TESTING PRINCIPLES AND PRACTISE

Testing Principles

- Testing shows presence of errors
  - Not their absence
- Exhaustive testing is not possible/practical
  - In most cases
- Test early and regularly
  - Early testing reduces multiple bug/defect relation and avoids bug masking
- Error clustering
  - Errors are not evenly distributed
  - Typically 20% of modules contain 80% of defects

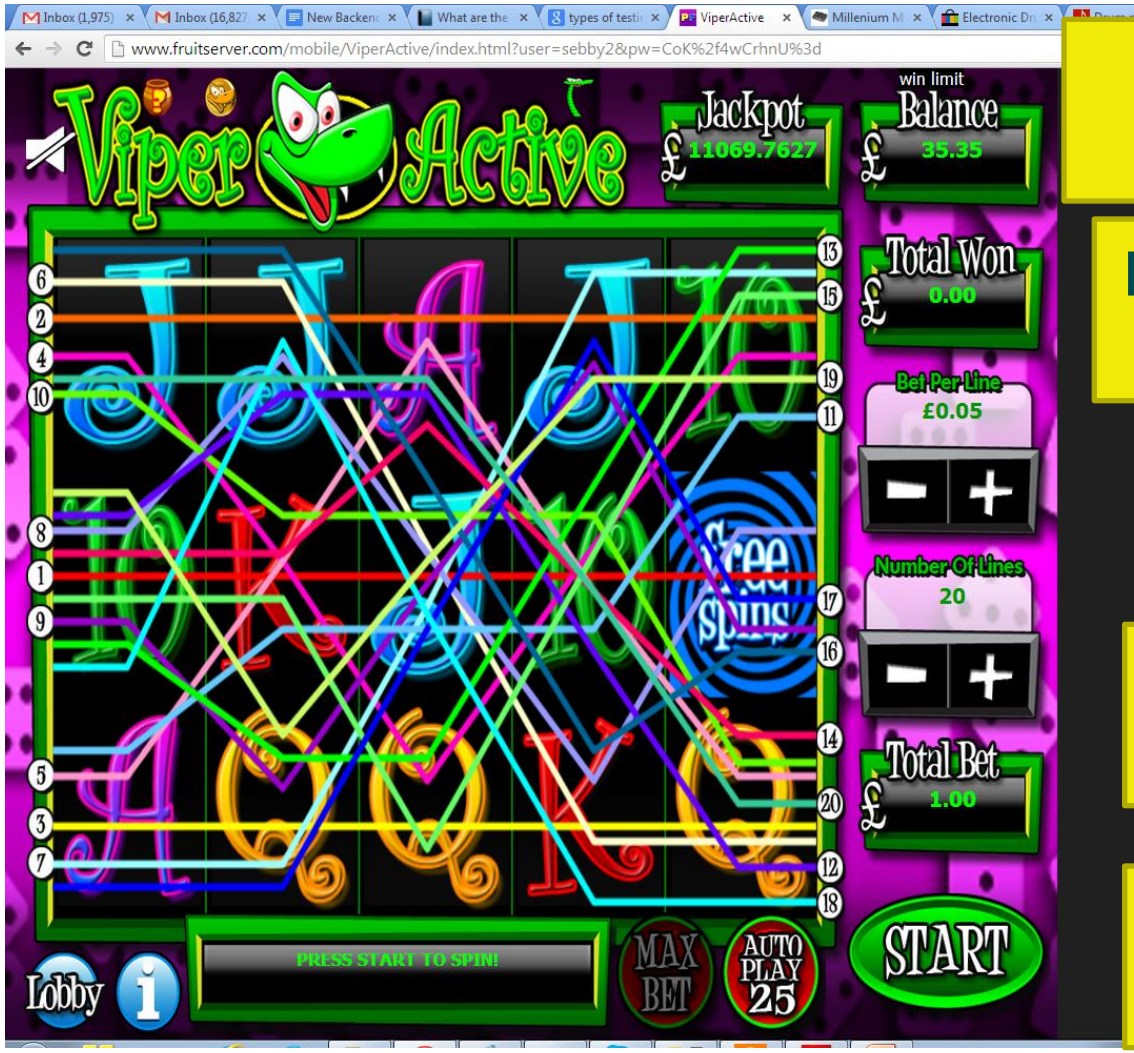
Testing Principles

- **Pesticide paradox**
  - **Unless tests change they often become invalid, as functionality changes so must tests**
- **Context dependency**
  - **Medical system → Safety testing**
  - **Website → Performance/load testing**
  - **Banking application → Security testing**
- **False conclusion: no errors equals usable system**

What is testing

- Determining the software conforms to the user's requirements
- Can include
  - Verification (against the spec.)
  - Validation (using customer or internal)
  - Performance
  - Security
  - Usability
  - Regulatory testing
  - Statistical testing

Case study.. game (how to test?)



- UI testing
- Performance testing
- Functional testing
- Regulatory testing
- Statistical testing

Win structure

Some Details

- Each reel has 256 symbols
- 5 Reels
- How many possible combinations?
  - (256)^5 = 1,099,511,627,776
  - 10,000 spins/second = 636 days
- Bigger problem
  - How to write 1,099,511,627,776 test cases?

Win structures and payout rules

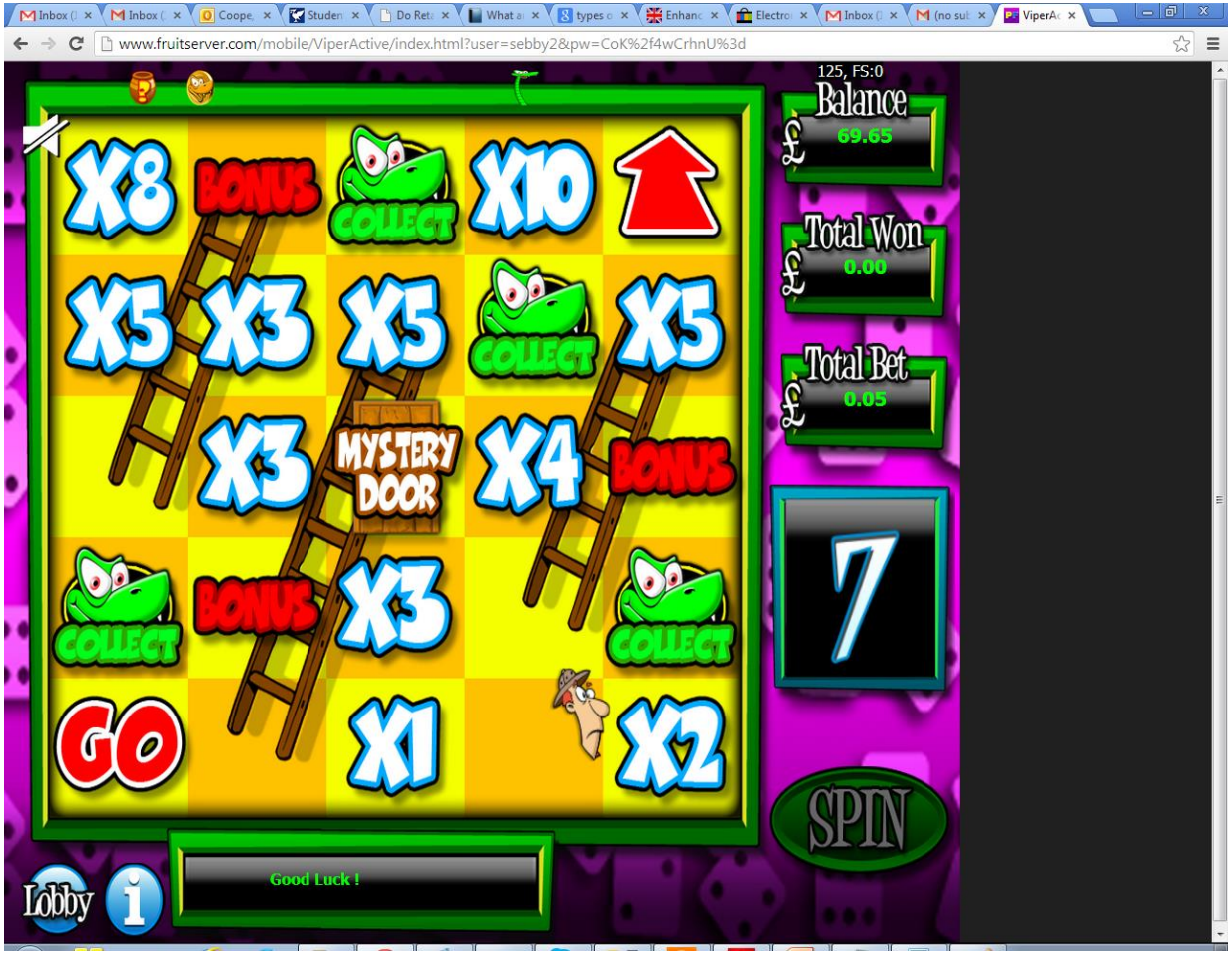
- TEN TEN TEN // 3 inline wins
  - WILDS substitute
  - So TEN WILD TEN pays a 3 x TEN win
- Four line wins
  - TEN TEN TEN TEN
  - TEN TEN WILD TEN
- How about
  - WILD WILD J TEN

Rules

- Wilds substitute for other symbols
- 20 pay lines, following from left to right
- 2, 3, 4, 5 in line possible wins
- Stake can be changed for each spin

Special features

- Some wins give a game within a game, this is called a feature…



Scattered symbols

- Some symbols appear anywhere in the window, so below generates 4 symbol feature win.

How many Combinations give 4 snakes?

Exercise

- How many combinations give 4 snakes?
- What combinations give 4 snakes?
- Can we test them all?
- Use software to generate test cases?
  - Write a small Java program to generate these test cases

Feature wins

- Generate a range of random prizes
- Testing issues
  - Distribution of prizes
  - Average prize distribution
  - Maximum prize
  - Minimum prize

Target code

- int getWinValue(int stake,int symbols[][])
- stake is amount of money awarded per payline
- stake is in pence/cents
- Symbols
  - 5 x 3 array displaying symbols in window
  - 1= TEN, 2 = J, 3 = Q

Testing approach

- At least orthogonal
- So each range of data but not every combination of each range
- Each win type including/excluding wilds
- For a given win, choose a range of stakes
  - Ensure the multiplier works
  - Try on each different payline

Orthogonal testing and modes

- Function
  - countDaysTill(int day,int month,int year)
- Bug might appear in test case for particular
  - day, month, year (triple mode fault)
  - Year (single mode fault)
  - day, month (double mode fault)
- Exhaustive testing of some single mode faults is sometimes possible

Orthogonal array testing

- Imagine you have a method with 3 integer arguments
- Arg1 range 1-10
- Arg2 range 2-20
- Arg3 range 5,6 or 7
- You want to catch all single mode errors
  - How many tests need?
  - What tests?

Orthogonal array testing

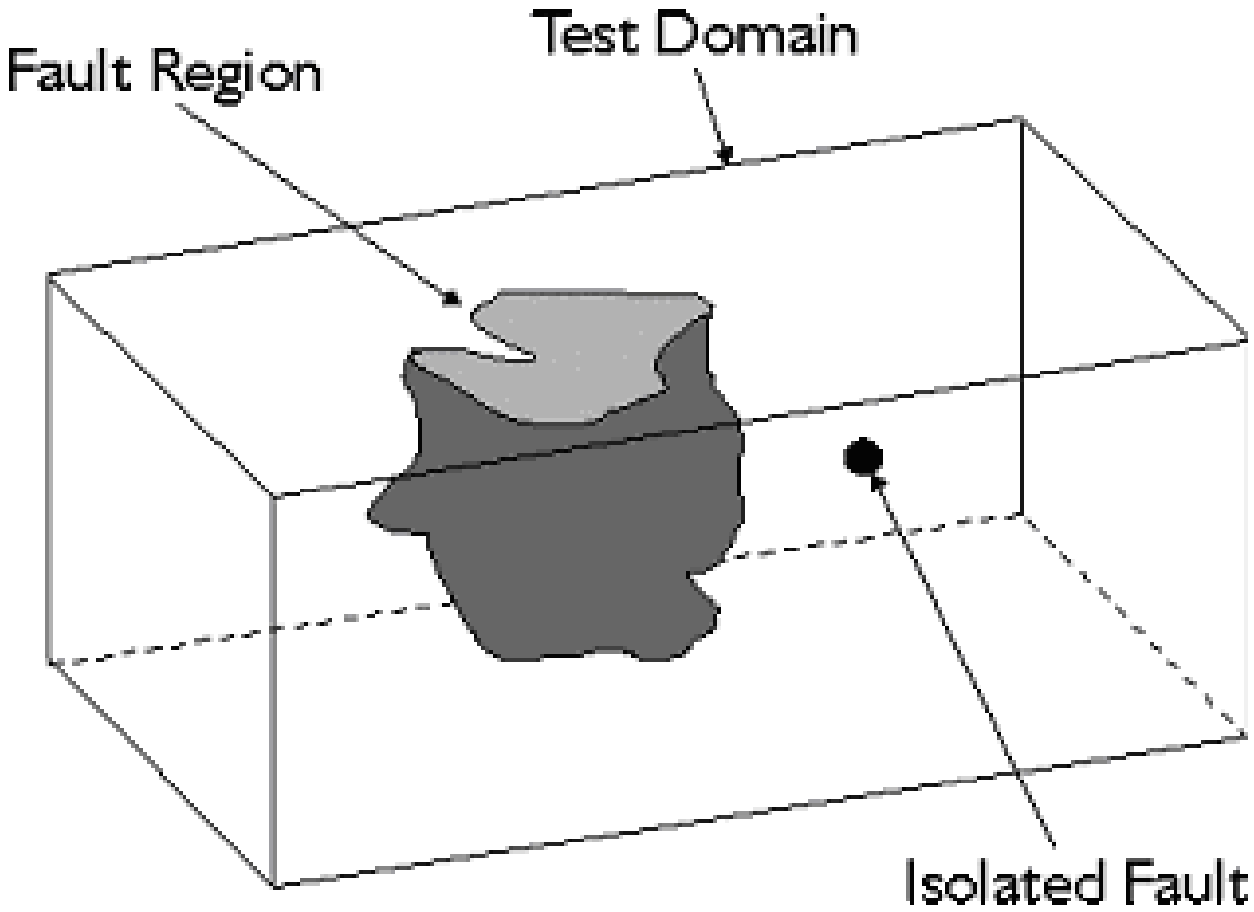
- Total test count (single mode)
  - 10 for arg1 + 19 for arg2 + 3 for arg3
  - = 10+19+3 = 32 tests (actually only 30, some are redundant)
- Single mode errors
  - Number of tests is order of N (where N is argument levels)
- Triple mode errors
  - Number of tests is order of N^3
  - In the case above 570 tests (10x19x3)

Orthogonal array testing

- Tests for single mode

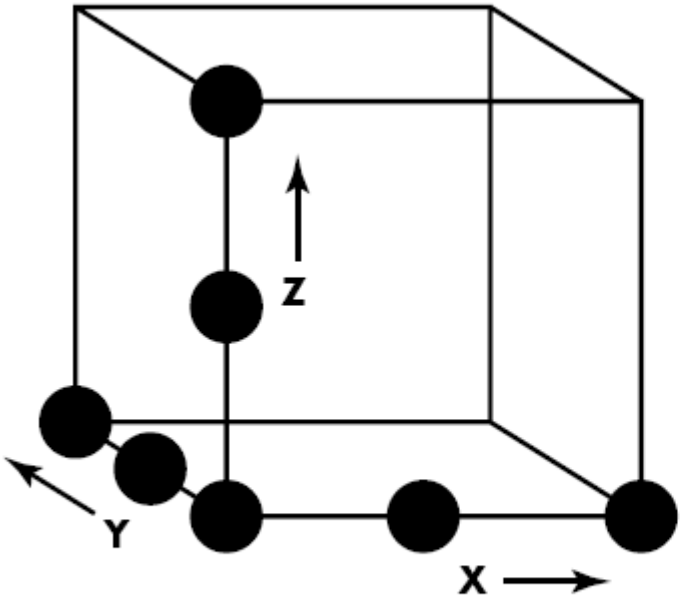
Test	Arg1	Arg2	Arg3
1	1	2	5
2	2	2	5
3	3	2	5
4	4	2	5
5	5	2	5
6	6	2	5
7	7	2	5
8	8	2	5
9	9	2	5
10	10	2	5
11	1	3	5
12	1	4	5
13	1	5	5
14	1	6	5
15	1	7	5
16	1	8	5
17	1	9	5
18	1	10	5
19	1	11	5
20	1	12	5
21	1	13	5
22	1	14	5
23	1	15	5
24	1	16	5
25	1	17	5
26	1	18	5
27	1	19	5
28	1	20	5
29	1	20	6
30	1	20	7

Defect density, clustered or isolated

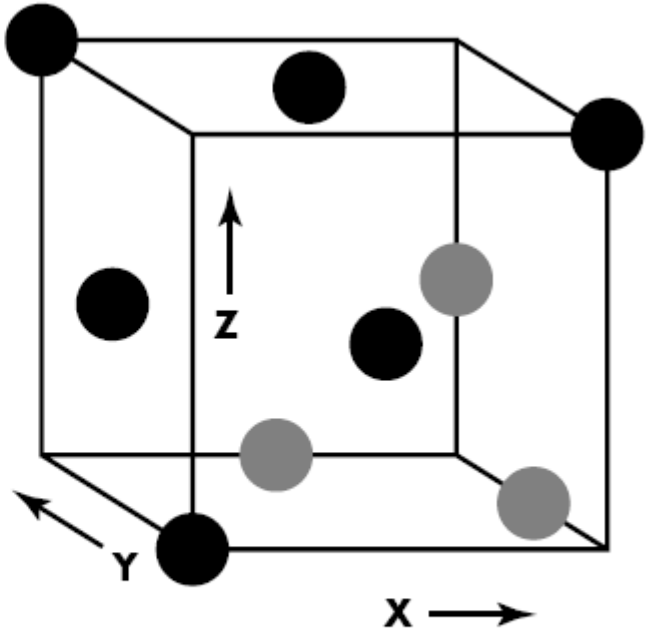


In practise

- Most recommendations is for at least all pair-wise combinations (this is what most people will call orthogonal array test)
- Use a tool to generate test array, then complete by hand
- My generating the “other” values randomly for each test, you distribute your tests around the state space



One input item at a time



L9 orthogonal array

Orthogonal testing for our game

- So if stake can be
  - 10p,50p,100p,200p,500p
- Single mode test for stake could use the following test cases

STAKE	REEL1	REEL 2	REEL 3	REEL 4	REEL 5	Output
10	TEN	TEN	TEN	JACK	JACK	30
50	TEN	TEN	TEN	JACK	JACK	150
100	TEN	TEN	TEN	JACK	JACK	300
200	TEN	TEN	TEN	JACK	JACK	600
500	TEN	TEN	TEN	JACK	JACK	1500

Orthogonal testing

- Now test 3 in line TEN wins (keep stake constant)

STAKE	REEL1	REEL 2	REEL 3	REEL 4	REEL 5	Output
10	TEN	TEN	TEN	JACK	JACK	30
10	WILD	TEN	TEN	JACK	JACK	30
10	TEN	WILD	TEN	JACK	JACK	30
10	TEN	TEN	WILD	JACK	JACK	30
10	WILD	WILD	TEN	JACK	JACK	30
10	WILD	TEN	WILD	JACK	JACK	30
10	TEN	WILD	WILD	JACK	JACK	30

More testing of the machine

- Do we/Can we testing all losing lines?
  - Alternatives?
    - Statistics
- Random test generation
  - Rand = random (but no Wild, no Ten)

STAKE	REEL1	REEL 2	REEL 3	REEL 4	REEL 5	Output
10	TEN	TEN	TEN	Rand	Rand	30
10	WILD	TEN	TEN	Rand	Rand	30
10	TEN	WILD	TEN	Rand	Rand	30
10	TEN	TEN	WILD	Rand	Rand	30
10	WILD	WILD	TEN	Rand	Rand	30
10	WILD	TEN	WILD	Rand	Rand	30
10	TEN	WILD	WILD	Rand	Rand	30

Adding in random numbers

- Teases out rarer bugs
- Can do different tests across Q&A phases
- Increases assurance
- Can be used to make statistical inference

Testing calculations

- If bug effects 0.01% of positions, evenly distributed and we test 10,000 positions randomly, what is the chance of a false positive
  - Chance of test giving false positive
  - Chance of passing bug for each test is
    - 1 – (0.0001) = 0.9999
    - Chance of not finding bug is
      - (0.9999)^10,000 = 0.36 or 36%

Testing calculations

- Assuming confidence of 1% (i.e. assuming we might be wrong 1% of time)
- If we do 1,000,000 random tests what is maximum error density
- (1-chance_of_bug)^10,000,000<0.01
- So
- 10,000,000log(1-chance_of_bug)<log(0.01)
- Log(1-chance_bug)<log(0.01)/10,000,000
- 1-chance_bug<10^(log(0.01)/10,000,000)
- chance_bug=1-10^(log(0.01)/10,000,000)
- 0.0000046 defect density

Testing levels

- At unit level
  - Confirming the following all function correctly
    - All methods provide primary function
    - All constructors and methods validate input
    - Exceptions are caught or thrown as required
  - Coding by contract
    - Each method has a required range of appropriate values
    - Use of methods should conform with contract

Test Example

- class Person.java
- Constructor takes name and Date of birth
- Has methods
  - int getAgeInYears() // whole number of years old
  - boolean isAdult() // true if Person is adult

- Is meant to provide validation checking of its attributes

Testing example code (boundary cases)

```
private void testYears() {
        Calendar dobCal = Calendar.getInstance(); // set up current
time
        dobCal.setTimeInMillis(System.currentTimeMillis()); // set
up time
        int testAge=18;
        dobCal.add(Calendar.YEAR,-testAge);
        setDateOfBirth(dobCal.getTime());
        if (getAgeInYears()!=testAge) {
                throw new TestFailException("Failed Age test
Birthday today");
        }
```

Fundamental problem with test code

- Code is being written to test code
- So if a bug happens
- Is the bug in
  - The target code
  - The test code
- Answers
  - Have 2 teams (test code team, testing team)
  - Use simple identities sqr(x) x sqr(x) = x

Testing Example

```
dobCal.add(Calendar.DAY_OF_MONTH, 1); // move to tommorow
        setDateOfBirth(dobCal.getTime());
        if (getAgeInYears()!=testAge-1) {
                throw new TestFailException("Failed Age test Birthday tommorow");
        }
```

Test Example

```
dobCal.add(Calendar.DAY_OF_MONTH, -2); // move to yesterday
setDateOfBirth(dobCal.getTime());
if (getAgeInYears()!=testAge) {
        throw new TestFailException("Failed Age test Birthday
yesterday");
```

Testing and debugging

- Bug report
  - Date, product name, platform (Windows, Linux, Chrome IE?)
  - Description
  - Logs (? Java console dump?), server trace log
  - Version of software (see svn version)
  - How to re-create the bug

Server logs (web debugging)

[10:07:23 AM] Cometa.Core.Server.CommandsNew.CommandGameInit.Execute. Begin...
,Arcadia 0.013215312 0.000165
[10:07:23 AM] Cometa.Core.Server.CommandsNew.CommandBase.AuthenticatePlayer. Begin...
,Arcadia 0.013653376 0.000438
[10:07:23 AM] Cometa.Core.Server.CommandsNew.AuthenticatePlayer. Loading user with login of sebby2
,Arcadia 0.01406112 0.000408
[10:07:23 AM] ExecuteDataset: Select * from $(user) Where LCASE(Login)='sebby2'
,Arcadia 0.014521312 0.000460
[10:07:23 AM] Cometa.Core.Server.CommandsNew.AuthenticatePlayer. User loaded okay, is valid, is in user role, is not blocked.
,Arcadia 0.019047712 0.004526
[10:07:23 AM] Cometa.Core.Server.CommandsNew.AuthenticatePlayer. Password matched.
,Arcadia 0.019882992 0.000835
[10:07:23 AM] ResponseMessageManager::GetMessageByShortCode: Begin...
,Arcadia

Validation testing (error case checking)

```
Calendar dobCal = Calendar.getInstance(); // set up current time
        dobCal.setTimeInMillis(System.currentTimeMillis()); // set
up time
        dobCal.add(Calendar.DAY,1); // move to tomorrow
Person person=new Person();
Boolean testFailed=true;
try {
   person.setDateOfBirth(badDate);
} catch (Exception) {
     testFailed=false;
}
```

This could equally be done and should be done with constructor

```
Person p=new Person(“test”,”test”,”test”,”test”,”test”,badDate);
```

Need for automation

- Many tests
- Requirement for continuous regression testing
- Rapid implementation of new functions
- Less human error

- Cost
  - High up front cost in time/money

Summary

- Hard to test all cases
- Automation is essential to make testing effective
- Orthogonal approach can be worth while to get coverage of modal bugs