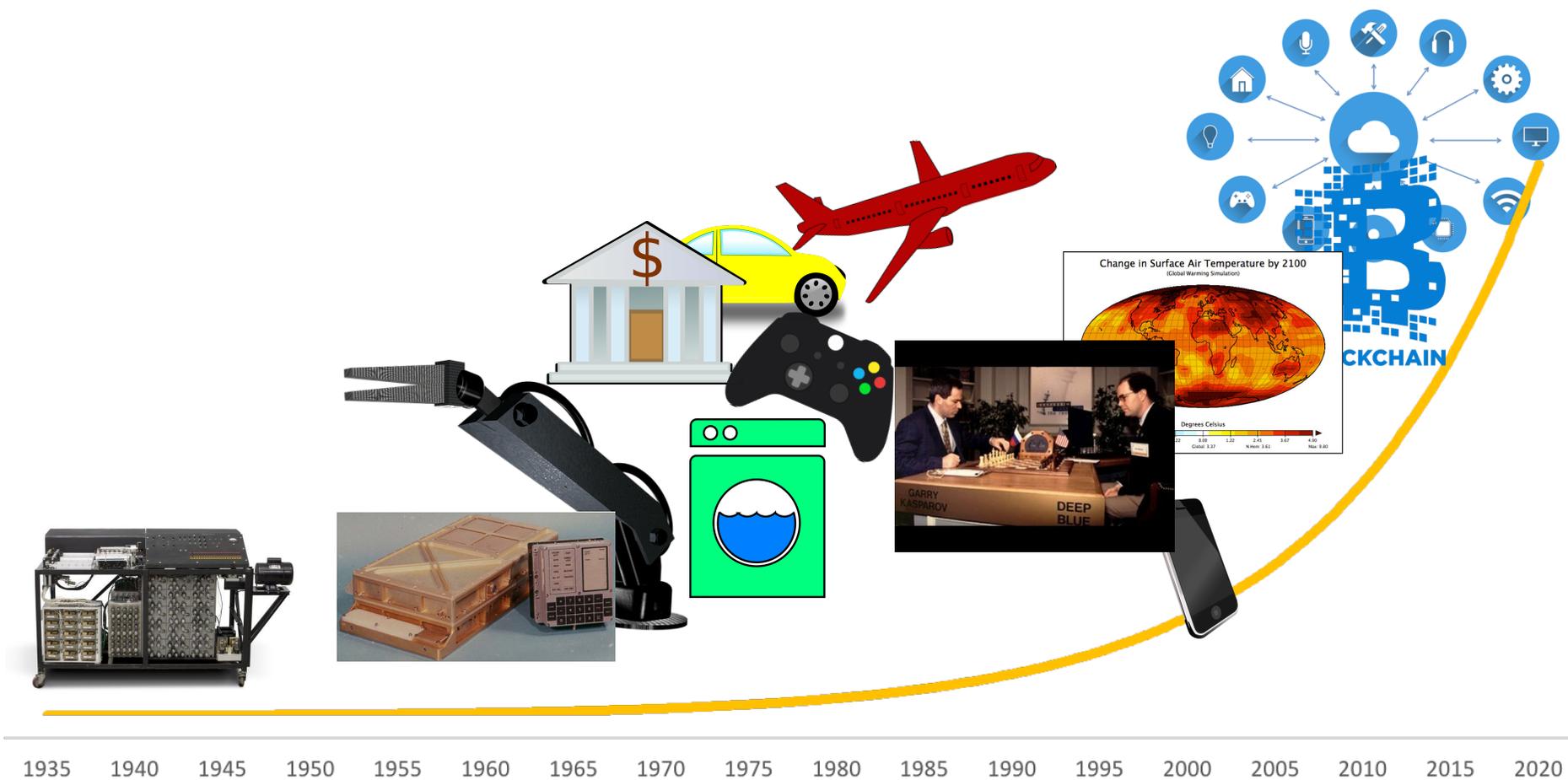


More Effective and More Maintainable Test Suites, with Combinatorial Testing

Razieh Behjati

Senior test automation engineer

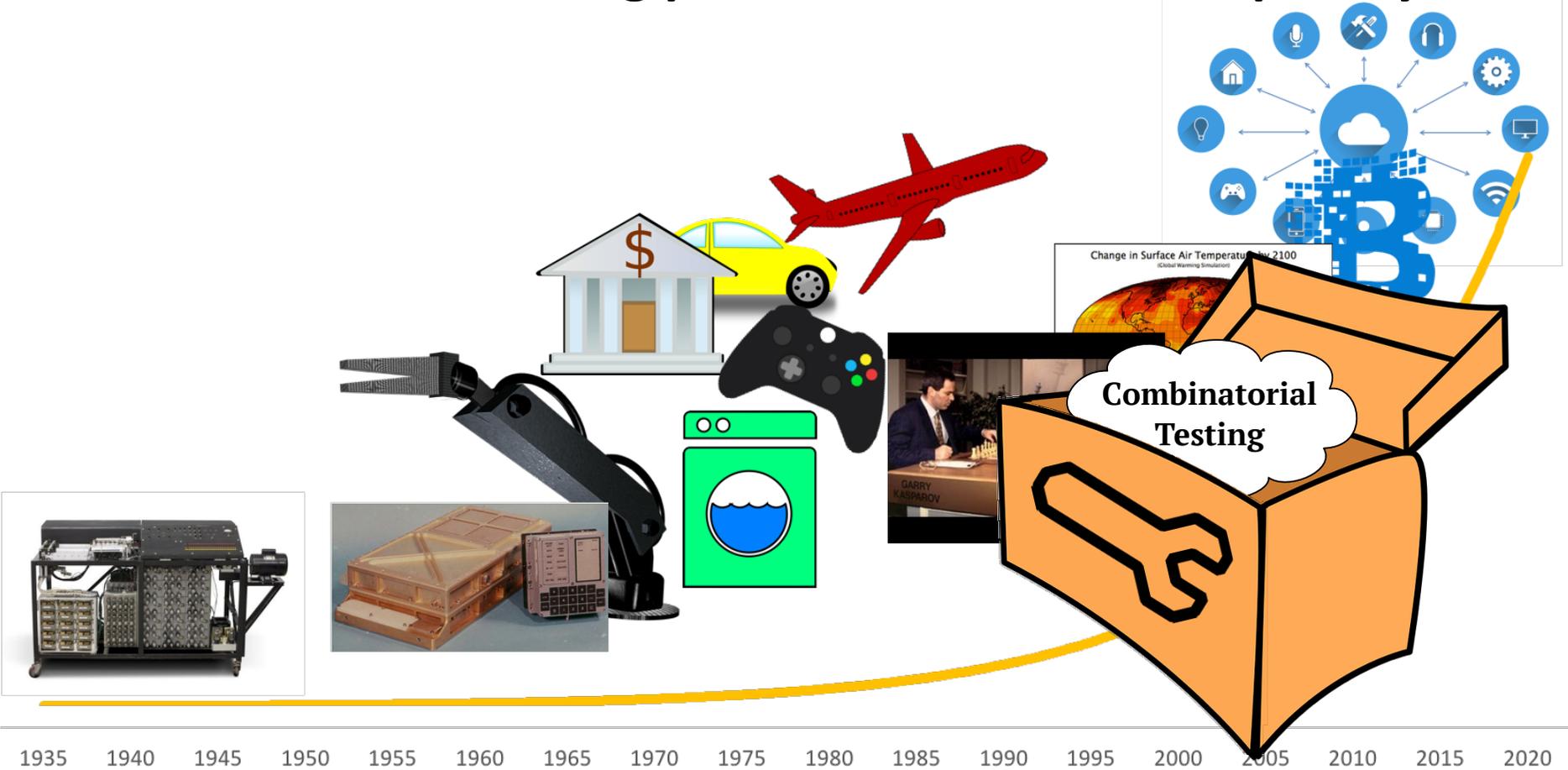




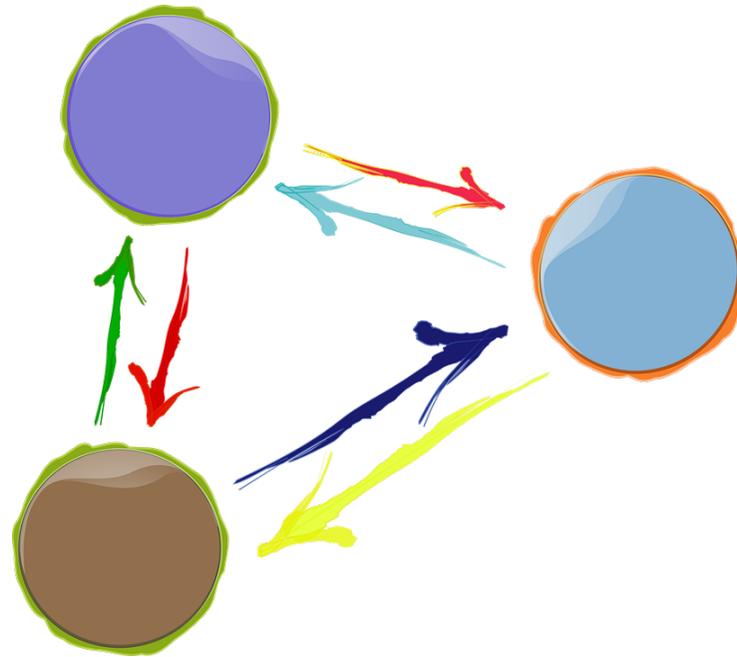
1935 1940 1945 1950 1955 1960 1965 1970 1975 1980 1985 1990 1995 2000 2005 2010 2015 2020



We need tools of matching power to test such complex systems



Combinatorial testing focuses on detecting *interaction failures*



Combinatorial testing focuses on detecting *interaction failures*

`/search?feild1=val1&field2<val2`

`/search?gender=female&age<18`

CUT

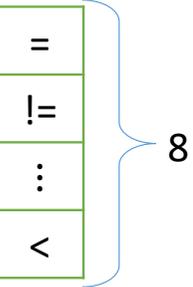


```

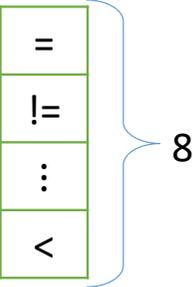
{
  "query": {
    "bool": {
      "must": [
        {
          "term": {
            "field1": "value1"
          }
        },
        {
          "range": {
            "field2": {
              "lte": "value2"
            }
          }
        }
      ]
    },
    "minimum_should_match": 1,
    "boost": 1
  }
}
    
```

Subquery-1	
field1	operator1
<div style="border: 1px solid black; padding: 2px; width: fit-content; margin: 2px;">age</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin: 2px;">gender</div>	=
	!=
	:
	<

factor



Subquery-2	
field2	operator2
<div style="border: 1px solid black; padding: 2px; width: fit-content; margin: 2px;">age</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin: 2px;">gender</div>	=
	!=
	:
	<



gender	=	age	<
--------	---	-----	---

combination

256 combinations!



Found two new bugs...

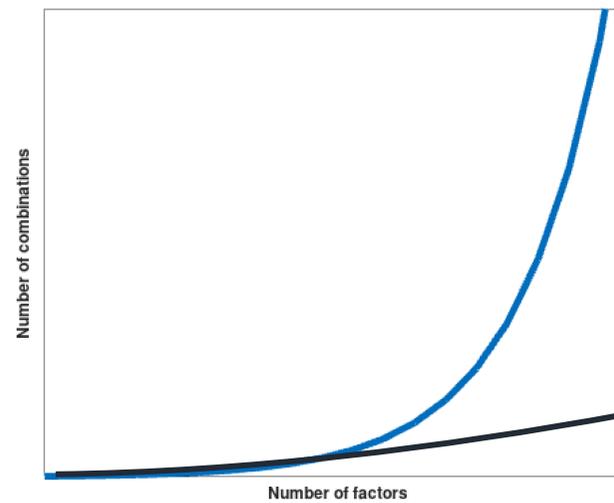
age	exists	age	=
-----	--------	-----	---

Domain coverage vs. code coverage



Exponential growth problem

Model	# Bugs found	# Combinations	Test exec. time	Test Suite exec. time
$2 \times 2 \times 8 \times 8$	2	256	50 + 445 ms	3.6 s



t-way combinatorial testing gives a practical way to detect interaction failures

	field1	op.1	field2	op.2
1	age	=	age	=
2	age	=	age	!=
...				
256	gender	<	gender	<

- All 2-way interactions:

- field1, op.1: 2×8
- field1, field2: 2×2
- field1, op.2: 2×8
- op.1, field2 : 8×2
- op.1, op.2: 8×8
- field2, op.2: 2×8

132 pairs to cover!

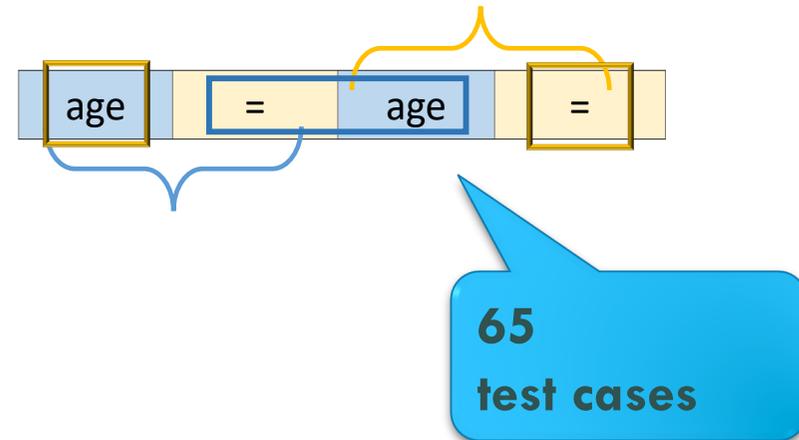


t-way combinatorial testing gives a practical way to detect interaction failures

	field1	op.1	field2	op.2
1	age	=	age	=
2	age	=	age	!=
...				
256	gender	<	gender	<

- All 2-way interactions:

- field1, op.1: 2×8
- field1, field2: 2×2
- field1, op.2: 2×8
- op.1, field2 : 8×2
- op.1, op.2: 8×8
- field2, op.2: 2×8



132 pairs to cover!



An NP-Hard problem

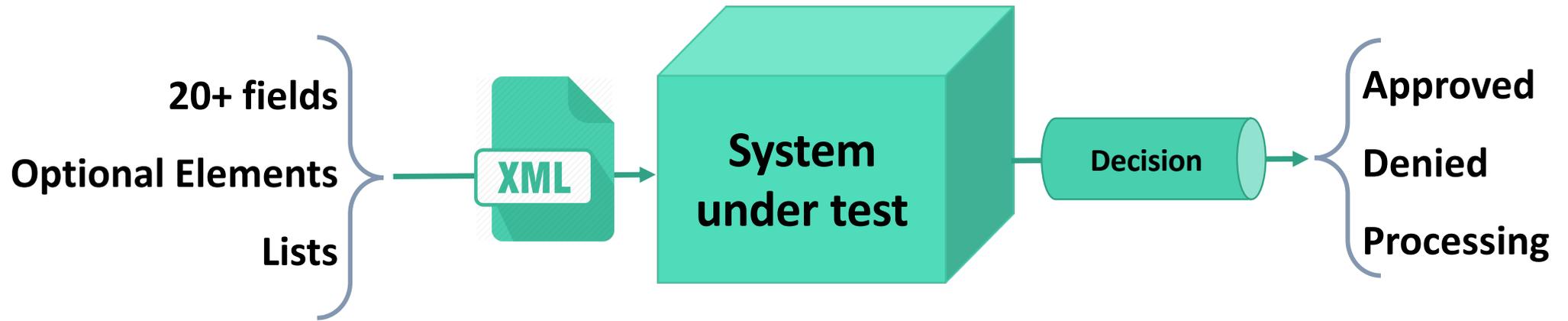


Suitable for testing larger systems

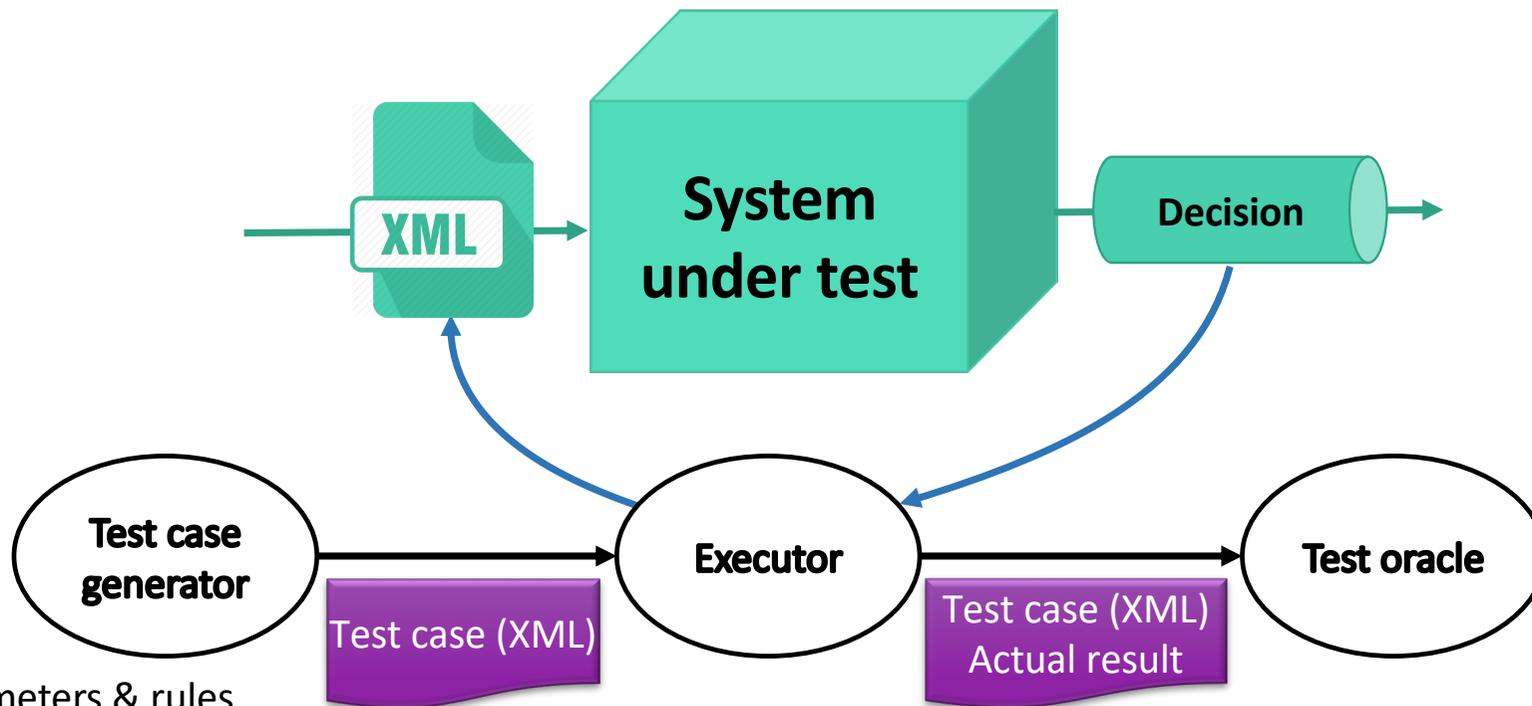
Use a top-down approach



End-to-end testing with combinatorial test case generator



End-to-end testing with combinatorial test case generator

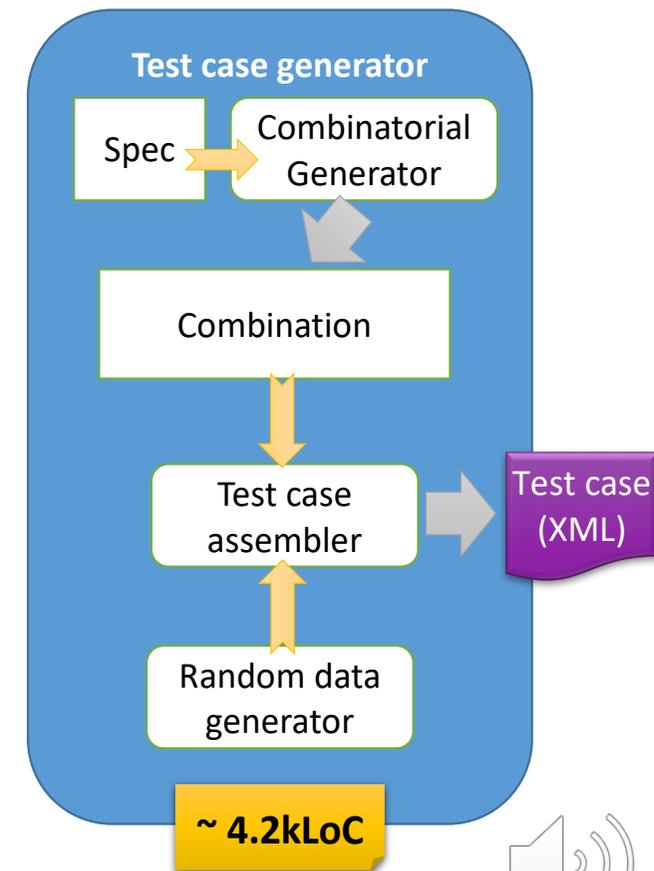
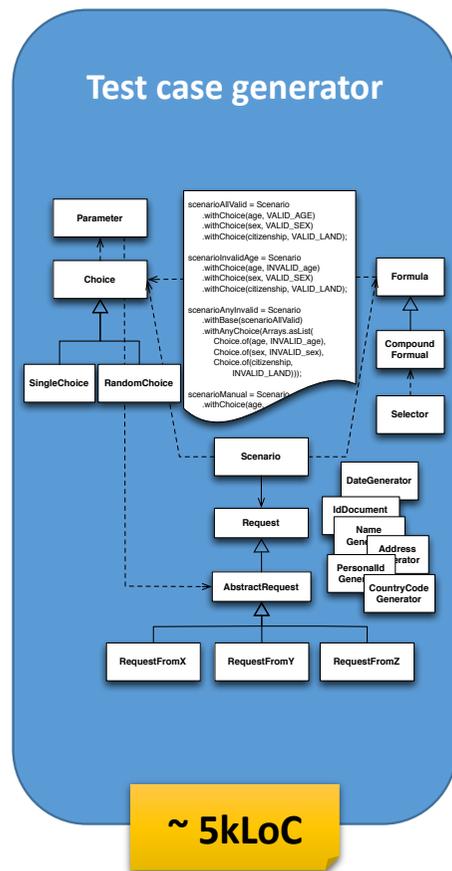


Complexity:

- Many parameters & rules
- Changing rules
- Cope with the dynamics of the environment
- Uniqueness constraints



End-to-end testing with a combinatorial test case generator



Was the use of combinatorial testing effective?

A particularly interesting bug



Tuning the test case generator

- Model
 - 11 parameters (factors)
 - Each 2-5 equivalence classes
 - 6 constraints
- Interaction strength
 - 3-way interactions

Approval	13
Denial	193
Processing	19
Total	225



Use mixed-strength interactions for a better scenario distribution

- Model
 - 11 parameters (factors)
 - Each 2-5 equivalence classes
 - 6 constraints
- Interaction strength
 - 3-way interactions

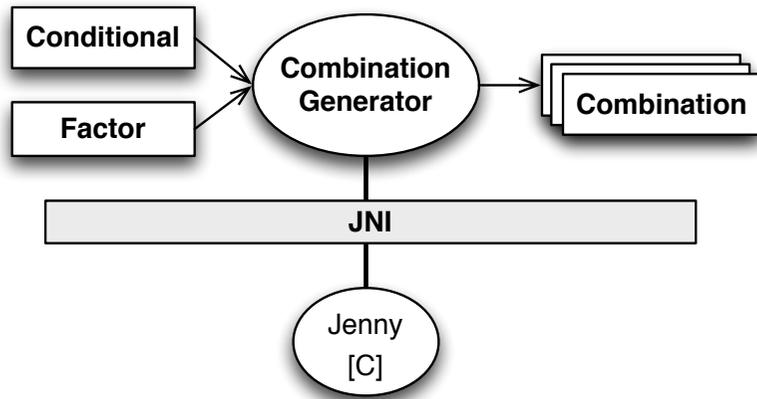
Approval	13
Denial	193
Processing	19
Total	225

- Added more test cases covering 4-way or 5-way interactions

Approval	76
Denial	1126
Processing	87
Total	1289

Approval	76
Denial	193
Processing	87
Total	356





<http://burtleburtle.net/bob/math/jenny.html>

```

<dependency>
  <groupId>io.github.rbehjati</groupId>
  <artifactId>genie</artifactId>
  <version>0.1.0</version>
</dependency>
  
```

<https://github.com/rbehjati/genie>

```

In [1]: %classpath add jar ./resources/genie-1.1.0.jar

In [2]: import io.rbehjati.genie.CombinationGenerator;
import io.rbehjati.genie.model.Combination;
import io.rbehjati.genie.model.Feature;

import java.util.Arrays;
import java.util.List;
CombinationGenerator generator = new CombinationGenerator();

Feature feature1 = new Feature("F1", "V1", "V2");
Feature feature2 = new Feature("F2", "V3", "V4");

List<Combination> combinations = generator.generateCombinations(2, Arrays.asList(feature1, feature2));

System.out.println("Total numebr of combinations: " + combinations.size());

Total numebr of combinations: 4
  
```

Thanks!

