



The Good, the Bad and the Ugly solutions in test design

Attila Kovács





Suppose you are working at the best organized company producing software

T_D



Planning



Scheduling



TEST AUTOMATION



Test TOOL

COLLABORATION



documentation of requirements

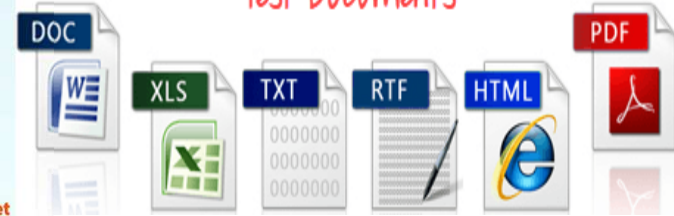
testing of requirements

tracking of requirements

prioritizing of requirements



Test Documents



TD



... but you still have quality problems ...



„I have checked every square foot in this house. I can confidently say that there are no mice here...”

... because of inefficient test case selection



Proper test case selection must be more than just „paid fishing”. We need better test design strategies



What is test design?

A test design technique is a **procedure for determining test conditions, test cases and test data during software testing**



ISO/IEC 29119 - 4



5	Test design techniques
5.1	Overview.....
5.2	Specification-based test design techniques.....
5.2.1	Equivalence partitioning.....
5.2.2	Classification tree method.....
5.2.3	Boundary value analysis.....
5.2.4	Syntax testing.....
5.2.5	Combinatorial test design techniques.....
5.2.6	Decision table testing.....
5.2.7	Cause-effect graphing.....
5.2.8	State transition testing.....
5.2.9	Scenario testing.....
5.2.10	Random testing.....
5.2.11	Metamorphic testing.....
5.2.12	Requirements-based testing.....
5.3	Structure-based test design techniques.....
5.3.1	Statement testing.....
5.3.2	Branch testing.....
5.3.3	Decision testing.....
5.3.4	Branch condition testing.....
5.3.5	Branch condition combination testing.....
5.3.6	Modified condition/decision coverage (MCDC) testing.....
5.3.7	Data flow testing.....
5.4	Experience-based test design techniques.....
5.4.1	Error guessing.....

How do we choose the BEST technique?

$$\Delta u = -V_{eq} \ln(M) \Big|_{m_0}^{m_f}$$

$$2 + 3x \quad dx$$

$$\sum_{i=1}^{100} i = \frac{n(n+1)}{2} = \frac{1000 + 1001}{2} =$$

$$PV = nRT$$

$$\omega = 2\pi f$$

$$\iint dx$$



i



$$\sum_{i=1}^{100} i \quad 3x^2 \quad ds$$

Newton's second law of motion:

$$\frac{500}{2} = F = V_{eq} \frac{dmp}{dt}$$

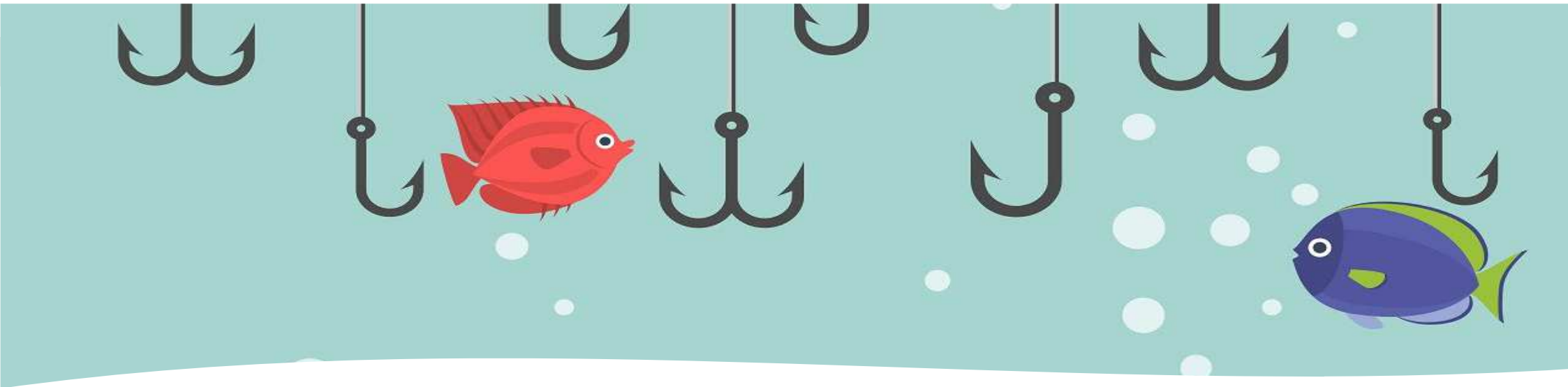
$$M^2 du + u dM = V_{eq} dmp$$

$$M du = -V_{eq} dM$$

$$du = -V_{eq} \frac{dM}{M}$$

Assume we have rocket





The problem: The traditional test design techniques are coverage-based, testers want to „catch more fish”

The traditional answer: Raise the coverage, „use more fishing rods”

T_D



However, raising the coverage is not enough...
Better techniques, different approaches are
needed (e.g. better fishing baits)



Example: Rollercoaster



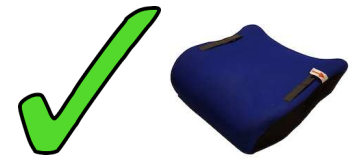
Scenario: We have to test the people entry and the round-release of a Rollercoaster. At the entrance the height, the weight and the free seat values are checked, and the entry is decided. The height (integer) values are given in cm, the total weight (integer) values in kg.

Requirement 1



The height of people is split into 3 categories:

- People with less than 120 cm are not allowed to enter,
- People from 120 to 140 cm are allowed to enter with a seat-extender,
- People taller than 140 cm are allowed to enter



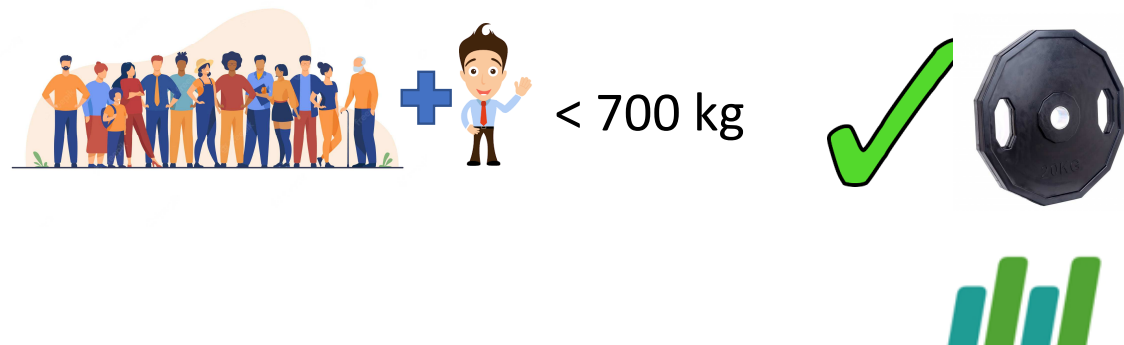
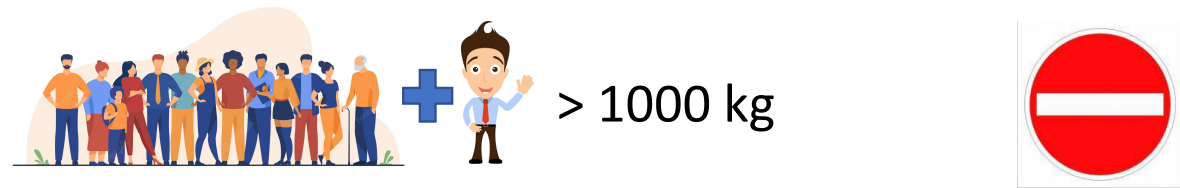
Requirement 2

If all seats are filled, then the gate closes (people should wait for the next round)



Requirement 3

- If a visitor would enter and the total weight exceeds 1000 kg, then the visitor must stay behind and the gate closes.
- If the gate is closed and the total weight of the people on the ride is between 700 and 1000 kg, the ride can go.
- If the gate is closed and the total weight is less than 700 kg, extra weight blocks are added.

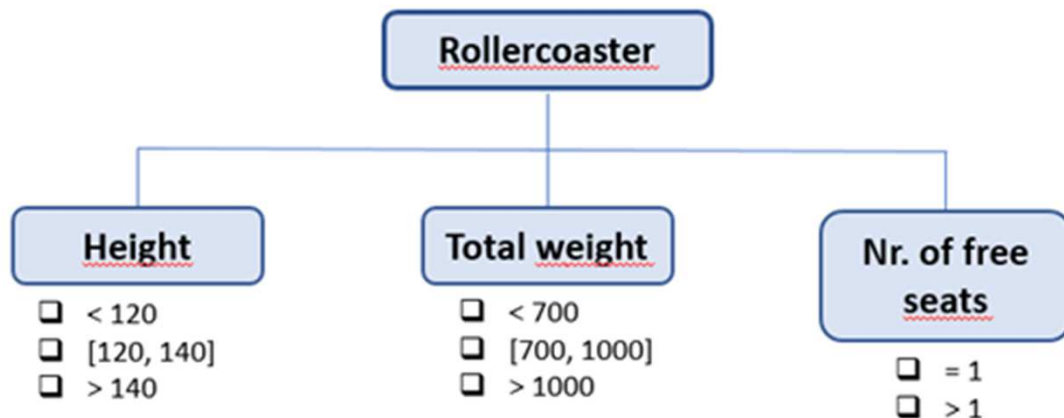




The Bad and Ugly solutions




CPH is always assumed. We can determine the equivalence partitions e.g. via the classification tree method:



Then various test selection criteria can be applied (depending on the risk):

- Linear techniques: each choice, base choice, diff-pair (combinative) testing,
- All-pairs testing,
- Combinatorial testing, etc.





Since the risk is very high, we apply the combinatorial one with 3 point BVA (it is the ugly solution, all the others are bad). Then, the test set is the Cartesian product

$$TS = \{119,120,121,139,140,141\} \times \{699,700,701,999,1000,1001\} \times \{1, 2\}$$

(72 test cases, this is the „strongest” test set we can produce with traditional techniques)



All the 72 tests passes against the code:

Suppose that there is a visitor 210 cm tall and 140 kg weight and the total weight before he enters is 960 kg. Suppose that there are three free seats available. The program `rc(210, 1100, 3)` gives the incorrect result “can enter”, seriously endangering people’s lives

```
def rc(height, totalWeight, freeSeat):
    msg = ""
    if height < 120:
        msg = "not allowed to enter"; return(msg)
    if totalWeight == 1001:
        msg = "gate closes"; return(msg)
    if freeSeat <= 0:
        msg = "gate closes"; return(msg)
    if height <= 140:
        msg = "can enter with seat-extender"
    else:
        msg = "can enter"
    if freeSeat == 1:
        if totalWeight < 700:
            msg = msg + ", " + "gate closes, extra-blocks needed"
        else:
            msg = msg + ", " + "gate closes"
    return(msg)
```

Fulfills the Competent Programmer Hypothesis





Is there any reliable test selection criterion for this problem?

A reliable test set T has the property that either for each $t \in T$, t passes if and only if the code is correct or there is at least one test t for which the software fails. A test selection criterion C is reliable if any generated test $T(C)$ satisfying C is reliable (according to Goodenough and Gerhart)

YES: General Predicate Testing (GPT)





The Good solution

We apply the GPT tool freely available at
<https://test-design.org>



BLACK BOX



ZERO KNOWLEDGE

Suppose that we know nothing about the SW

```
// Rollercoaster
height(int); totalWeight(int); freeSeats(int)
|
<120;    *; *
>=120; >1000; *
[120,140]; <=1000; $=1
[120,140]; <=1000; $=2
>140;    [700,1000]; $=1
>140;    <700; $=1
>140;    <=1000; $=2
```

GPT results in
25 test cases
that able to
reveal **ANY**
predicate fault





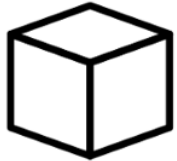
Suppose that we know the following (correct) architecture (decomposition):

- Step 1 Handle the exceptional cases, when the visitor is not allowed to enter;
- Step 2 Handle the cases when the visitor can enter;
- Step 3 Check the need for the extra weight blocks.

GPT results in 18 reliable test cases



WHITE BOX



FULL KNOWLEDGE

IF height < 120 THEN the visitor is not allowed to enter
ELSE IF total weight > 1000 THEN gate closes, visitor should wait for the next ride
ELSE IF height ≤ 140 THEN the visitor can enter with seat-extender,
ELSE IF height > 140 THEN the visitor can enter,
IF total weight < 700 AND free seat number = 1 THEN gate closes, extra bocks needed
IF total weight ≥ 700 AND free seat number = 1 THEN gate closes

```
// Rollercoaster  
  
height(int); weight(int); seat_number(int)  
<120; *; *  
*; >1000; *  
<=140; *; *  
>140; *; *  
*; <700; $=1  
*; >=700; $=1
```

Generated test cases

■ Show interval values

* can be any value you like

	height	weight	seat_number
T1	120	1000	*
T2	118	1002	*
T3	142	698	1
T4	141	699	1
T5	119	1001	1
T6	140	700	1

GPT results in 6 reliable test cases



Comparision

	Nr. of tests	Cumulative design time (effort)	Bug revealing capability
Traditional 3 point BVA with combinatorial TC selection	72	45 mins (15 mins design + 30 mins expected output computation)	Non-reliable
Black-box GPT	25	15 mins (5 min design + 10 mins expected output)	Reliable
Gray-box GPT	18	15 mins (3 mins design + 5 mins review + 7 mins expected output)	Reliable
White-box GPT	6	15 mins (3 mins design + 9 mins review + 3 mins expected output)	Reliable



Applicability

- The usage of GPT in safety-critical systems is a must
- During regression testing you can reduce the number of tests to be run





Summary

- Choosing the right test design technique and test selection criterion is crucial
- ...especially for safety-critical systems
- The tool support may drastically reduce the design time
- GPT is reliable for revealing any predicate fault

Detailed explanation of the Rollercoaster example can be found at <https://test-design.org>





Thank you! 😊

Q/A

