

# Combinatorial Test Design

Entwicklertag 2014, Karlsruhe

Juergen Heymann, SAP AG, May 2014



# The Problem of Testing

---

- ▶ SAP produces very complex and highly configurable software
  - ▶ using direct input, configuration settings, transactional data
- ▶ We cannot test all combinations!
- ▶ ... but customers do use 'all combinations'



*There are more theoretically possible execution paths in a typical program of 10000 lines of code than atoms in the universe.  
Assuming 250 independent branches:  $2^{250} \approx 10^{77}$*

# What is "Combinatorial Test Design"?

---

- **A 'black box' test technique**

(= views function of software from the outside)

- **that combines several test design techniques**

- **to deal well with the problem of 'all combinations'**

# Outline

---

**A. Introduction to the Method**

**B. Examples from Application Modeling & Testing**

**C. Is it worth it?**





# Introduction to the Method

# Step 1: Finding the Use Cases / Test Cases

Assume you have a certain scope of functionality to test...

## Step 1. Find 'use cases' / test cases (UI example)

- Ask "What can I do here?" ('user action')
- Each action triggers a behavior / function
  - 'Enter' / action-buttons but also menu items, tool bar clicks, ...
  - API: call of a service / function / method / ...
- Each action / 'use case' becomes a test case

The screenshot shows a SAP web interface for 'EU: Request Long-Term Vendor Declarations'. The form contains several sections: 'Selection Data' (Organizational Data, VendorProduct), 'Default Data for Long-Term Vendor Declarations' (Validity Period), and 'Form Output' (Contact Person, Form Language Control, Level of Detail, Sort Sequence). Annotations include: 'p1, p2, p3' pointing to the top navigation bar; 'F8' pointing to the 'Print' button; 'menu1' pointing to the 'Organizational Data' section; 'p4' pointing to the 'Form Output' section; and 'tool3' pointing to the 'Sort Sequence' dropdown. A yellow box at the bottom of the screenshot contains the text 'What can I do here?'.

This should give you a complete list of possible actions / test cases that cover the test object – and have to be tested (test = setup-action-verify)

# Step 2: Find the parameters of the test / action

## Step 2. Find parameters of the test case

### Differentiate between

- **Dummy Data:** just needed to run tests, not varied in testing, e.g. customer address, article name
- **Test Model Relevant Parameters:** affect the behavior of code under test (CUT)

### Test Relevant Parameters can be found in

- **Direct Input** e.g. parameters in API, on screen
- **Application Configuration**
- **Object Data** in DB

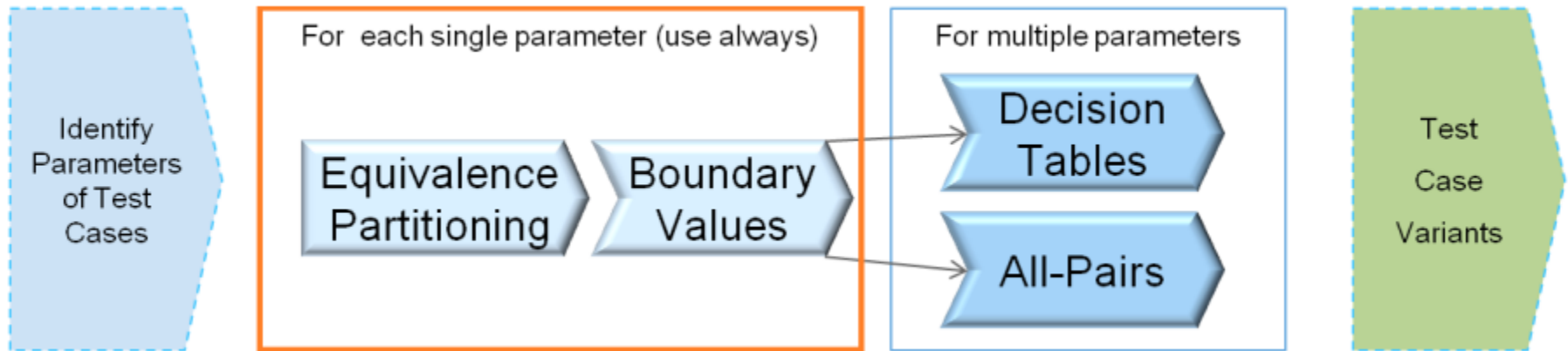
The screenshot shows a SAP web form titled "EU: Request Long-Term Vendor Declarations". The form is divided into several sections:

- Selection Data:** Includes "Organizational Data" with fields for "Address: Unit" (FTD\_DE1000) and "Foreign Trade Organization DE/19190 Waldorf".
- Vendor/Product:** Includes "Business Partner" and "Product Number" fields, both highlighted with red boxes.
- Default Data for Long-Term Vendor Declaration:** Includes "Validity Period" (01.01.2009 to 31.12.2009).
- Form Output:** Includes "Contact Person" (AND, BMMO), "Form Language Control" (Business Partner's Language), "Level of Detail" (Standard), and "Sort Sequence" (Sort Products by Material Number), all highlighted with red boxes.
- Print Output:** Includes checkboxes for "No" and "Yes".

A yellow circle labeled "Exec" is positioned to the right of the form, with a line pointing to the top right corner of the form. A yellow box labeled "Find parameters" is positioned below the form, with a line pointing to the red boxes in the "Vendor/Product" and "Form Output" sections.

This gives you the set of parameters that are relevant for this test case

# Parameter Oriented Test Design



These 4 test design methods helps us to cover the parameter space in a smart way to find **more defects** with **less effort!**

The methods focus on **functional correctness**.



# 1: Identify Equivalence Classes

## Example: Insurance Handling

- ▶ 'Age' field is age of applicant
- ▶ Contracts / formulas are different for *age groups*: **18...30, 31...50, 51...65**
- ▶ Outside the above age groups, no insurance policies are offered

Begin Date:	<input type="text" value="27.09.2010"/>
Age:	<input type="text"/>

### Equivalence Class Representatives

**Classes** "<18", "18-30", "31-50", "51-65", ">65"

#### Values for Test

**illegal:** 16, 66

**valid:** 20, 40, 55

## 2: Analyze Boundary Values

... to be considered in edge cases

---

Programming errors often happen at the boundaries of

- ▶ value domains  
e.g. at edges of allowed value ranges (equivalence classes)
- ▶ implementation logic  
e.g. first/last item in table, zero runs of a loop, ...

### Boundary Values (from Insurance Application)

**Classes** "<18", "18-30", "31-50", "51-65", ">65"

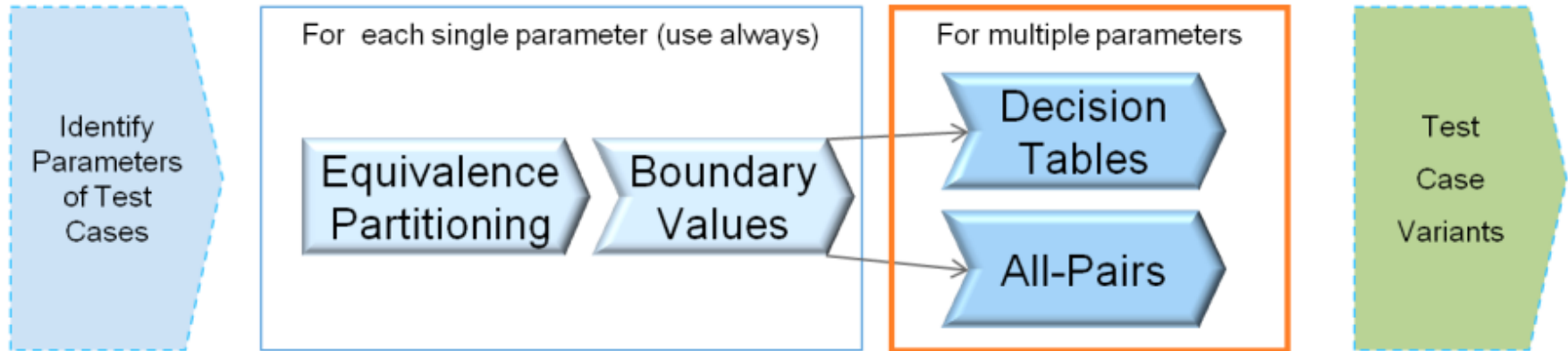
**Values for Test**

**illegal:** -1, 17, 66

**valid:** 18, 30, 31, 50, 51, 65

⇒ Test with both boundaries, not just a middle representative! ⇐

# Handling Multiple Parameters



We have applied 'Equivalence Classing' and 'Boundary Value Analysis' to individual parameters.

Now: How define **combinations of parameters** for functional testing?

# 3: Decision Tables – for small models

## Example: Account Opening

---

### Approach (again): Find the *parameters* and the values

Specification: Successful opening of an account requires that the applicant identifies himself through an **identity card**. Minors also require **consent of their parents**. When an account is opened, an overdraft limit can be granted. The preconditions for this are that the **credit check** is successful and that the applicant is **legally an adult**.

But many tests have more than 4 parameters ...

---



Let's focus on a method that supports this case very well ...

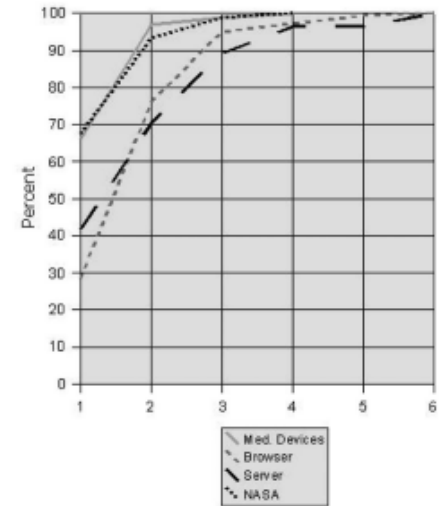


# 4: All-Pairs Testing – The Foundation

**The Empirical Foundation:** Most faults (70-90%) occur by combining two parameters 'just right'

## All-Pairs Testing

- ▶ Generates minimal test case variants that cover all pair combinations
- ▶ Is a proven industry best practice



# All-Pairs Testing – the Key Concept

What it means to 'combine all pairs':

**All values**

P1	P2	P3
0	0	0
1	1	1

Not effective

**All combinations**

P1	P2	P3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Not efficient

**All value pairs**

P1	P2	P3
0	0	0
0	1	1
1	0	1
1	1	0

Efficient and effective

The 'savings' increase exponentially with the number of parameters

- ▶ E.g. 500 binary parameters can be covered with All-Pairs with 20 test cases

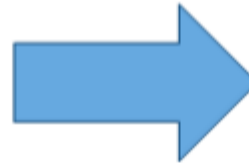
# An All-Pairs Example

## Some parameters of a function ...

Input value

Radio  opt1  Check1  
 opt2  Check2  
 opt3  Check3  
 Check4

Choices



## Model defines variables and values (PICT)

```
# Demo model for All-Pairs Training
#
Radio: 1,2,3
check1: 0,1
check2: 0,1
check3: 0,1
check4: 0,1

choices1: 1,2,3,4
choices2: 1,2,3
```



**Exhaustive test:**  
 $3 \times 2 \times 2 \times 2 \times 2 \times 4 \times 3 = 576 !$

	Radio	Check1	Check2	Check3	Check4	choices1	choices2
1	1	0	0	0	0	1	1
2	3	1	1	1	1	2	1
3	2	1	0	1	0	2	1
4	2	0	1	0	1	1	1
5	3	1	0	0	1	1	2
6	2	0	1	1	0	4	2
7	1	0	1	0	1	4	3
8	1	1	0	1	0	3	1
9	1	1	0				
10	3	0	1				
11	3	1	1				
12	2	1	0	0	1	3	3
13	2	0	0	1	0	2	1
14	3	1	0	1	0	4	3

**All-Pairs: 14 test cases!**

# All-Pairs Tool Support

---

**You cannot generate 'minimal pair coverage' by hand!**

- We use the PICT engine from Microsoft + XLS as wrapper for convenience





# Examples



# Example 1: Retail Pricing

## Retail Management System – Sales Pricing

**POS Manager** Store Code: 1 User ID: John Smith

All these fields are irrelevant for model (→ 'other test data')

Single field errors need to be tested, e.g. date in the future etc. BUT: Error checks are outside of All-Pairs model scope

**Detail**

Mix Match ID  \* Effective Date  \* Effective Time   
Description  \* Expiration Date  \* Expiration Time   
Discount Description  \*

**Criteria**

Unique Item  Complete Mix Match  
 Match Item  Prorated  
 Any Item Combination  Allow Return Item

**Pricing Rules**

Attribute Name	Op.	Qty	Discount	Amount	Apply To
			<input checked="" type="radio"/> Dollar Off		<input type="text" value="Item"/>
	<input type="text" value="="/>	<input type="text"/>	<input type="radio"/> Pct Off	<input type="text"/>	<input type="checkbox"/> Lowest Priced Items <input type="text" value="D"/>
			<input type="radio"/> Sell Price		Count <input type="text"/>

First set of relevant parameters:

**MMCriteria: Unique, Match, Any**  
**CompleteMM: 0,1**  
**ProratedMM: 0,1**  
**AllowRetMM: 0,1**

New Rule Save

Local intranet 100%

# Combining all techniques 2

**Criteria**

Unique Item                       Complete Mix Match  
 Match Item                               Prorated  
 Any Item Combination                   Allow Return Item

---

**Pricing Rules**

Attribute Name	Op.	Qty	Discount	Amount	Apply To
<input checked="" type="radio"/> Item Key		<input type="text"/>	<input checked="" type="radio"/> Dollar Off	<input type="text"/>	Item
<input type="radio"/> Eligibility Rule ID	=	<input type="text"/>	<input type="radio"/> Pct Off	<input type="text"/>	<input type="checkbox"/> Lowest Priced Items
			<input type="radio"/> Sell Price		Count <input type="text"/>

**ItemKey:** must exist, but is single-field error condition  
 → ignore in model

**EligibilityRule1:** ItemKeyExisting, NoDiscountsToAll, AllItems, Cosmetics, VenderSelection, UnitChargeDiscount

**Operation1:** Greater|GreaterEQ, EQ, LessEQ|Less  
 # aliasing : we consider Greater|GreaterEQ 'almost the same' / equivalent; they are iterated but generate no addtl. pair coverage

**Qty :** 2  
 # Qty: states how many items of the same type (item key) need to be bought before the special price / ... applies.  
 # Qty=2 covers the case that the program identifies at least 2 separate items to belong to the 'special' (as 'condition items').  
 # You need the proper test data to 'trigger' this quantity

**Discount1:** DollarOff, PctOff, SellPrice

**# Amount:** ignored for model, just use any test data; error checks: amount off > price, pct off > 100, Sell price > orig price

**ApplyTo1:** Item, Group

# constraint: LowestPriceltem only used with EligibilityRules; item/group only with item

**LowestPriceltems1:** 0,1

# count semantics?

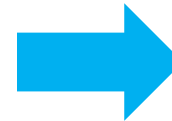
**Count1:** 0, 1, 10

# Example 2: FINDSTR (Windows Cmd tool)

Searches for strings in files.

```
FINDSTR [/B] [/E] [/L] [/R] [/S] [/I] [/X] [/V] [/N] [/M] [/O] [/P] [/F:file]
[/C:string] [/G:file] [/D:dir list] [/A:color attributes] [/OFF[LINE]]
strings [[drive:][path]filename[ ...]]

/B      Matches pattern if at the beginning of a line.
/E      Matches pattern if at the end of a line.
/L      Uses search strings literally.
/R      Uses search strings as regular expressions.
/S      Searches for matching files in the current directory and all
        subdirectories.
/I      Specifies that the search is not to be case-sensitive.
/X      Prints lines that match exactly.
/V      Prints only lines that do not contain a match.
/N      Prints the line number before each line that matches.
/M      Prints only the filename if a file contains a match.
/O      Prints character offset before each matching line.
/P      Skip files with non-printable characters.
/OFF[LINE] Do not skip files with offline attribute set.
/A:attr Specifies color attribute with two hex digits. See "color /?"
/F:file Reads file list from the specified file(/ stands for console).
/C:string Uses specified string as a literal search string.
/G:file Gets search strings from the specified file(/ stands for console).
/D:dir Search a semicolon delimited list of directories
strings Text to be searched for.
[drive:][path]filename
        Specifies a file or files to search.
```



## Model File (PICT Syntax):

```
# PICT model for FINDSTR
#
# Consider all switches indepe
#-----

# which files
F: /F, _
S: /S, _
P: /P, _
OFF: /OFF, _
D: /D:1;2, _

# type of match
B: /B, _
E: /E, _
X: /X, _
regex: /L, /R, _
V: /V, _

# print format
N: /N, _
M: /M, _
O: /O, _
```



9	Testcase variants (copy from pict output)													1
	1	2	3	4	5	6	7	8	9	10	11	12		
	F	S	P	OFF	D	B	E	X	regex	V	N	M	O	
1	_	_	/P	/OFF	/D:1;2	/B	/E	_	_	/V	/N	_	/O	
2	/F	/S	/P	_	_	_	_	/X	/L	_	_	/M	_	
3	/F	_	_	/OFF	_	/B	/E	/X	/L	_	/N	_	_	
4	_	/S	_	_	/D:1;2	_	_	_	/L	/V	_	/M	/O	
5	_	/S	_	_	_	/B	/E	_	_	_	_	/M	_	
6	/F	_	/P	_	/D:1;2	_	_	/X	/R	_	/N	_	/O	
7	/F	/S	_	/OFF	_	_	/E	_	/R	/V	/N	/M	/O	
8	_	_	/P	/OFF	/D:1;2	/B	_	/X	/R	/V	_	/M	_	
9	/F	/S	_	/OFF	/D:1;2	_	_	/X	_	_	_	_	/O	
10														

# Example 3: Code Level (C++)

Generated test cases → generated test code

```
enum t_dataType      { _int, _string };
enum t_dataDist     { _empty, _single_val, _single_val_full, _sequential, _random,
enum t_scanScenario { _no_values, _one_value, _two_values, _all_values, _non_exist
enum t_operator     { _equal, _non_equal, _match, _similarity_search, _all };
//...

typedef struct t_caseRow {
    t_dataType      dataType;
    t_dataDist      dataDist;
    t_scanScenario  scanScenario;
    t_operator      op;
//...
};

//----- AllPairs generated combination cases -----
t_caseRow testCases[] = {
    { _int, _uniform, _two_values, _equal, _first_row, _bit_vector, _and, _result_docs
    { _int, _complex_w_null, _no_values, _similarity_search, _hit_before_last_row, _ve
    { _string, _sequential, _non_existing, _non_equal, _full_result, _bit_vector, _or,
    // ... (more cases left out)
};

//----- CODE -----
// first define a fixture class
class TestClass : public ::testing::TestWithParam<t_caseRow>
{
protected:
    virtual void SetUp();
    virtual void TearDown() {};

void runTest(); // runs one case of the testCases table
t_caseRow& _tc; // ref to current test case (row)
};
```

parameters and values

cases table

case interpreter  
(incl. check code)



**Is it worth it?**



# Is it worth it?

---

## Results from 10 applications

- Average of 4x less find-cost-per-defect ('best case' was 10x)
- Modeling effort 1-2 hours per model

## Results from Code Example

- 500x less test cases than 'brute force' (=all combinations)
- 100x faster
- 99.6% code coverage (line)





# Thank you

Contact information:

Juergen Heymann  
juergen.heymann@sap.com

Resources:

- ❖ Youtube: [youtube.com/user/agilese](https://www.youtube.com/user/agilese)  
→ channel 'Combinatorial test design'
- ❖ Tools: <https://scn.sap.com/docs/DOC-48472>
- ❖ [www.pairwise.org](http://www.pairwise.org)





ENTWICKLERTAG

meet the **SPEAKER**  
@speakerlounge



1. OG DIREKT ÜBER DEM  
EMPFANG