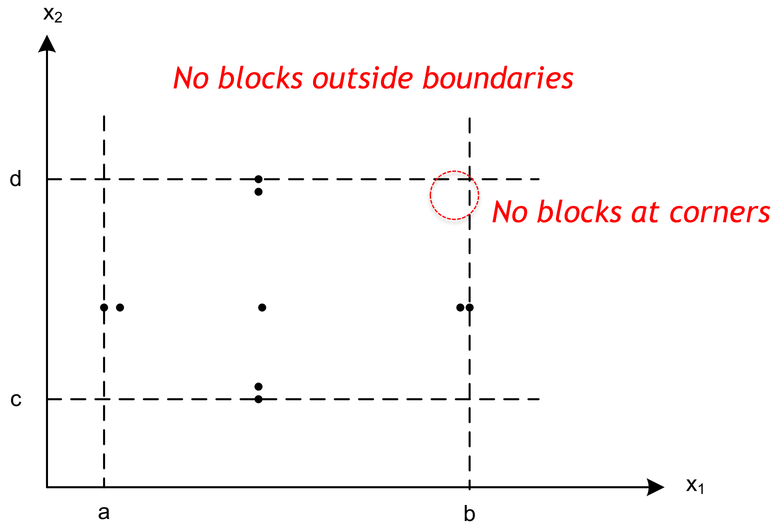


Normal test blocks for attribute x_1



Normal boundary-value test cases for $F(x_1, x_2)$

If “single-fault assumption” holds, two attributes rarely both assume their extreme values at the same time



9 cases

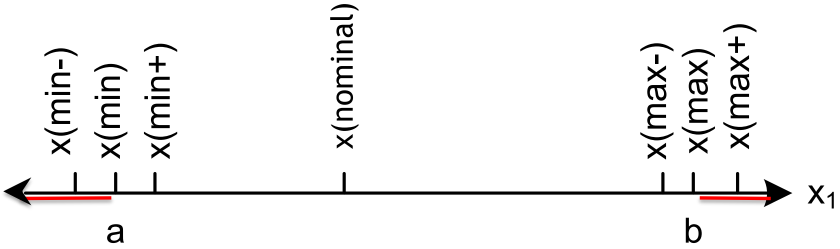
Method for normal boundary-value test cases

Similar to Base Choice (BC) with base cases = nominal blocks

- Hold all attributes at their nominal blocks
- Let one attribute assume its boundary blocks
- Repeat this for each attribute

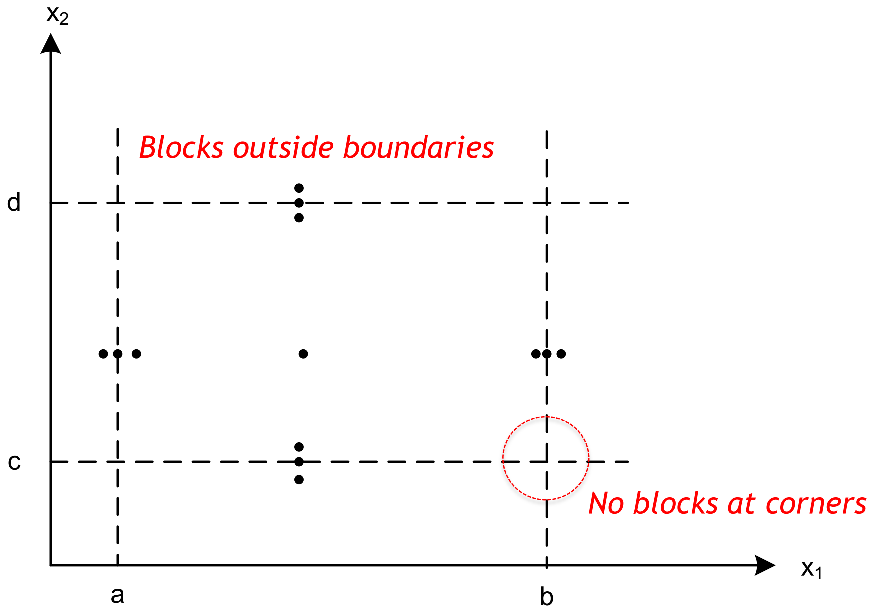
- This will (hopefully) reveal all faults that can be attributed to a single attribute

Robust test values for attribute x_1



- Stress boundaries
- Advantage
  - explore hidden functionality
  - focuses attention on exception handling
- But…
  - what are the expected outputs (oracle)?

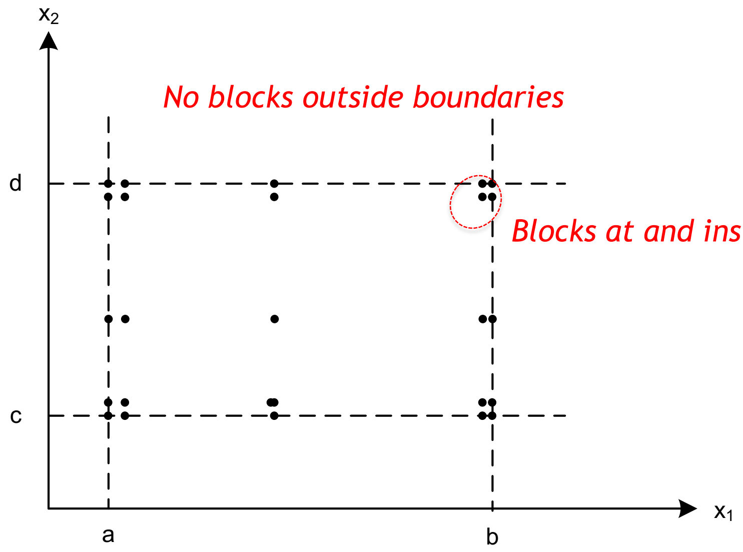
Robust boundary-value test cases for $F(x_1, x_2)$



13 cases

Worst-case boundary-value test cases for F(x_1, x_2)

Allows us to break the single-fault assumption...
We don't stress boundaries…



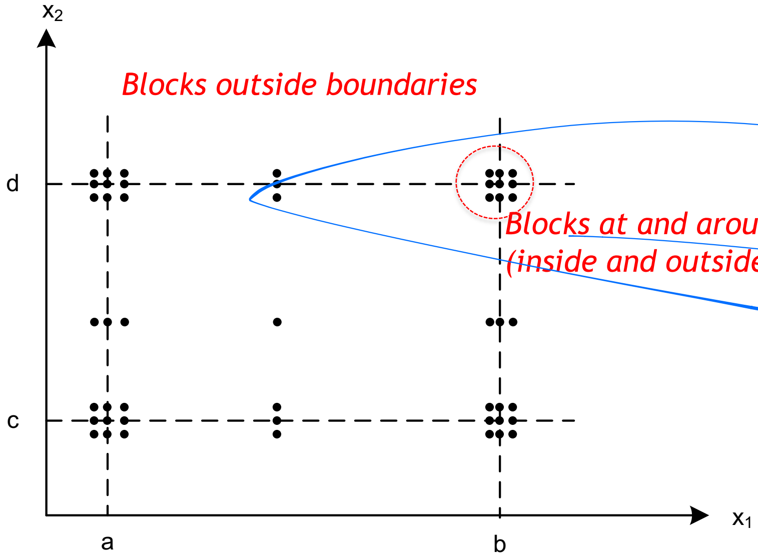
No blocks outside boundaries

Blocks at and inside-corners

25 cases

Robust worst-case boundary-value test cases for $F(x_1, x_2)$

Combines violation of single-fault assumption with stressing the boundaries…



Blocks outside boundaries

Blocks at and around corners (inside and outside)

49 cases

Example: NextDate function

- NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012
- It returns the date of the next day

NEXTDATE(Dec, 31, 1991) returns Jan 1 1992

NEXTDATE(Feb, 21, 1991) returns Feb 22 1991

NEXTDATE(Feb, 28, 1991) returns Mar 1 1991

NEXTDATE(Feb, 28, 1992) returns Feb 29 1992

Boundary-value testing for NextDate

NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012. It returns the date of the next day.

- Questions
  - are boundary values relevant?
    - year?
    - month?
    - day?
  - is there any good reasons for worst-case testing?
  - is there any good reasons for robustness testing?

Boundary-value testing for NextDate

NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012. It returns the date of the next day.

- Questions
  - are boundary values relevant?
    - year? Yes
    - month? Yes
    - day? Yes
  - is there any good reasons for worst-case testing?
    - Yes: attributes interact
  - is there any good reasons for robustness testing?
    - Yes: invalid values matter: robustness testing

Boundary-value testing for NextDate

NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012. It returns the date of the next day.

- Large number of test cases
  - 13 normal test cases (why?)
  - 125 worst-case test cases (why?)

Assuming both lower and upper boundary values relevant, let's ignore leap year, 31-day months

Normal: no interactions, no invalid blocks

Worst-case: interactions, no invalid blocks

Number of test cases for NextDate

NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012. It returns the date of the next day.

3 attributes with 4 boundary blocks each: (lower, lower + 1, upper - 1, upper)
Plus one nominal block for each attribute

combos for all nominal blocks

combos for all-but-one nominal block (two nominal, one boundary)

Normal test cases: 1 + 3 x 4 = 13 (Just like BC)

Worst-case test cases: 13 + 3 x (4 x 4) + (4 x 4 x 4) = 125 = 5 x 5 x 5

Combos for one nominal, two boundary

Combos for no nominal, all boundary

Boundary-value testing for NextDate: further questions

NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012. It returns the date of the next day.

- Questions
  - lower and upper boundary values
    - year: both equally relevant?
    - month: both equally relevant?
    - day: both equally relevant?

Boundary-value testing for NextDate: further questions

NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012. It returns the date of the next day.

- Questions
  - lower and upper boundary values
    - year: upper more relevant
    - month: upper more relevant
    - day: upper more relevant

Because it's next date, not previous date!

Boundary-value testing for NextDate: further questions

NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012. It returns the date of the next day.

- Question
  - is naïve boundary-value testing sufficient?

Boundary-value testing for NextDate: further questions

NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012. It returns the date of the next day.

- Question
  - is naïve boundary-value testing sufficient?
  - ***No*, because of 31-day months & leap years**

NextDate function: extra domain knowledge

Leap year

- Non-century year: must be divisible by 4
- Century year: must be divisible by 400
- Leap years include 1992, 1996, 2000, but not 1900

NEXTDATE(Feb, 28, 1992) returns Feb 29 1992

- max(day) **depends on** month & year
- Naïve (purely syntactic) boundary value testing without domain knowledge would miss wrong end-of-month and leap year calculations
- “31-day month” and “leap year” are *characteristics* that should be considered – extend # of boundary values for day
  - not fixed at 30, 31, 32, but…
    - subset of 27, 28, 29, 30, 31, 32 depending on month & year