From Software Testing to Intelligent Validation of Autonomous Systems

Du Test Logiciel à la Validation Intelligente des Systèmes Autonomes

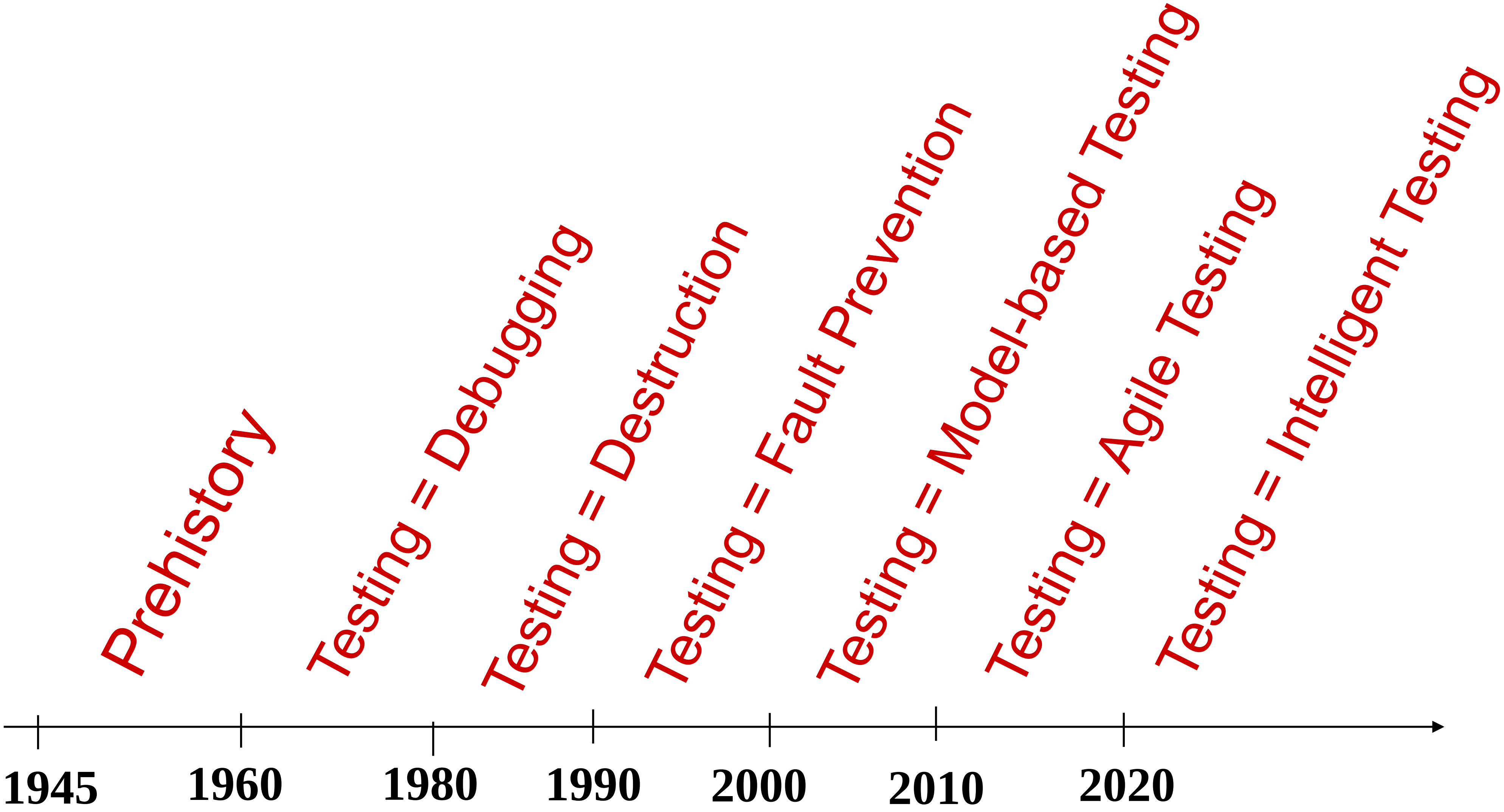
Ecole des Jeunes Chercheurs en Programmation
2023

Arnaud Gotlieb
Simula Research Laboratory
Norway

Course Overview

- Software Testing Introduction
- Code-based Testing
- Testing of Autonomous Systems
- Open Challenges in Software Testing

A Historical Perspective on Software Testing



Prehistory

Testing = Debugging

Testing = Destruction

Testing = Fault Prevention

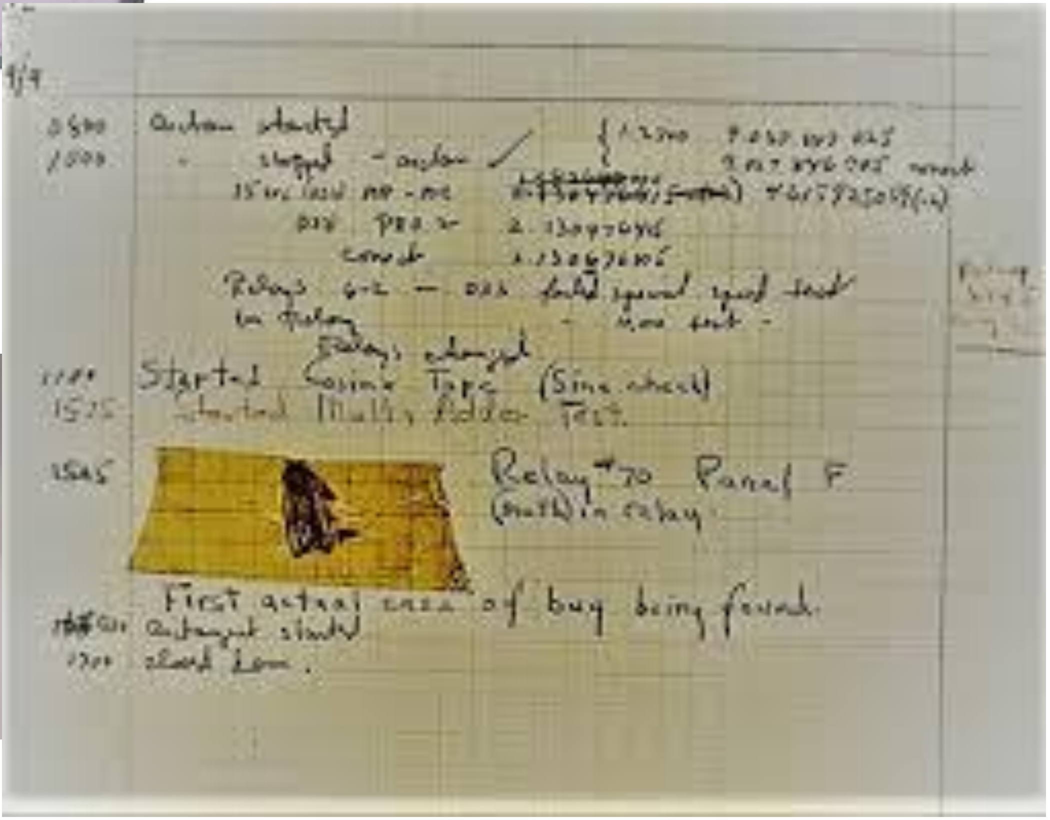
Testing = Model-based Testing

Testing = Agile Testing

Testing = Intelligent Testing

9/9/1947

Grace Hooper



9/9

0800 Anctan started

1000 " stopped - anctan ✓ {1.2700 9.037 847 025
9.037 846 995 conect
[illegible]
Relays 6-2 in 033 failed special speed test
in relay - [illegible] test -
Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Multi Adder Test.

1545 Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

1630 antangent started.

1700 closed down.

1960-80: Testing = Debugging

What have we learnt since then?

Causality: **Error → Fault → Failure**

In fact, 3 distinct activities:

* Failure detection	(Testing purpose)
* Fault localization	(Debugging purpose)
* Error correction	(Debugging purpose)

1980-90: Testing = Destruction

“Testing is the process of executing a program with the intent of finding errors” [G. Myers *The Art of Software Testing* 1979]

<u>Consequently:</u>

validation team ≠ development team

But, there is no specification to test the program against

That dogmatic position was progressively given up!

1990-2000: Testing = Fault Prevention

“To ***convince*** *that a* ***program conforms to its <u>specifications</u>*** *by using* ***static or dynamic analysis*** *techniques”*

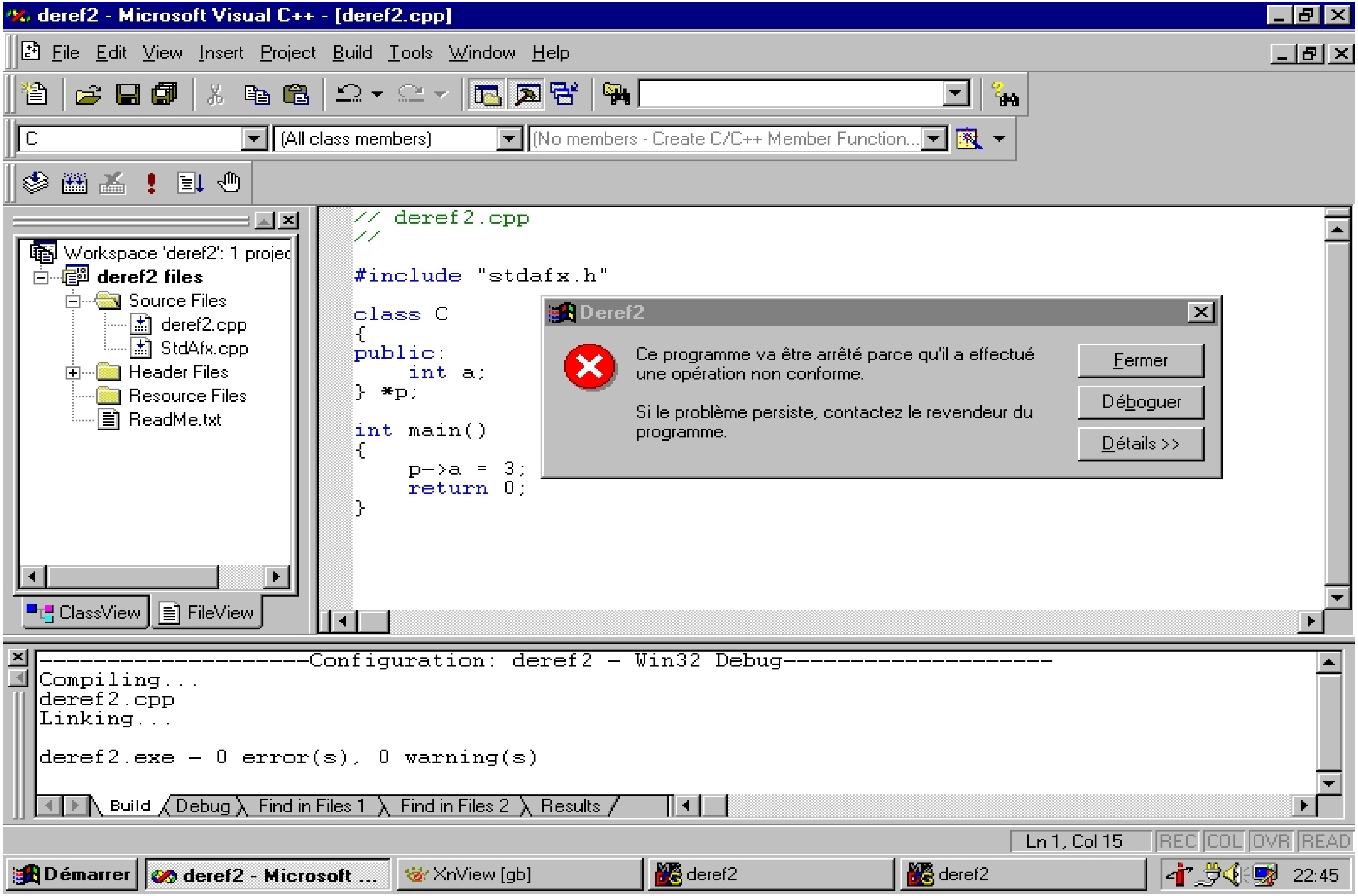
- Program analysis → Control: Property checking

Before execution

- Program execution → Testing: Result evaluation

After execution

Visual 1998



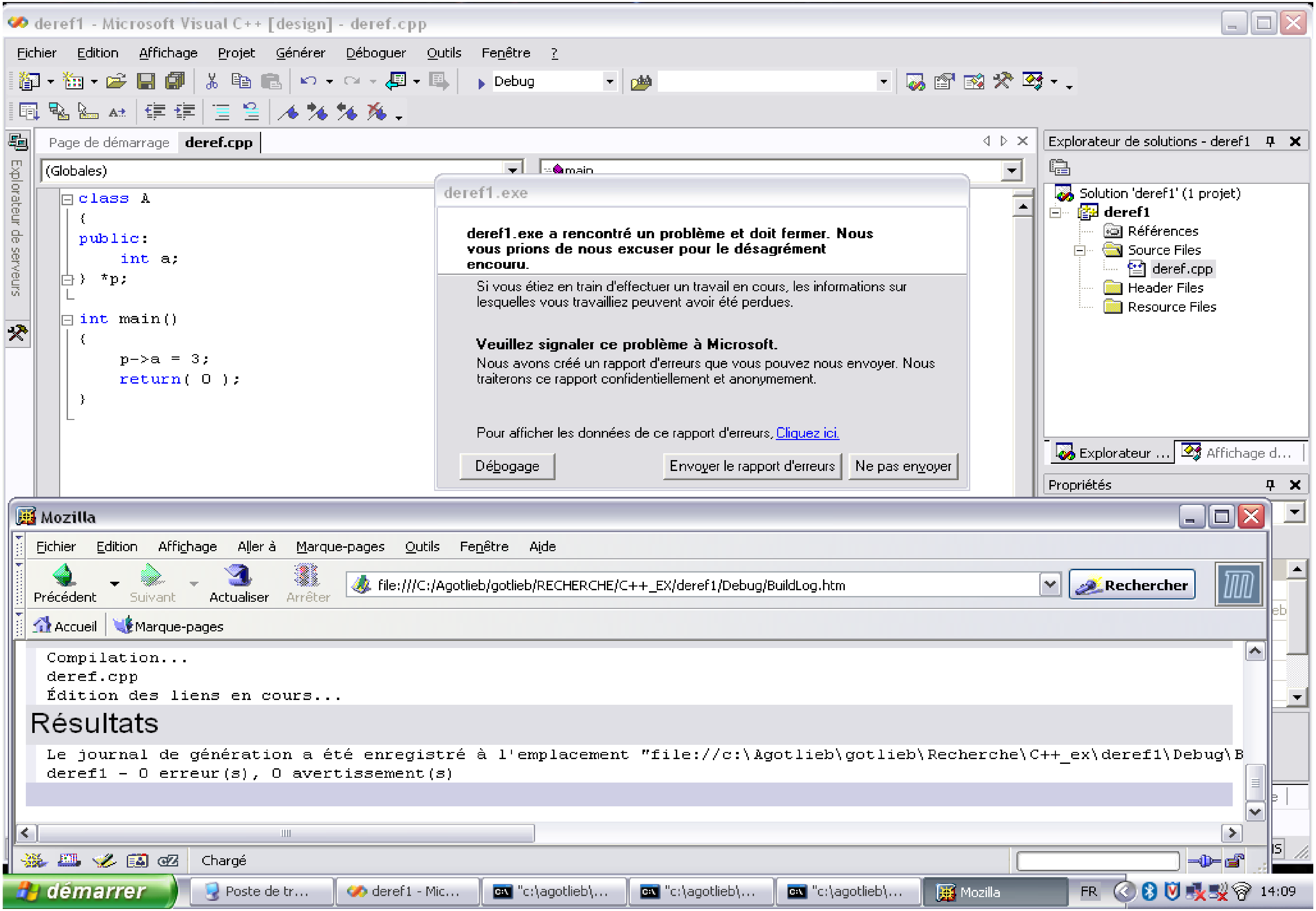
```
// deref2.cpp
//

#include "stdafx.h"

class C
{
public:
	int a;
} *p;

int main()
{
	p->a = 3;
	return 0;
}
```

Ce programme va être arrêté parce qu'il a effectué une opération non conforme.

Si le problème persiste, contactez le revendeur du programme.

```
--------------------Configuration: deref2 - Win32 Debug--------------------
Compiling...
deref2.cpp
Linking...

deref2.exe - 0 error(s), 0 warning(s)
```

Visual 2017



```
class A
{
public:
    int a;
} *p;

int main()
{
    p->a = 3;
    return( 0 );
}
```

deref1.exe

deref1.exe a rencontré un problème et doit fermer. Nous vous prions de nous excuser pour le désagrément encouru.

Si vous étiez en train d'effectuer un travail en cours, les informations sur lesquelles vous travailliez peuvent avoir été perdues.

Veuillez signaler ce problème à Microsoft.

Nous avons créé un rapport d'erreurs que vous pouvez nous envoyer. Nous traiterons ce rapport confidentiellement et anonymement.

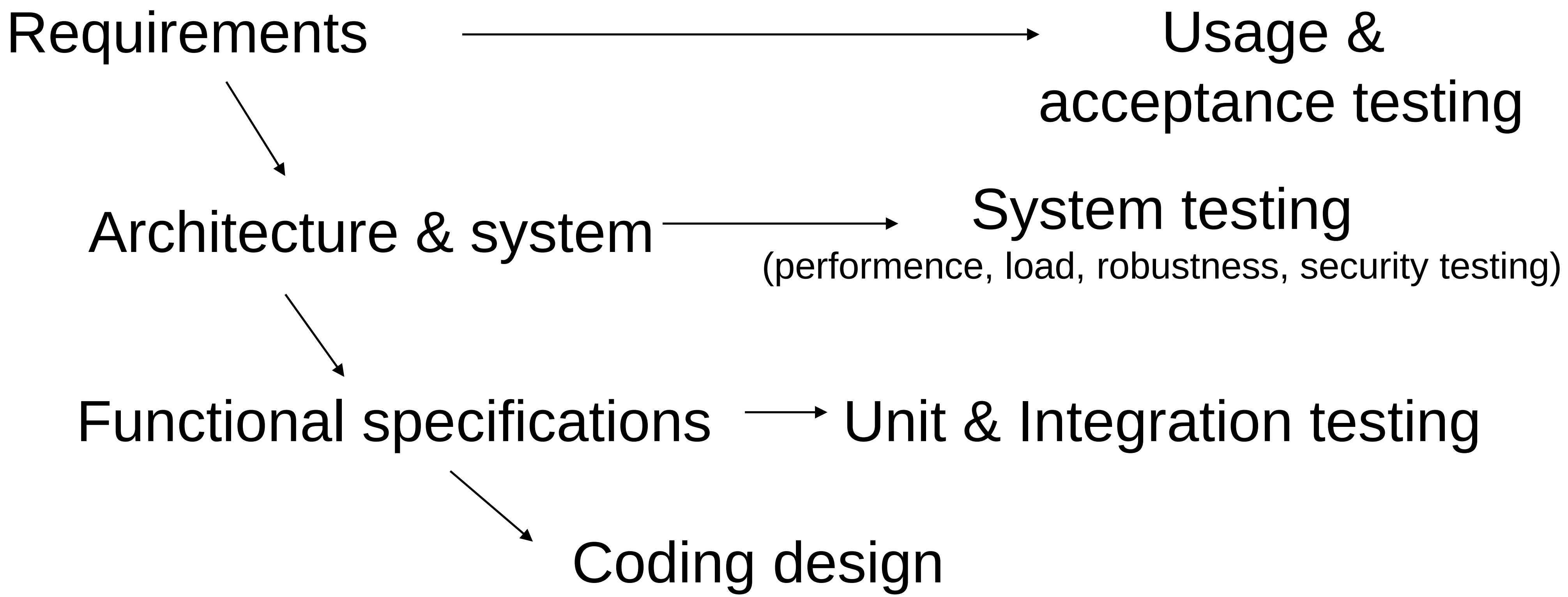
Pour afficher les données de ce rapport d'erreurs, Cliquez ici.

```
Compilation...
deref.cpp
Édition des liens en cours...
```

Résultats

```
Le journal de génération a été enregistré à l'emplacement "file://c:\Agotlieb\gotlieb\Recherche\C++_ex\deref1\Debug\B
deref1 - 0 erreur(s), 0 avertissement(s)
```

Software testing in the V software developpement process:



Requirements

Usage & acceptance testing

Architecture & system

System testing

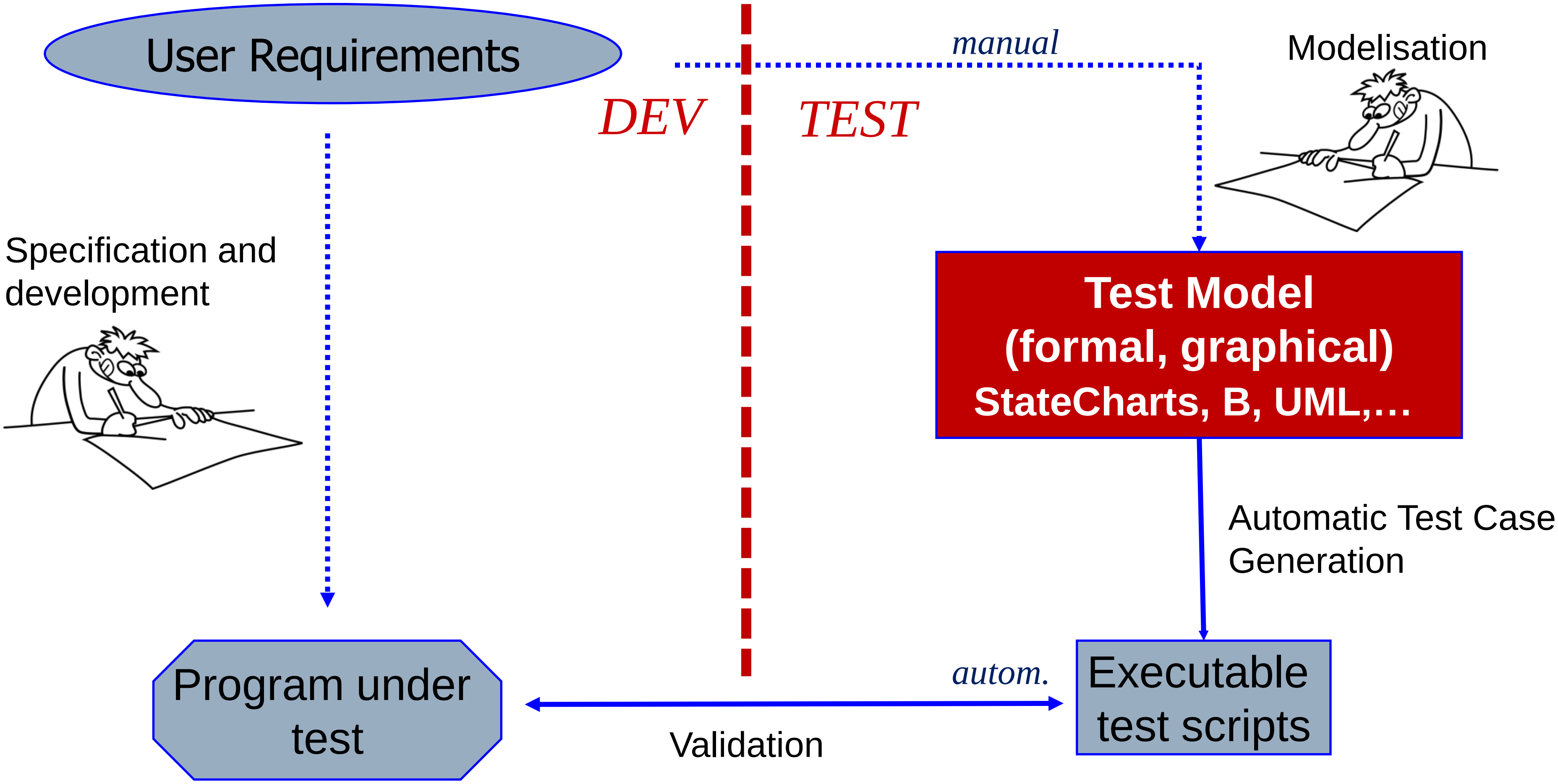
(performence, load, robustness, security testing)

Functional specifications

Unit & Integration testing

Coding design

2000-2010 : Testing = Model-Based Testing (MBT)



User Requirements

DEV

TEST

manual

Modelisation

Specification and development

Test Model (formal, graphical)
StateCharts, B, UML,…

Automatic Test Case Generation

Program under test

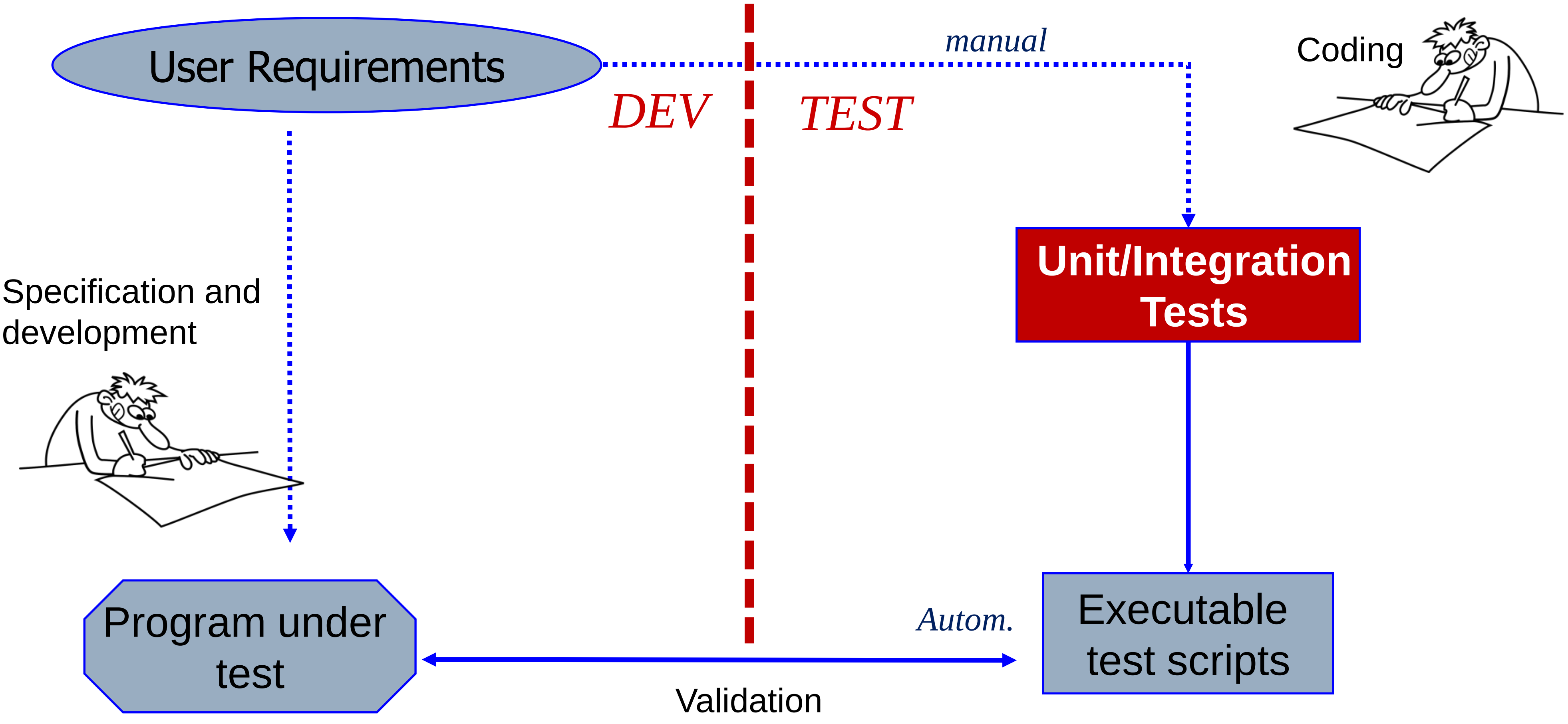
autom.

Executable test scripts

Validation

MBT added-value: Build a (test) model instead of test cases to validate/verify the program

2010-2020: Testing = Agile Testing/Test-Driven Dev. (TDD)



User Requirements

DEV

TEST

manual

Coding

Unit/Integration Tests

Specification and development

Program under test

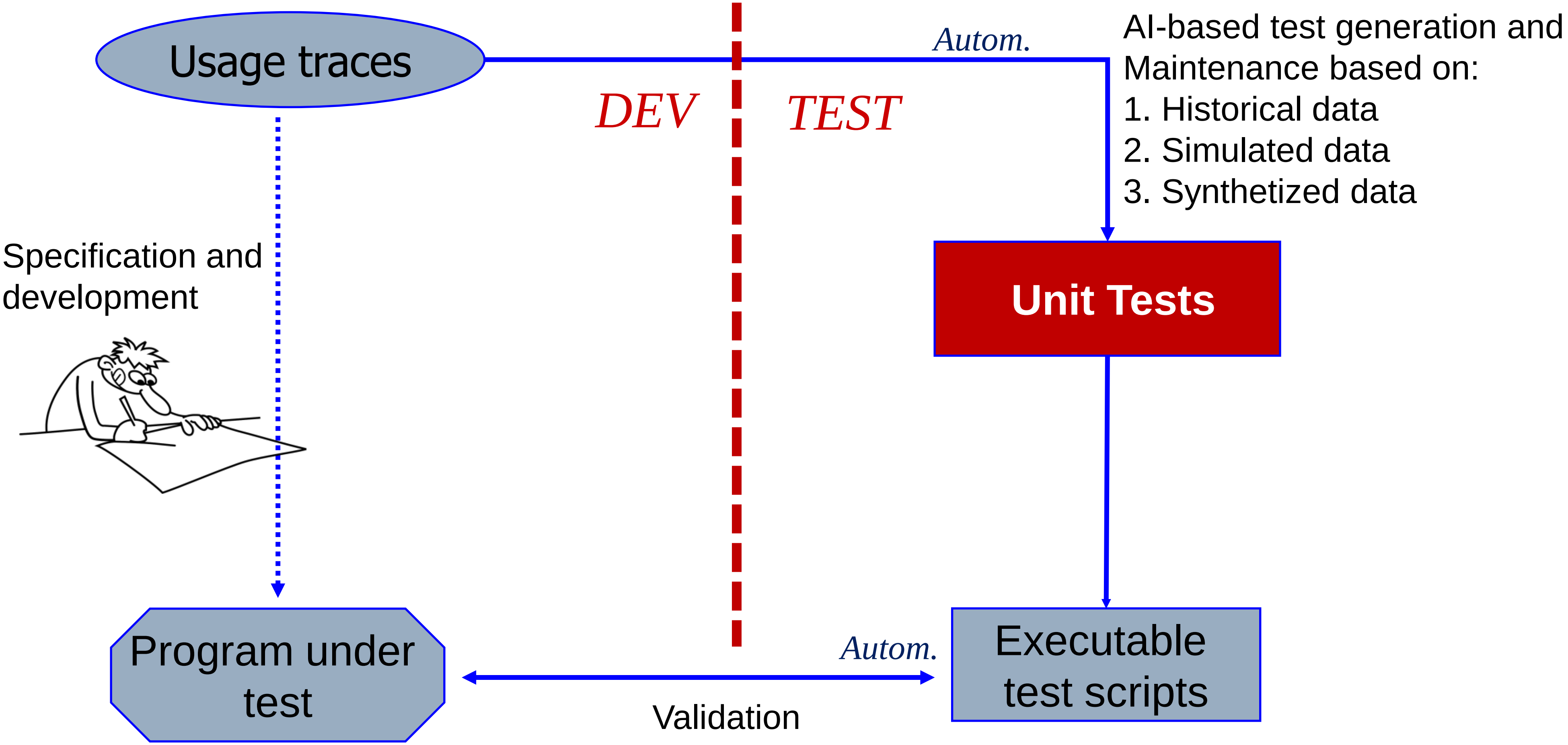
Autom.

Executable test scripts

Validation

Writing tests instead of a specification model is considered more agile

2020-20..: Testing = Intelligent Testing / AI-driven Testing



AI-based test generation and Maintenance based on:
1. Historical data
2. Simulated data
3. Synthetized data

AI is revolutionizing the way software systems are developed and tested

Terminology

(IEEE Standard Glossary of SE, BCS’s standard for Softw. Testing)

- ***Validation***: “The process of evaluating software at the end of software development to ensure compliance with intented usage” -- Are we developing the right product ?
- ***Verification***: “The process of determining whether the products of a given phase of the software development process fulfill the requirements established during the previous phase” -- Are we developing the product right ?
- ***Testing:*** “Evaluating software by observing its execution”

Program Testing: Our Definition

- Testing = **Execute** a program `P` to detect **faults,** which are **non-conformities** w.r.t. the program specification `F`

- Looking for counter-examples:

$$\exists ?X \text{ tq } P(X) \neq F(X)$$

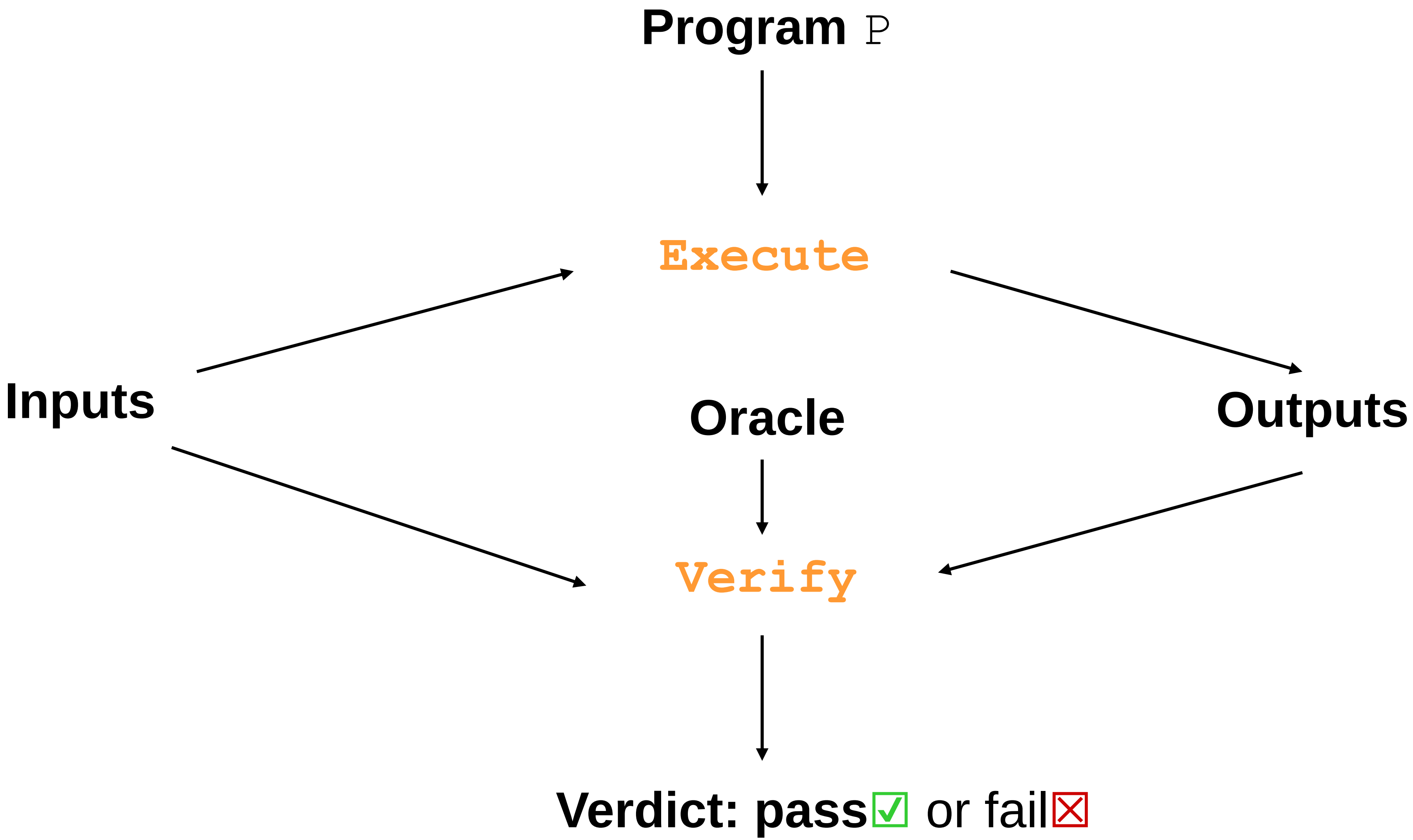
Program Correction: Fundamental Limitation

Impossibility to demonstrate the correction of a program in the general case as a consequence of **the undecidability of the Halting problem of a Turing machine**

“Program Testing can be used to prove the presence of bugs, but never their absence” [Dijkstra 74]

PS : Expert developer → ~1 fault / 10 LOC
~163 faults / 1000 instructions
[B. Beizer *Software Testing Techniques* 1990]

Test Process



Program `P`

`Execute`

Inputs

Oracle

Outputs

`Verify`

Verdict: pass☑ or fail☒

Oracle Problem :
How to verify the computed outcomes?

In Theory:

- By predicting the expected result
- By using a formulae extracted from the specification
- By using another program
- By using known properties about multiple executions of the program

In Practice :

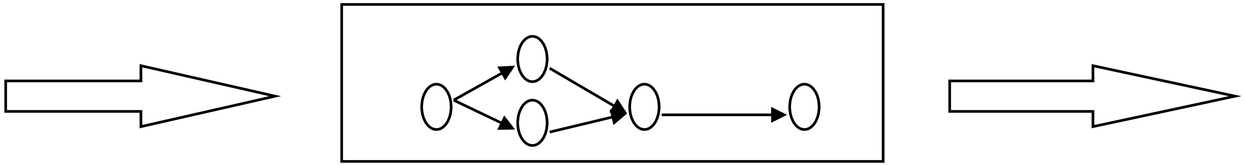
- Approximative predictions (due to floating-point computations,…)
- Unknown formula (because Program = Formulae)
- Non bug-free oracles and incorrect properties

Test Input Selection Problem

How to choose inputs for testing?

A. Black-box Testing: Using sepcifications to generate test inputs

B. Code-Based Testing: Using the program code and structure



A. Black-box Testing

Using a specification model:

- **Informal** (Partition Testing, Boundary Testing, ...)
- **Half-formal** (Use cases, Sequence diagrams, UML/OCL, Causes/effects graphs…)
- **Formal** (Algebraic specifications, B Machines, Transition systems, IOLTS, …)

B. Code-Based Testing

Using a model computed from the source code of the program under test

- model = Internal representation of the program structure
- Heavy usage of **Graph Theory**, in particular, coverage techniques

Code-Based Testing is indispensable (1)

Specification:
Return the product of
i by j

(i = 0, j = 0) --> 0
(i = 10, j = 100) -->1000
…

--> OK

```
prod(int i,int j )
  {
    int k ;
    if( i==2 )
        k := i << 1 ;
    else
       (…)
    return k ;
  }
```

Code-Based Testing is indispensable! (2)

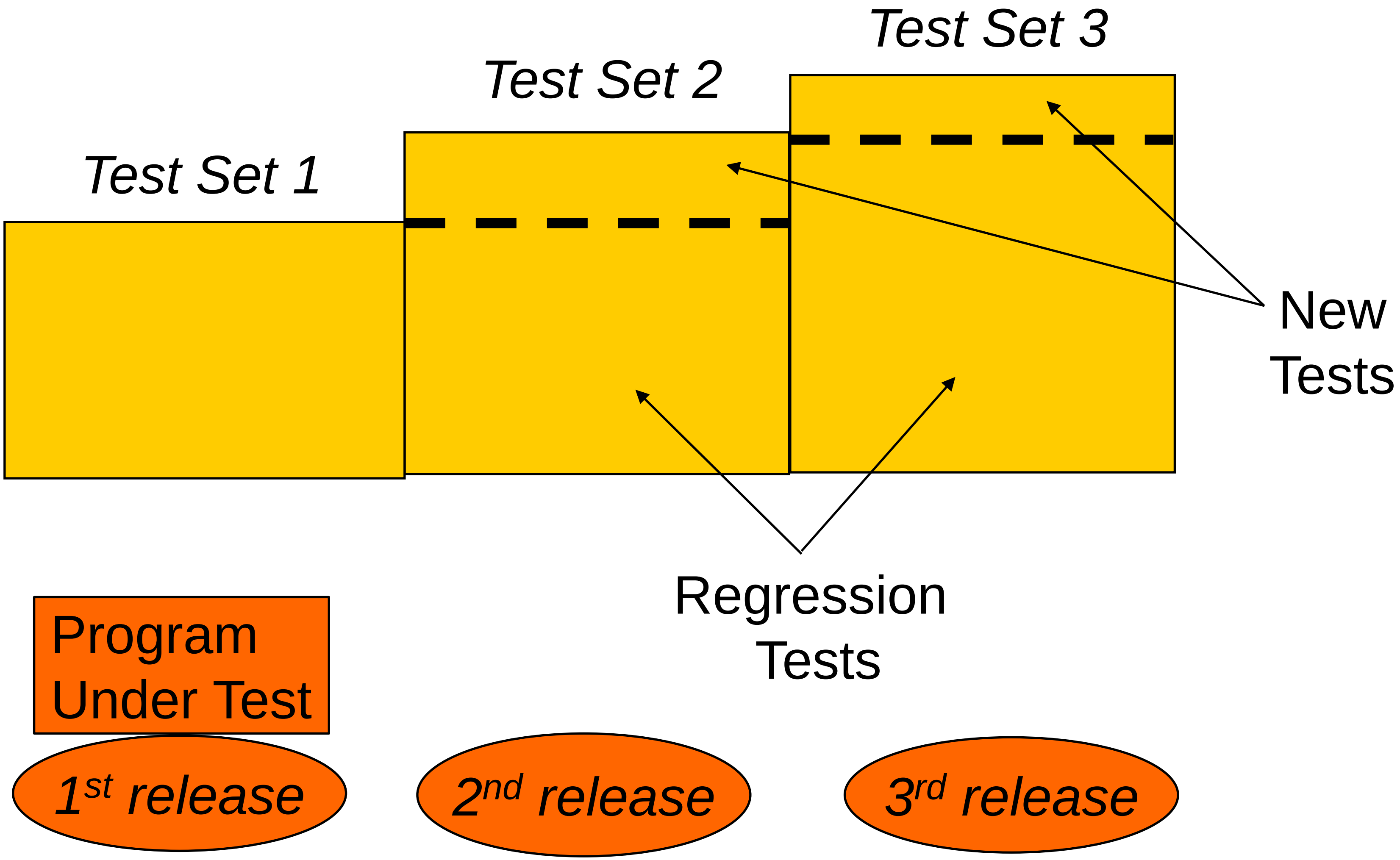
Specifications :
renvoie le produit de
i par j

(i = 0, j = 0) --> 0
(i = 10, j = 100) -->1000
…

```
prod(int i,int j )
  {
    int k ;
    if( i==2 )
       k := i << 1 ;
    else
       (…)
    return k ;
  }
```

Undetected fault if only
black-box testing is
used par
patch → **k := j << 1**

Regression Testing



Test Set 1

Test Set 2

Test Set 3

New Tests

Regression Tests

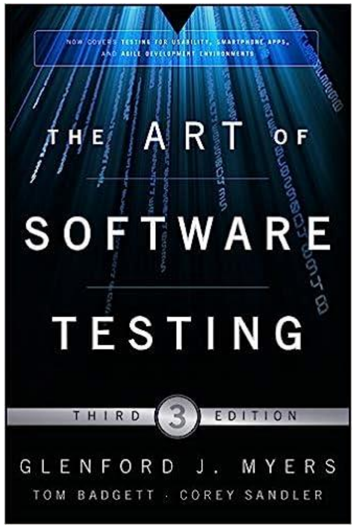
Program Under Test

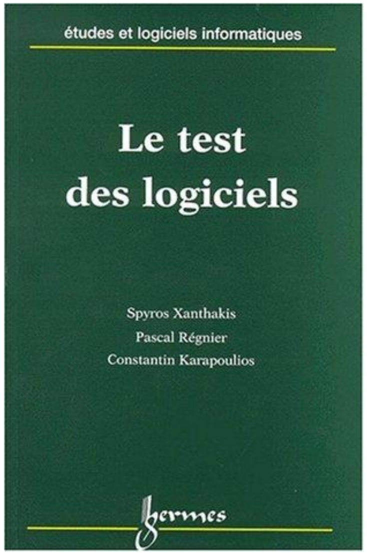
1st release

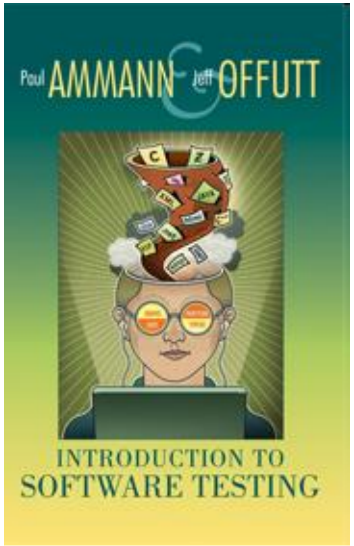
2nd release

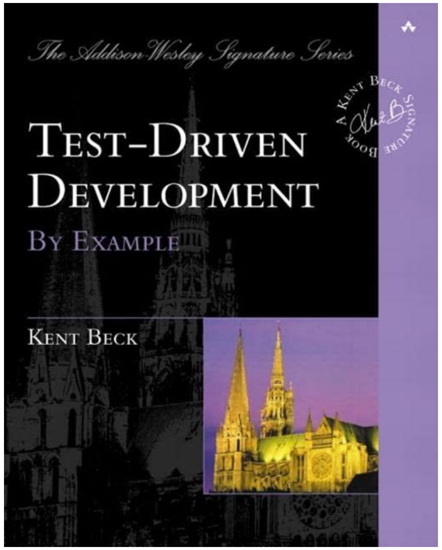
3rd release

Bibliography: Reference Books









Bibliography: Journals

L'optimisation par essaim particulaire
Maurice Clerc

Optimisation génétique et interactive de sites web
Antoine Oliver, Omar Regragui, Nicolas Monmarché, Gilles Venturini

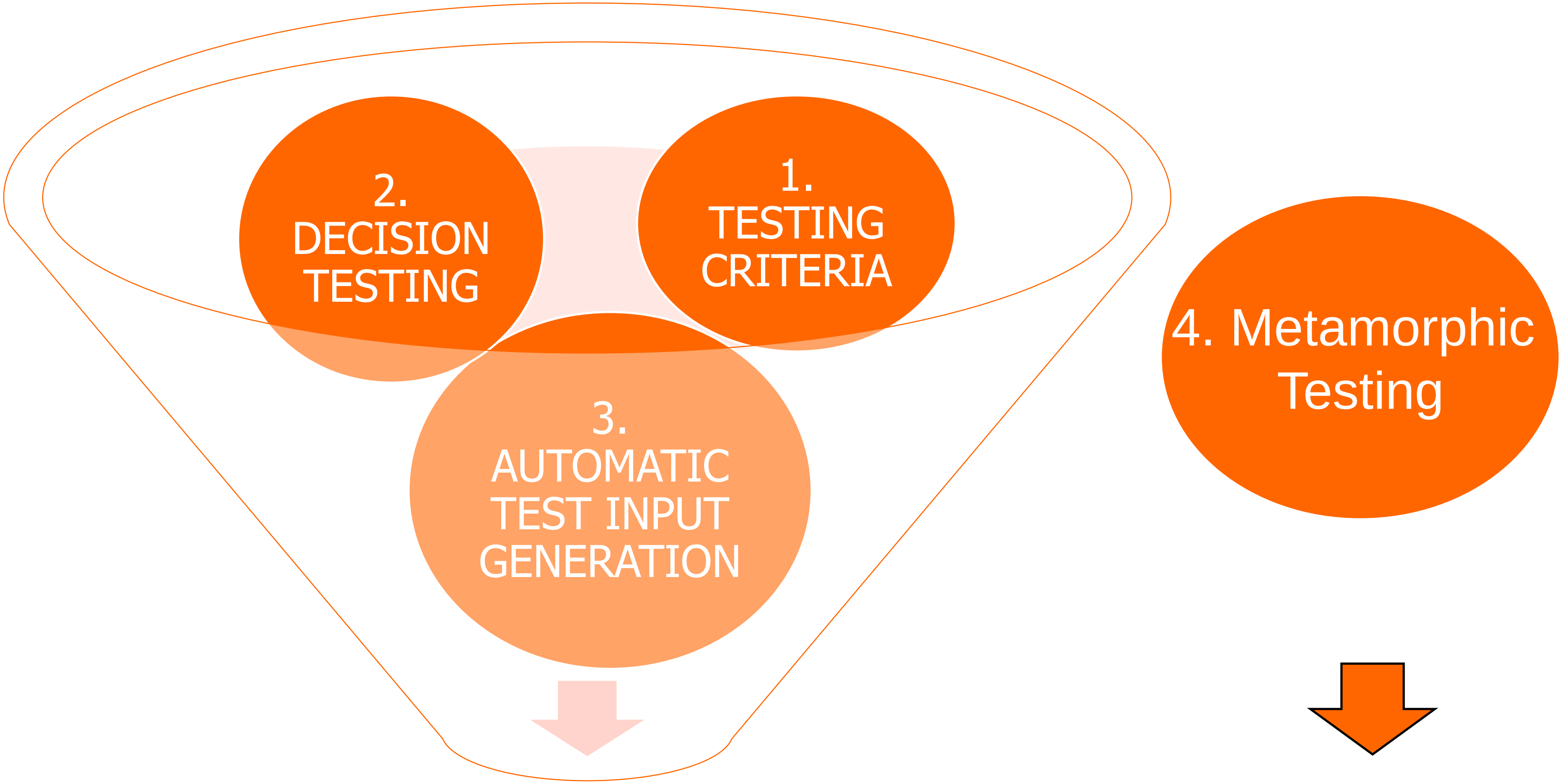
Construction de services distribués : une approche à base d'agents mobiles
Siegfried Rouvrais

Une approche pour réduire la complexité du flot de contrôle
dans les programmes C
Hugues Cassé, Louis Féraud, Christine Rochange, Pascal Sainrat

Edsger W. Dijkstra
P.A. de Marneffe

Course Overview

- Software Testing Introduction
- Code-based Testing
- Testing of Autonomous Systems
- Open Challenges in Software Testing



2. DECISION TESTING

1. TESTING CRITERIA

3. AUTOMATIC TEST INPUT GENERATION

4. Metamorphic Testing

Code-Based Testing

Oracle Problem

1. TESTING CRITERIA

Internal Representations

Program Structure Abstractions

- Control Flow Graph (CFG)

- Def/Use Graph

- Program Dependence Graph

Control Flow Graph (CFG)

Oriented and connex graph `(N,A,e,s)` where

N: set of nodes =
Instructions block sequentially executed

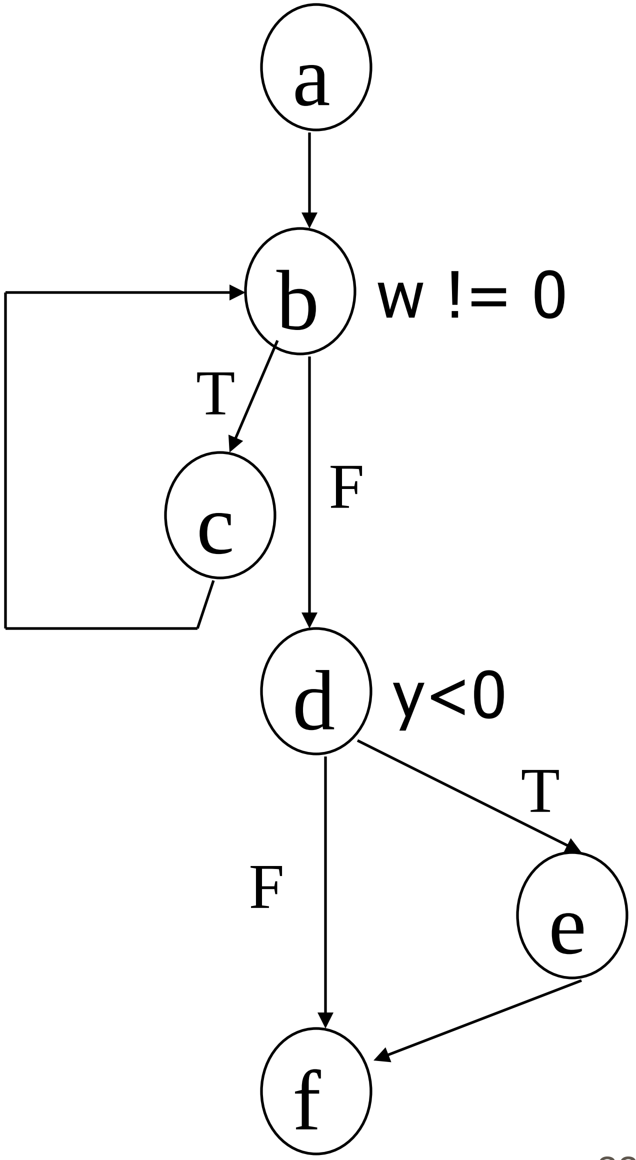
E: set of arcs, N x N relation,
Some arcs are labelled with {T, F} = Possible branching of the control flow

e: Program input node

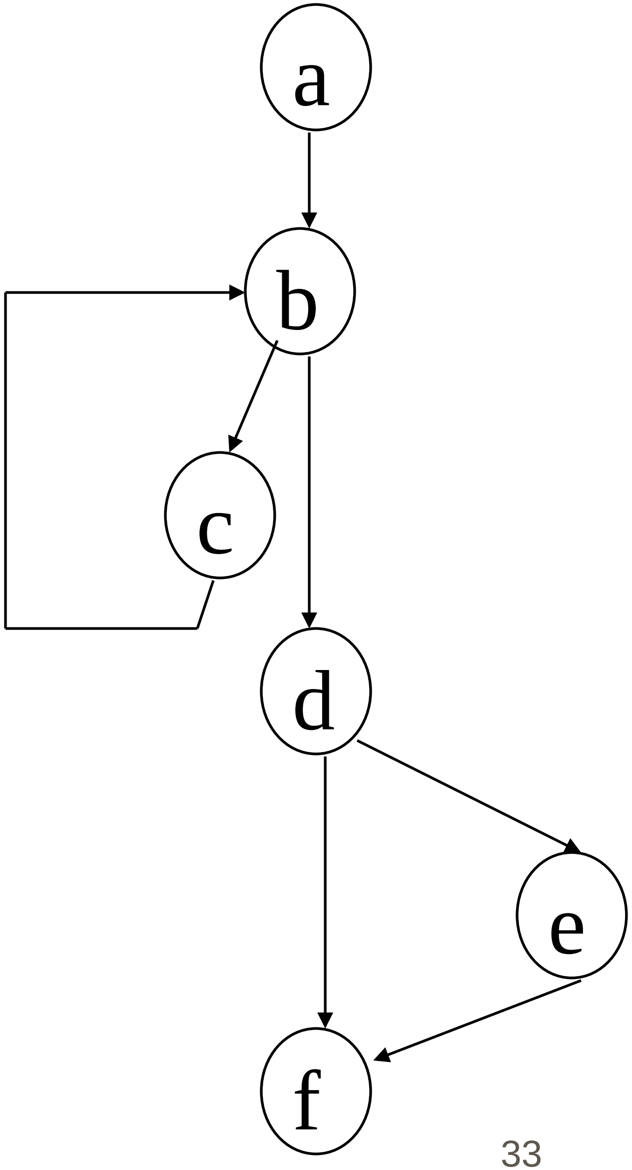
s: Program output node

Control Flow Graph (CFG): Example

```
double P(short x, short y) {
     short   w = abs(y) ;
     double z  = 1.0 ;

     while ( w != 0 )
         {
             z  = z * x ;
             w = w - 1 ;
          }

     if ( y<0 )
          z = 1.0 / z ;
   return(z) ;
   }
```



Structural Criterion: *All_nodes* | *All_statements*



Motivation: To cover all program instructions at least once during testing

Def: A subset `C` of program paths of the CFG `(N,A,e,s)` satisfies ***All_nodes***
iff $\forall n \in N, \exists C_i \in C$
such that `n` is a node of C_i

Example: Here, only one path is necessary
`a-b-c-b-d-e-f` [6/6 nodes]

Structural Criterion: *All_arcs* | *All_decisions*

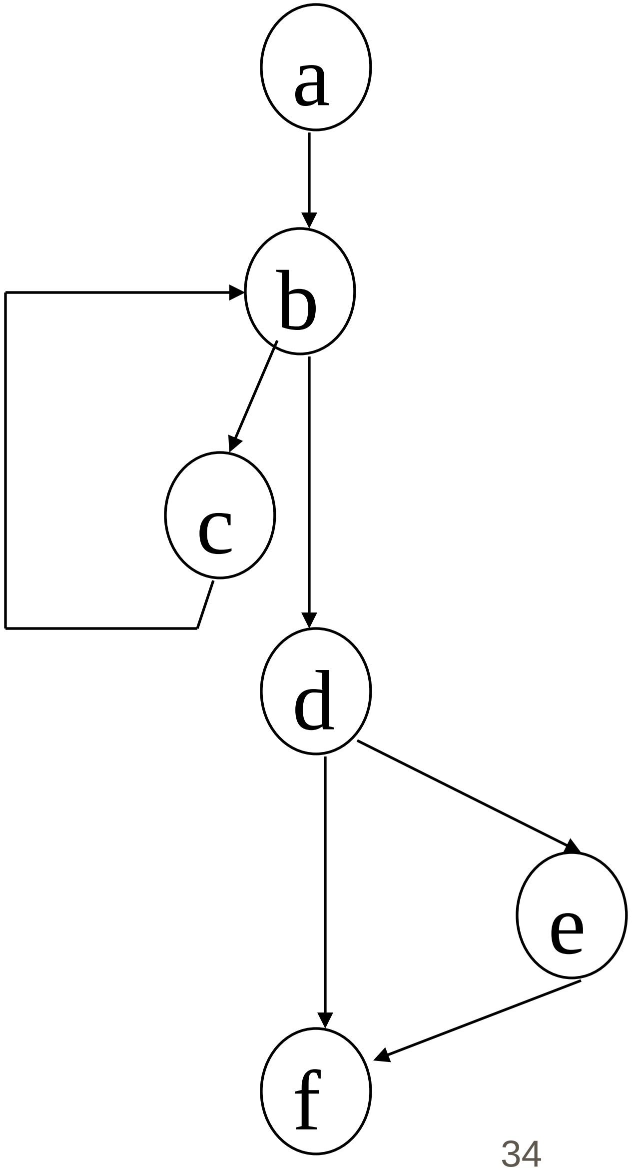
Motivation: To cover all program decisions at least once during testing

Def: A subset `C` of paths of the CFG `(N,A,e,s)` satisfies ***All_arcs*** iff $\forall$`a` $\in$ `A`, $\exists$$C_i$ $\in$ `C` such that `a` is an arc of C_i

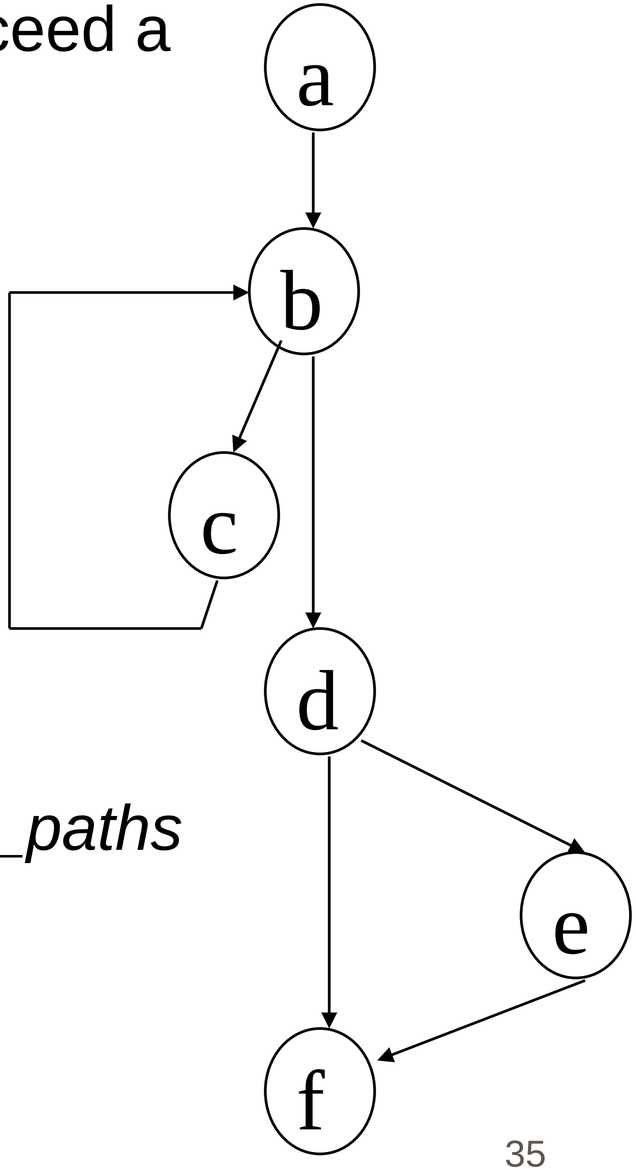
Example: Here, 2 paths are necessary

`a-b-c-b-d-e-f` [6/7 arcs]

`a-b-d-f` [3/7 arcs]



Structural Criterion: *All_simple_paths* | *All_k_paths*



Motivation: To cover all execution paths which do not iterate more than once in loops or do not exceed a given length

Example: Here, 4 simple paths are necessary to cover *All_simple_paths*

`a-b-d-f`

`a-b-d-e-f`

`a-b-c-b-d-f`

`a-b-c-b-d-e-f`

Example: 2 paths are necessary to cover *All_5_paths* (Paths with less than 5 instruction blocs)

`a-b-d-f`

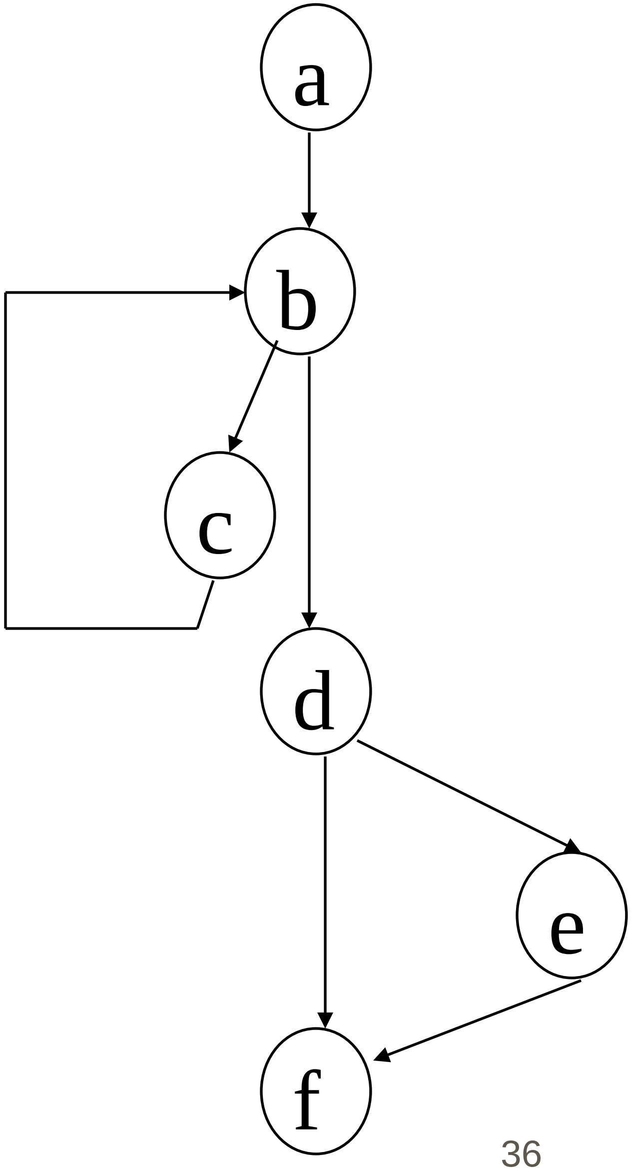
`a-b-d-e-f`

Structural Criterion: *All_paths*

Def: A set `C` of paths of the CFG `(N,A,e,s)` satisfies ***all_paths*** if `C` contains all paths from `e` to `s`

Here, it is **impossible** as there is an ∞ of paths. Note also that some paths may be **infeasible**!

All_paths is stronger than ***All_k_paths***
All_k_paths *is stronger than* ***All_arcs***
All_arcs is stronger than ***All_nodes***
…



Executed Path: `exec(P,X)`

Principle:

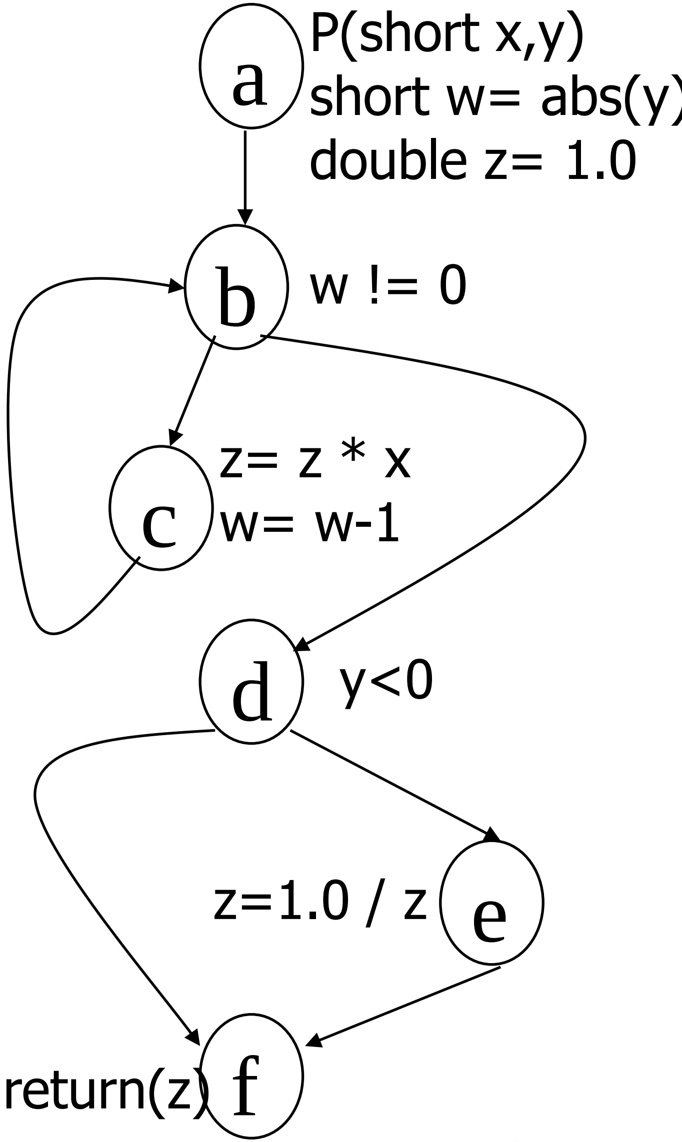
`X` executes a **single path** of the CFG (no concurrency, no dynamic bindings)

Def: Sequence of CFG nodes, not necessarily finite, followed by the execution flow when `P` is feeded with `X` as input

Examples:

`exec(P,(0,0)) =a-b-d-f`

`exec(P,(3,2)) =a-b-(c-b)`^{2}`-d-f`



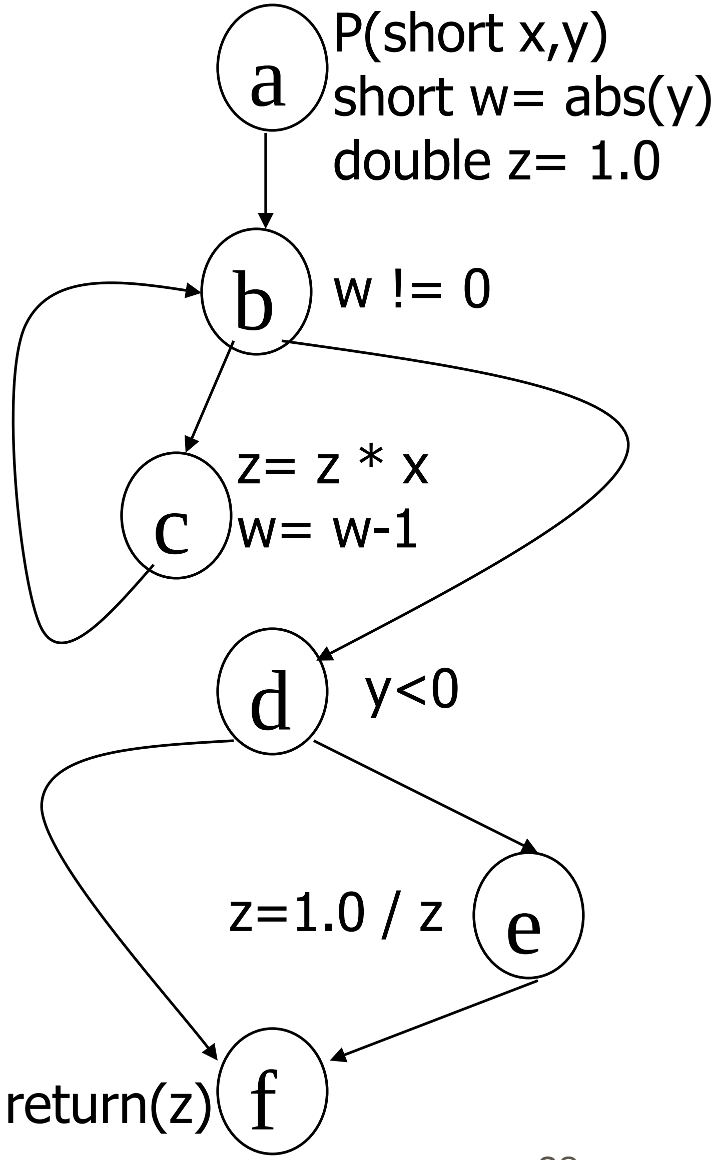
Infeasible Path Problem

Let `c` be a CFG path of `P`,
Does `X` exist such that `c=exec(P,X)` ?

Here, `a-b-d-e-f` is infeasible!

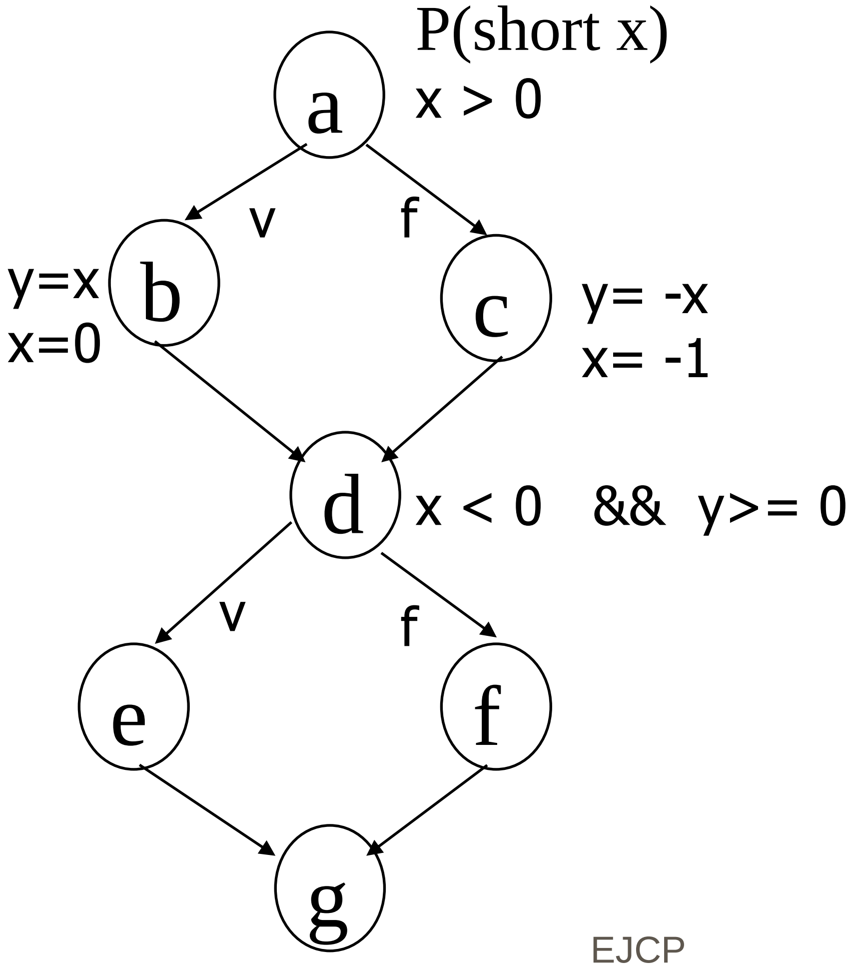
Weyuker 79
Determining if a node, an arc, or a path of the CFG is feasible is undecideable in the general case

Sketch of proof: Reduction to the Halting problem of a Turing Machine



Exercise:

Find the infeasible paths of the program



Measuring code coverage

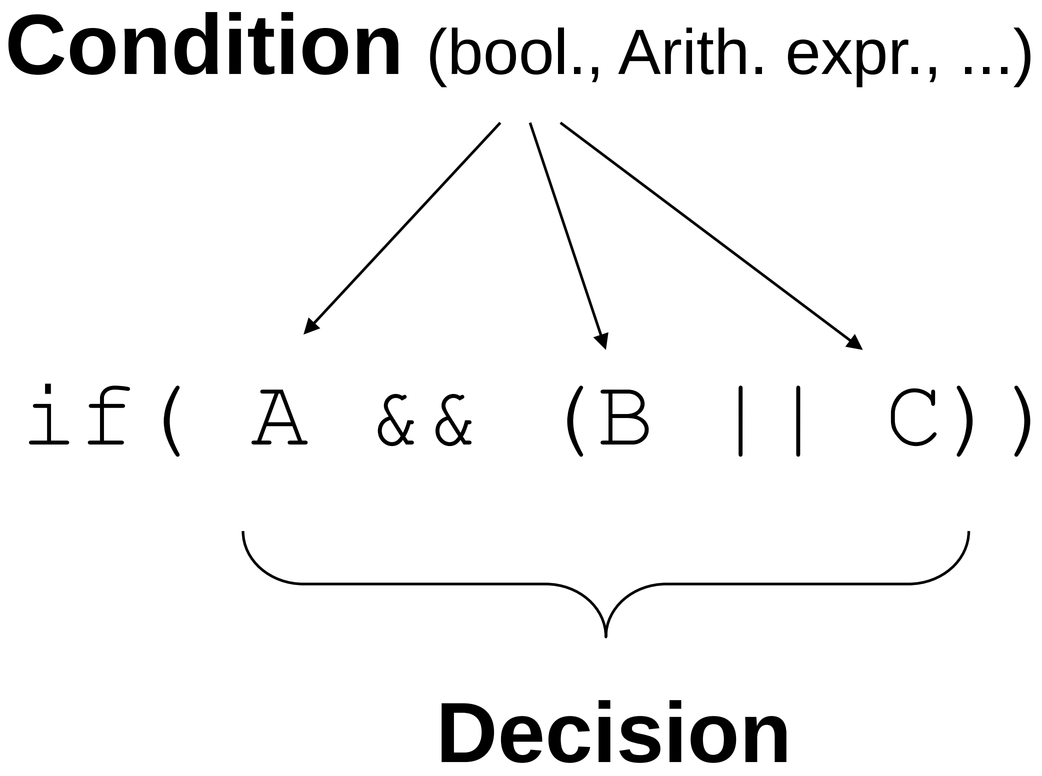
- 3 distinct techniques
  - Instrumenting source code
    - + Easy to implement
    - + Powerful as everything regarding executions can be recorded
    - - Add untrusted code in trusted source code

  - Instrumenting binary code
    - + Do not modify source code
    - - Difficult to implement

  - Use a debugger
    - + Do not modify source code
    - - Specific to each compiler

2. DECISION TESTING

Condition / Decision in a Program



Condition (bool., Arith. expr., ...)

`if( A && (B || C))`

Decision

(Logical predicate in a control structure of the program)

Notation: `Dec` is the truth value of the decision

Some Testing Criteria associated to Decisions

`if( A && (B || C))`

1. Decision Criterion (DC) : `A=1,B=1,C=0 – Dec=1`
   `A=0,B=0,C=0 – Dec=0`

2. Condition Criterion (CC) : `A=1,B=1,C=0 – Dec=1`
   `A=0,B=0,C=1 – Dec=0`

3. Modified Condition/Decision Criterion (MC/DC)

4. Multiple Condition/Decision Criterion: $2^3=8$ test cases

Modified Condition/Decision Criterion (1)

Objective: Démontrer l'action de chaque condition sur la valeur de vérité de la décision

```
if( A && (B || C))
```

Principe : for each condition, find 2 test cases which flip `Dec` when all the other conditions are fixed

Ex: For A

A=0, `B=1,C=1` -- **Dec=0**
A=1, `B=1,C=1` -- **Dec=1**

Modified Condition/Decision Criterion (2)

`if( A && (B || C))`

for A `A=0, B=1,C=1 -- Dec=0`

`A=1, B=1,C=1 -- Dec=1`

for B `A=1, B=1,C=0 -- Dec=1`

`A=1, B=0,C=0 -- Dec=0`

for C `A=1, B=0,C=1 -- Dec=1`

~~`A=1, B=0,C=0 -- Dec=0`~~

Here, 5 test cases are sufficient for covering MC/DC !

Exercise: Can we do better?

```
if( A && (B || C))
```

for A

```
A= , B= ,C=    -- Dec=
A= , B= ,C=    -- Dec=
```

for B

```
A= , B= ,C=    -- Dec=
A= , B= ,C=    -- Dec=
```

for C

```
A= , B= ,C=   -- Dec=
A= , B= ,C=   -- Dec=
```

Modified Condition/Decision Criterion (3)

Property: If n = #conditions then
covering MC/DC requieres at least n+1 TC and max 2n TC

$$n+1 \leq \#\text{Test cases} \leq 2*n$$

Coupled Conditions: Flipping the truth value of one condition impacts the truth value of another one

When there is no coupled conditions, the minimum (n+1) can always be reached [Ref ?]

Links with object-code coverage?

Covering MC/DC $\Rightarrow$ covering all the decisions of the object-code

But

Covering MC/DC $\xcancel{\Leftarrow}$ covering all the decisions of the object-code

Covering all paths of the object-code $\Rightarrow$ covering MC/DC

But

Covering all paths of the object-code $\xcancel{\Leftarrow}$ covering MC/DC

From the Galileo development standard

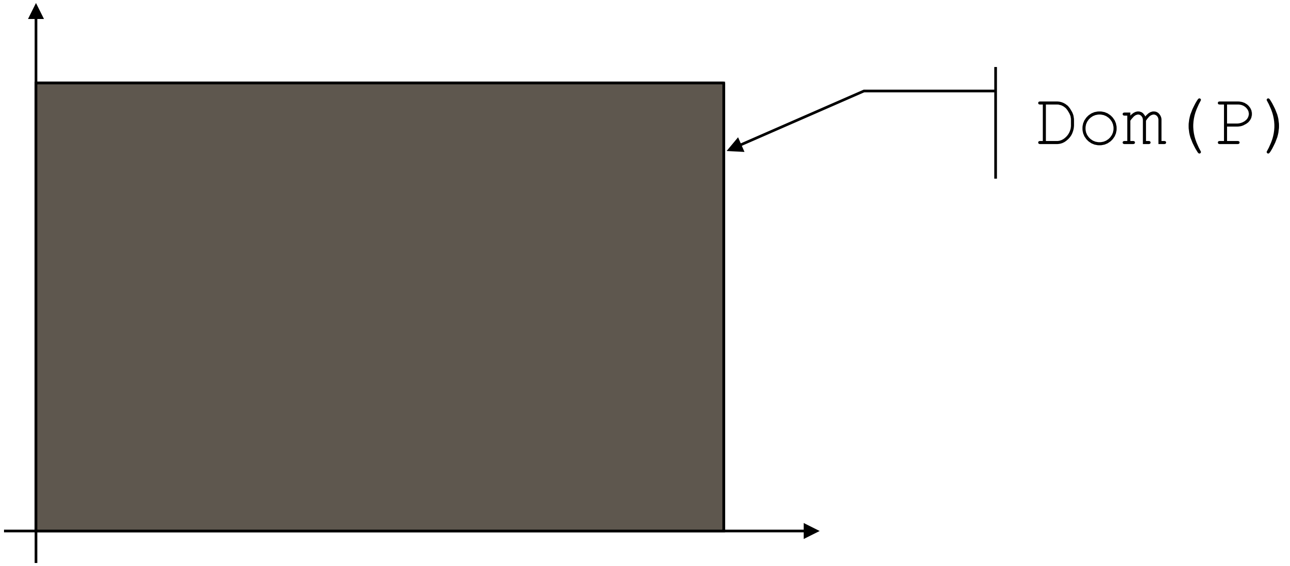
Structural coverage	DAL A	DAL B	DAL C	DAL D	DAL E
Statement coverage (source code)	**100%**	**100%**	**100%**	**90%**	N/A
Statement coverage (object code)	**100%**	N/A	N/A	N/A	N/A
Decision coverage (source code)	**100%**	**100%**	N/A	N/A	N/A
Modified Condition & Decision Coverage (Source code)	**100%**	N/A	N/A	N/A	N/A

3. Automatic Test Input Generation

Most Used Techniques

- Exhaustive Testing
- Testing by Sampling
- Random Testing (a.k.a. Fuzzing)
- Symbolic Execution

Exhaustive Testing



- Exhaustive sampling of the program input space

- Selection of all inputs and execution of the program

- Equivalent to a correction proof (when the execution terminates)

Exhaustive Testing: Limitations and Advantages

- Usually untractable!

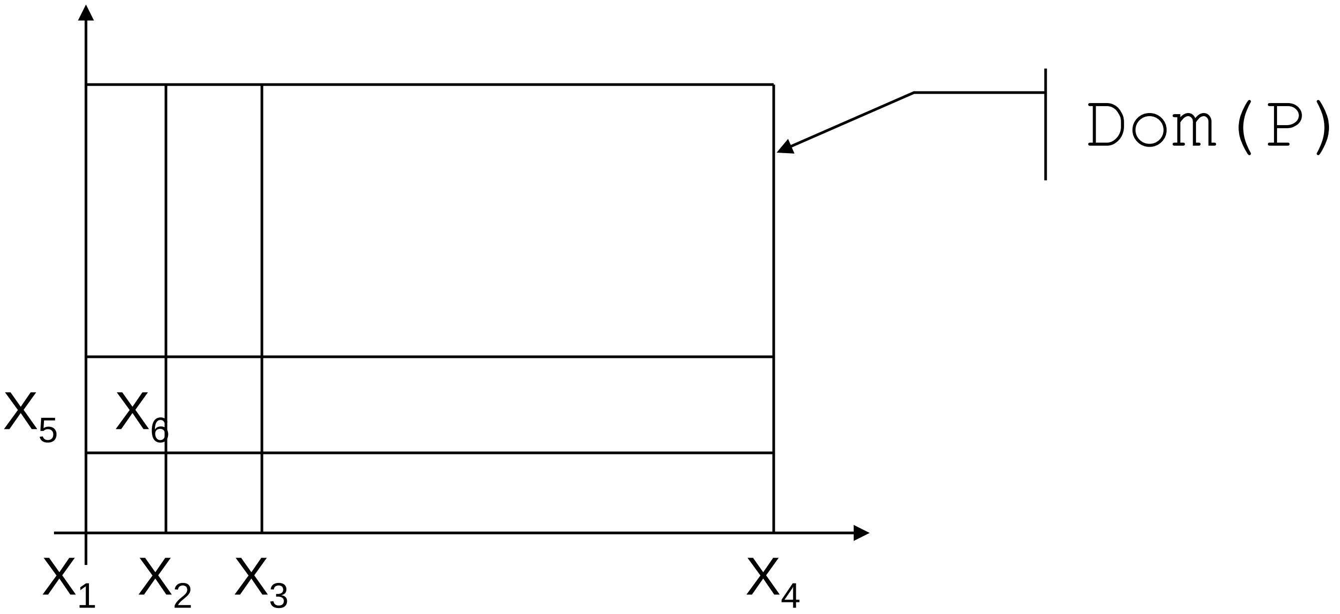
$$P\,(ush\ \ x_1,\ \ ush\ \ x_2,\ \ ush\ \ x_3) \qquad \{\ \ldots\ \}$$

$2^{32} \times 2^{32} \times 2^{32}$ values = 2^{96} distinct test inputs

- Interesting estimation of the size of the input search space, against a test objective

  Test Objective Example: To reach a selected instruction in the code

Testing by Sampling



Weak version of exhaustive testing

Examples :

$\{0, 1, 2, 2^{32}-1\}$ pour un ush

{NaN, -INF,-3.40282347e+38, -1.17549435e-38, -1.0, -0.0,.. }

for a 32-bit floating-point number (IEEE 754)

Random Testing

Uniform probability distribution on the program input space

(i.e., each test input is equi-probable)

- Using **pseudo-random generators**

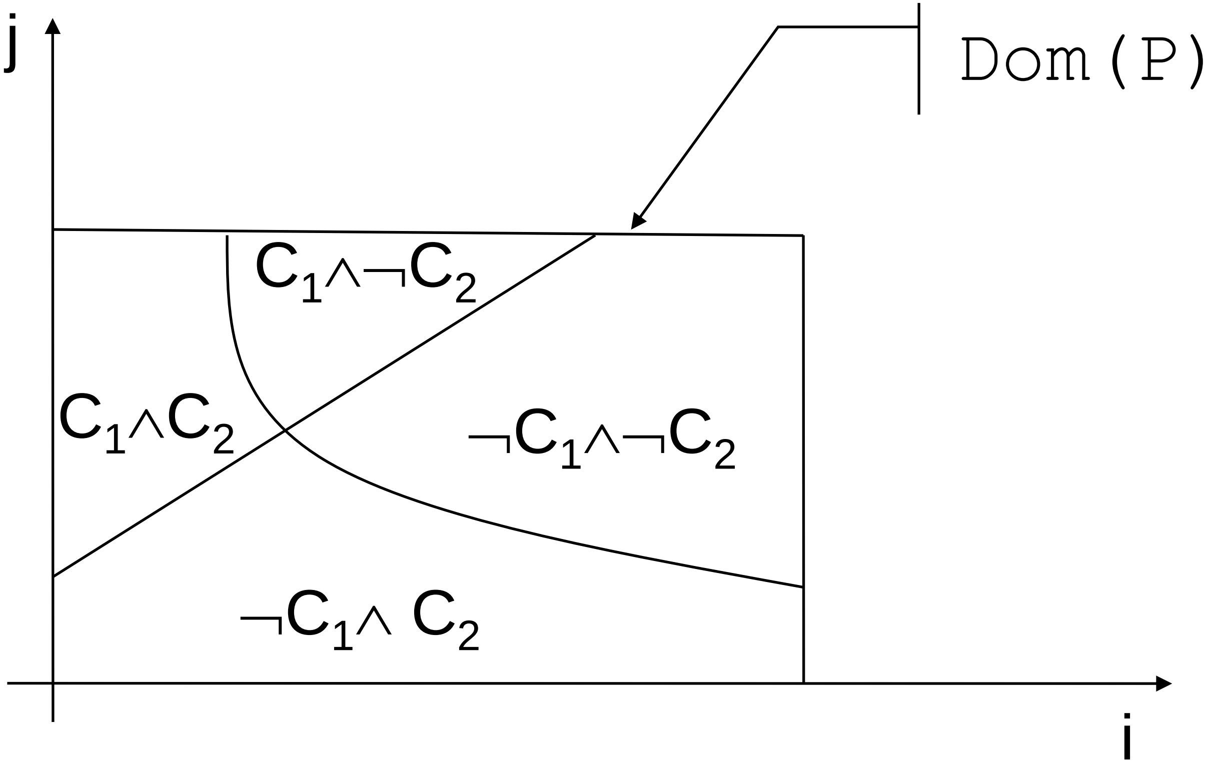
- Require an **automated oracle** (e.g., Metamorphic Testing)

- Stopping criteria must be fixed (number of test inputs, covering a structural criterion, time-out, etc.)

Selection Criterion C

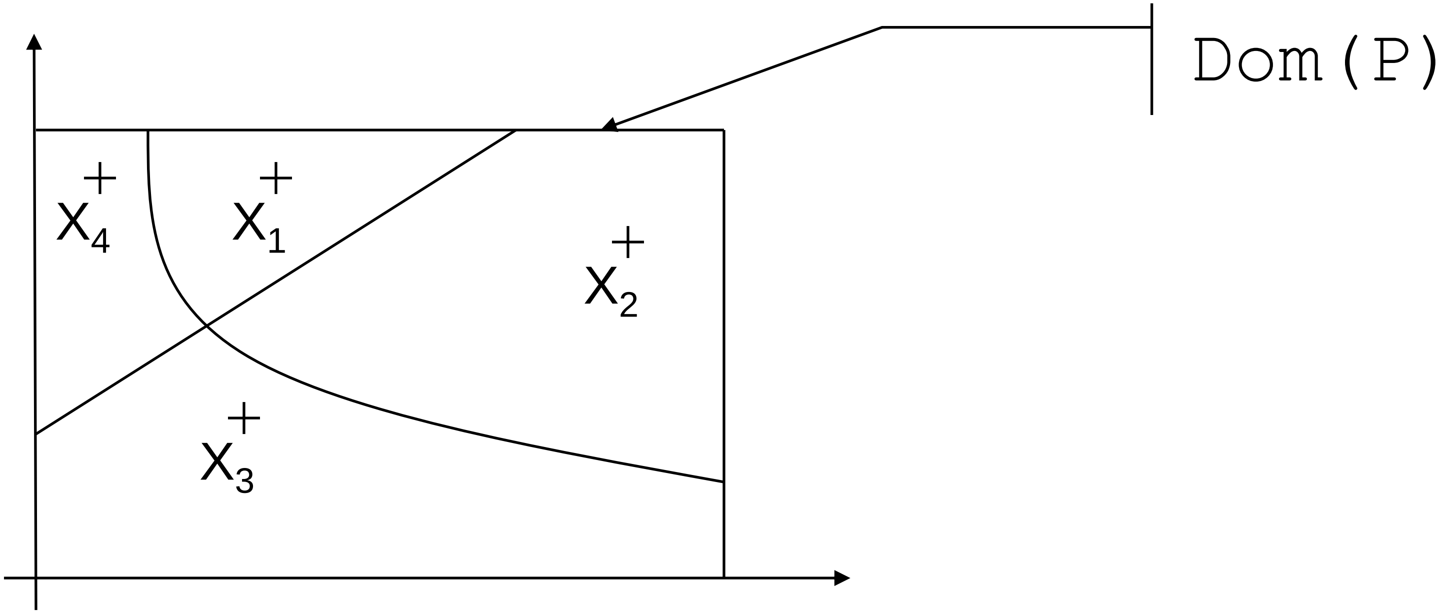
- Process of test inputs selection
- Sometimes, it induces a « partition » over the program input space (e.g., All_paths of P)

```
P(int i,int j)
{
if( C1 )
else ...

if( C2)
else ...
}
```



Deterministic Coverage of Criterion C

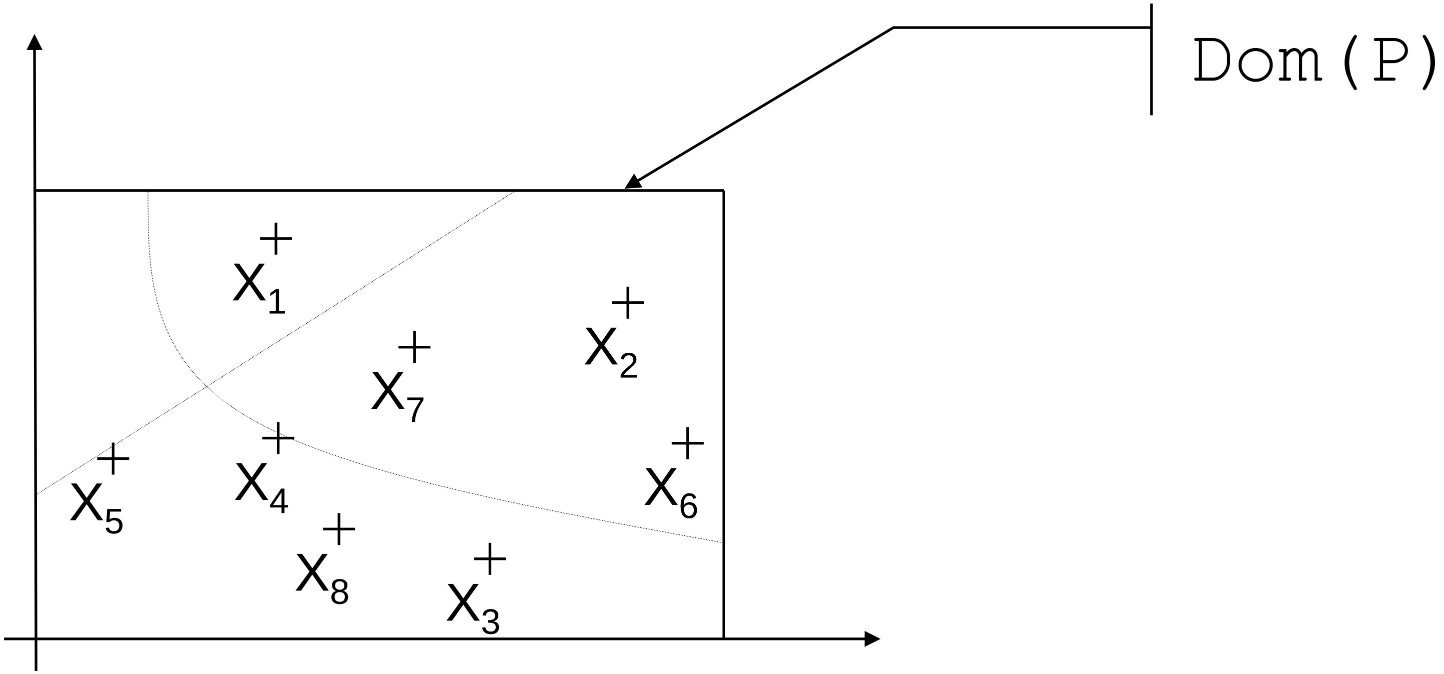
Selection of at least one element per subdomain of the partition



Based on the uniformity assomption that a single input is sufficient to test the whole subdomain

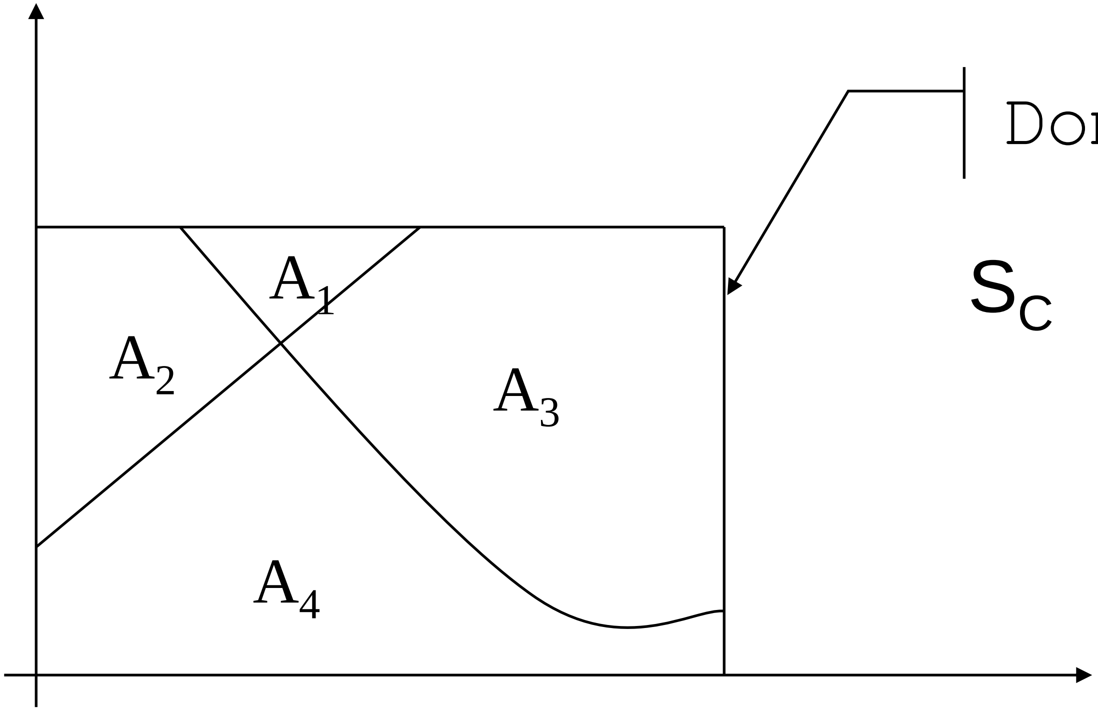
Probabilistic Coverage of Criterion C

Random selection of test inputs according to a distribiution profile



Is Random Testing Efficient to Cover a Criterion?

`p{x∈A}`: probability that a random test input x covers an element A



$S_C = \{A_1, .., A_4\}$

Here $p\{x \in A_1\} < p\{x \in A_2\} < p\{x \in A_3\} < p\{x \in A_4\}$
Hence, random testing covers better A_4 than A_1

RT is well adapted to test the program robustness, but hill-conditioned to test corner-cases

Symbolic execution

Symbolic state: <Path, State, Path Conditions>

Path = n_i-..-n_j is a path expression of the CFG

State = $<v_i, \varphi_i>_{v \in Var(P)}$ where φ_i is an algebraic expression over **X**

Path Cond. = $c_1,..,c_n$ where c_i is a condition over **X**

X denotes symbolic variables associated to the program inputs and Var(P) denotes internal variables

Symbolic execution

Ex : a-b-(c-b)^2-d-f with X,Y

<a, <z,1.>, <w,abs(Y)>, true >

<a-b, <z,1.>, <w,abs(Y)>, abs(Y) != 0 >

<a-b-c, <z,X>, <w,abs(Y)-1>, abs(Y) != 0 >

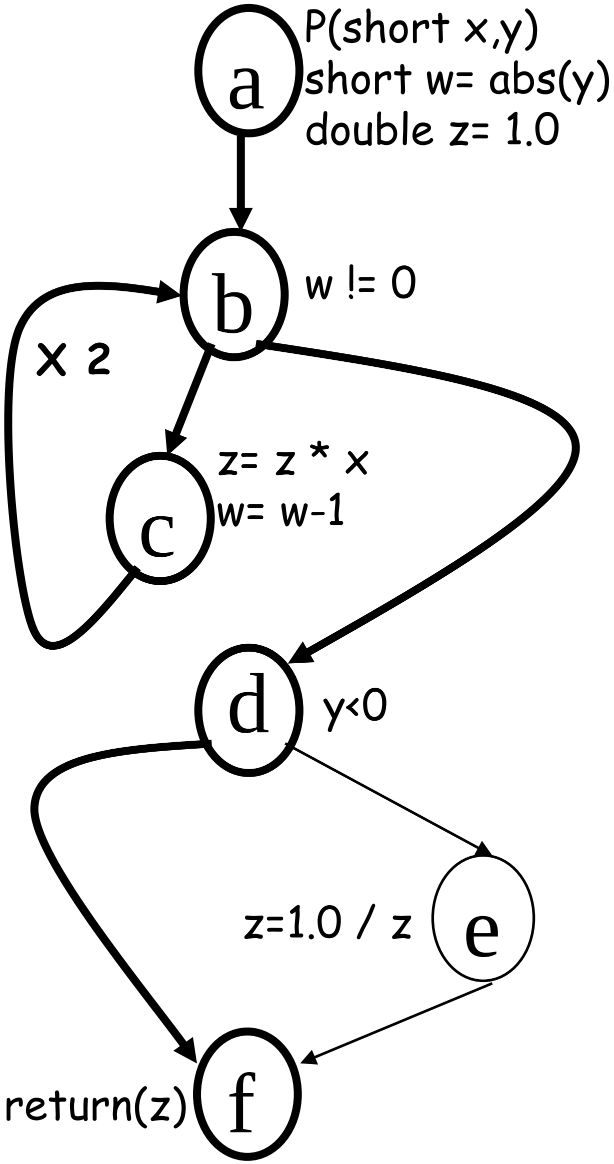
<a-b-c-b, <z,X.>, <w,abs(Y)-1>,
abs(Y) != 0, abs(Y)-1 != 0 >

< a-b-c-b-c, <z,X^2>, <w,abs(Y)-2>,
abs(Y) != 0, abs(Y)-1 != 0 >

<a-b-(c-b)^2, <z,X^2>, <w,abs(Y)-2>,
abs(Y) != 0, abs(Y) != 1, abs(Y)–2 = 0 >

<a-b-(c-b)^2-d, <z,X^2>, <w,abs(Y)-2>,
abs(Y) != 0, abs(Y) != 1, abs(Y) = 2, Y $\geq$ 0 >

<a-b-(c-b)^2-d-f, <z,X^2>, <w,0>, Y=2 >



Computing Symbolic States

- <Path, State, PC> is computed by induction over each statement of Path

- When the Path conditions are unsatisfiable then Path is non-feasible and reciprocally (i.e., symbolic execution captures the concrete semantics)

  ex : <~~a-b-d-e-f,{…}~~, $\mathbf{abs(Y)=0} \wedge \mathbf{Y<0}$ >

- Forward vs backward analysis:

  Forward → interesting when states are needed
  Backward → saves memory space, as complete states are not computed

Backward analysis

Ex : a-b-(c-b)^2-d-f with X,Y

f,d: **Y ≥0**

b: **Y ≥0,** w = 0

c: **Y ≥0,** w-1 = 0

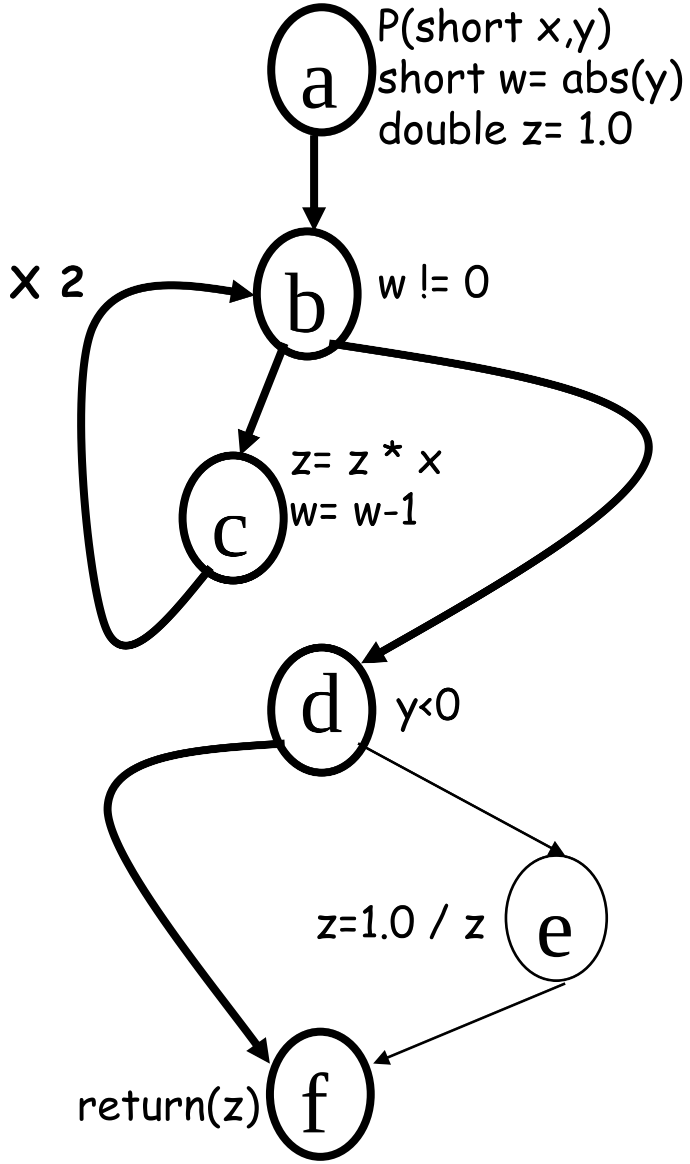
b: **Y ≥0,** w-1 = 0, w != 0

c: **Y ≥0,** w-2 = 0, w-1 != 0

b: **Y ≥0,** w-2 =0, w-1 != 0,w != 0

a: **Y ≥0, abs(Y)-2 = 0,**
abs(Y)-1 != 0, abs(Y) != 0

Y = 2



Constraint Solving in Symbolic Evaluation

- Mixed Integer Linear Programming approaches
  (i.e., simplex + Fourier's elimination + branch-and-bound)

CLP(R,Q) in **ATGen**	(Meudec 2001)
lpsolve in **DART/CUTE**	(Godefroid/Sen et al. 2005)

- SMT-solving (= SAT + Theories)

STP in **EXE and KLEE**	(Cadar et al. 2006)
Z3 in **PEX and SAGE**	(Tillmann and de Halleux 2008)

- Constraint Programming techniques (constraint propagation and labelling)

Colibri in **PathCrawler**	(Williams et al. 2005)
Disolver in **SAGE**	(Godefroid et al. 2008)
EUCLIDE	**(Gotlieb 2009)**
ECLAIR	(Bagnara Bagnara Gori 2013)

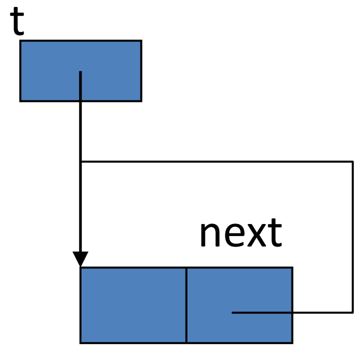
Problems for Symbolic Evaluation Techniques

→ Combinatorial explosion of paths

→ Symbolic execution constrains the shape of dynamically allocated objects

```
int P(struct cell * t) {
   if( t == t->next ) { …
```

constrains t to:



→ Floating-point computations ➪

*F Charreteur, B Botella, A Gotlieb. **Modelling dynamic memory management in constraint-based testing**. Journal of Systems and Software. Elsevier, 2009*

```
float foo( float x) {
   float y = 1.0e12, z ;
1. if( x < 10000.0 )
2.      z = x + y;
3. if( z > y)
4.      …
```

Is the path 1-2-3-4 feasible ?

Path conditions:

$x < 10000.0$

$x + 1.0e12 > 1.0e12$

On the reals : $x \in (0,10000)$

On the floats : no solution !

Conversely,

```
float foo( float x) {
   float y = 1.0e12, z ;
1.  if( x > 0.0 )
2.      z = x + y;
3.  if( z == y)
4.      …
```

Is the path 1-2-3-4 feasible ?

Path conditions:

$x > 0.0$

$x + 1.0e12 = 1.0e12$

On the reals : no solution

On the floats: $x \in (0, 32767.99\ldots)$

Solution: build a dedicated constraint solver over the floats !

B Botella, A Gotlieb, C Michel. **Symbolic execution of floating-point computations**. STVR 2006

R Bagnara, M Carlier, R Gori, A Gotlieb. **Symbolic path-oriented test data generation for floating-point programs**. IEEE ICST 2013

Dynamic Symbolic Evaluation (DSE)

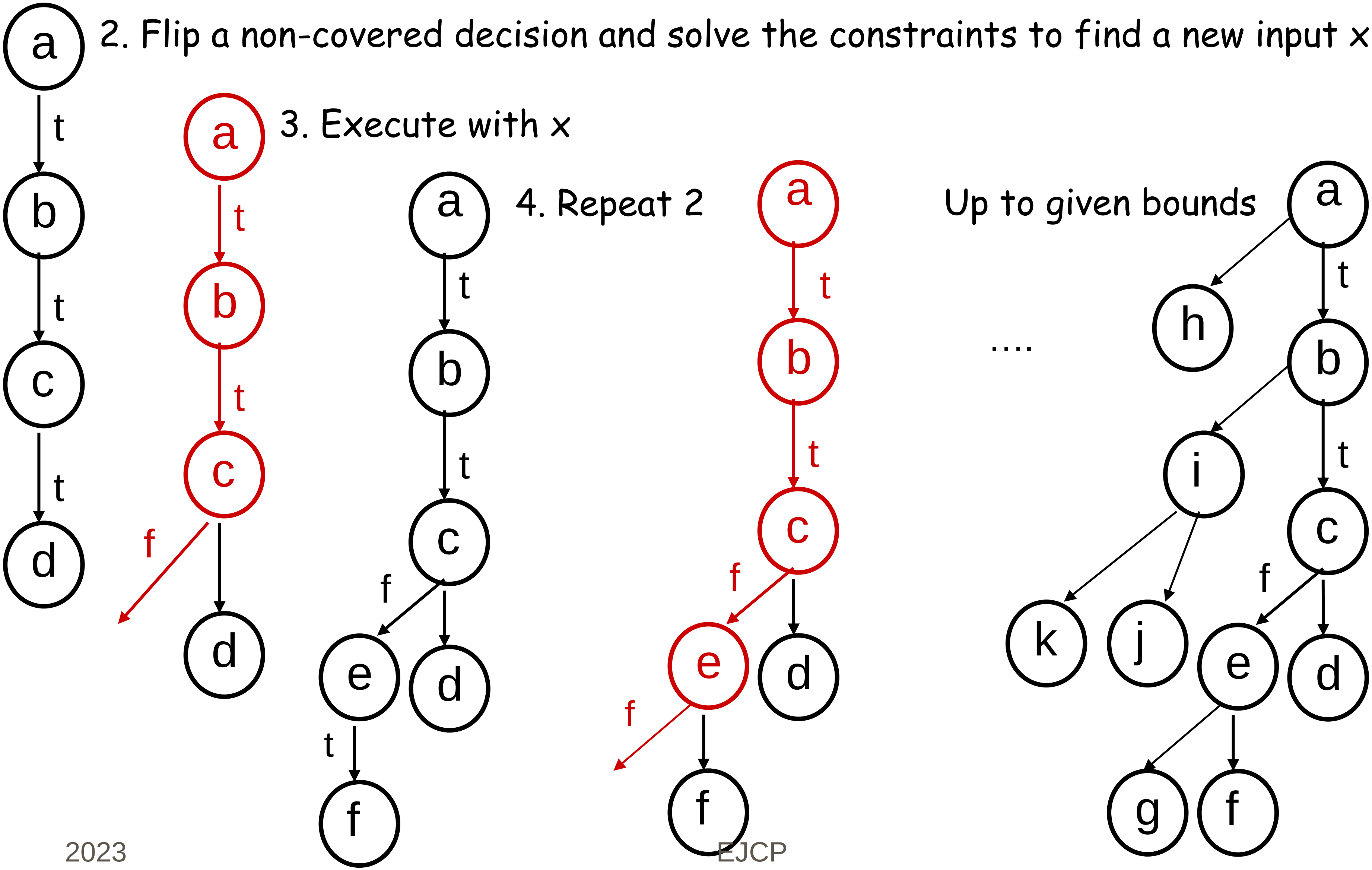
- Symb<u>olic</u> execution of a <u>con</u>crete execution (also called <u>concolic</u> execution)
- By using input values, **feasible paths only** are (automatically) selected
- Randomized algorithm, implemented by instrumenting each statement of P

<u>Main CBT tools:</u>

PathCrawler (Williams et al. 2005),
PEX (Tillman et al. Microsoft 2008),
SAGE (Godefroid et al.2008)
KLEE (Cadar et al. 2008)

Dynamic Symbolic Execution for All-k-paths

1. Draw an input at random, execute it and record path conditions



2. Flip a non-covered decision and solve the constraints to find a new input x

3. Execute with x

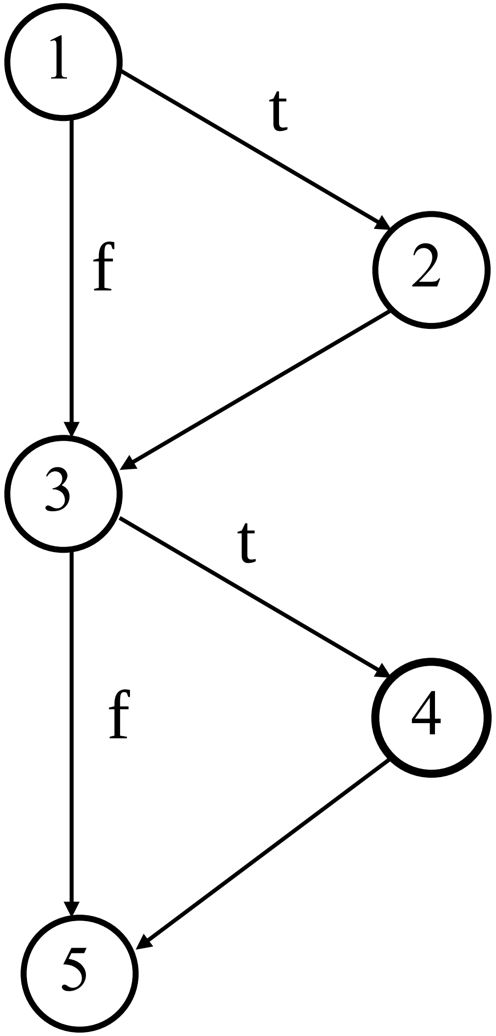
4. Repeat 2

….

Up to given bounds

Example (1)

```
f(  int i  )
{
   j = 2;
   if( i ≤ 16 )
       j = j * i;

   if( j > 8)
       j = 0;

   return j;
  }
```



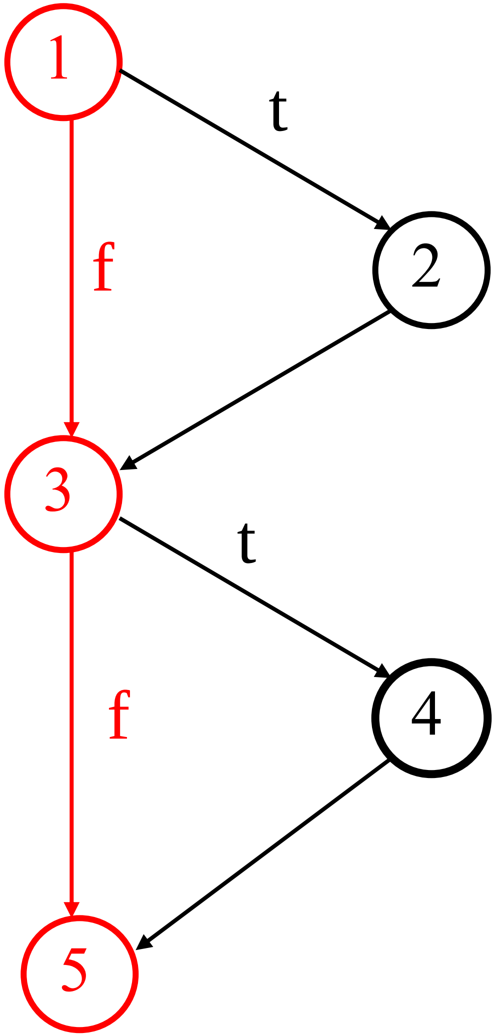
Example (2)

```
f(  int i  )
{
   j = 2;
   if( i ≤ 16 )
       j = j * i;

   if( j > 8)
      j = 0;

   return j;
  }
```

Random imput generation

(i = 15448)

→ Path 1-3-5



Example (3)

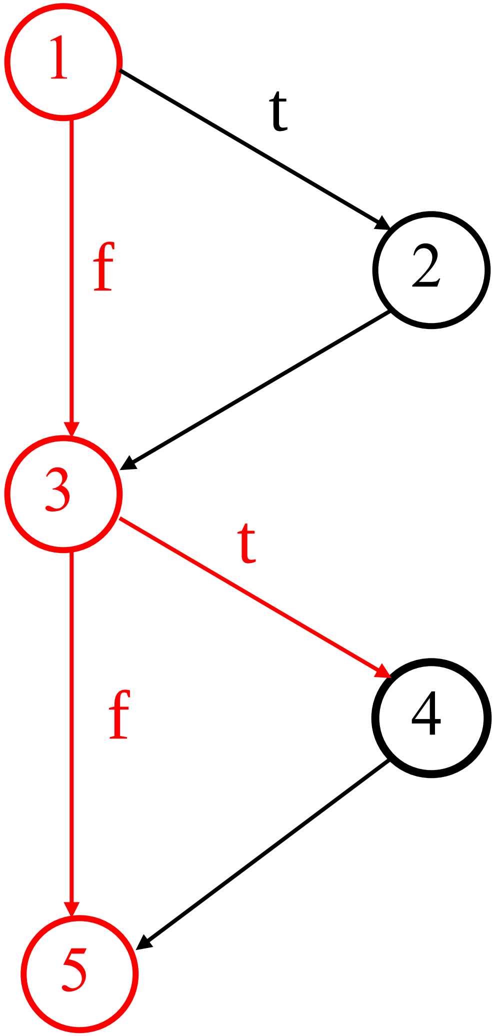
```
f(  int i  )
{
   j = 2;
   if( i ≤ 16 )
       j = j * i;

   if( j > 8)
      j = 0;

   return j;
  }
```

Try to solve

$j_1=2$
$i > 16$

$j_1 > 8$

Unsatisfiable, therefore
Path 1-3-4 is non-feasible



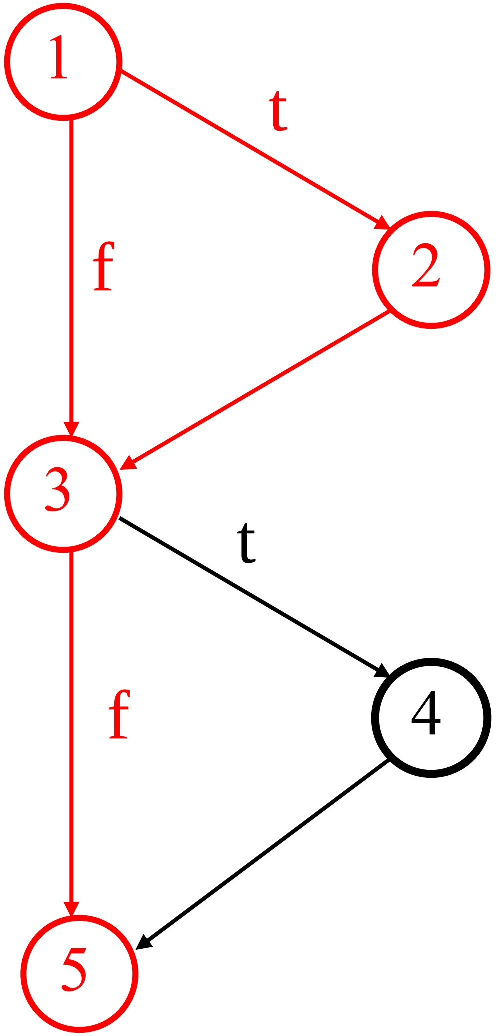
Example (4)

```
f(  int i  )
{
   j = 2;
   if( i ≤ 16 )
       j = j * i;

   if( j > 8)
      j = 0;

   return j;
  }
```

Bactrack and try to solve

j_1=2
i <= 16

→ (i = 2) -- Path 1-2-3-5



Example (5)

```
f(  int i  )
{
   j = 2;
   if( i ≤ 16 )
       j = j * i;

   if( j > 8)
       j = 0;

   return j;
  }
```

Bactrack and try to solve

$j_1 = 2$

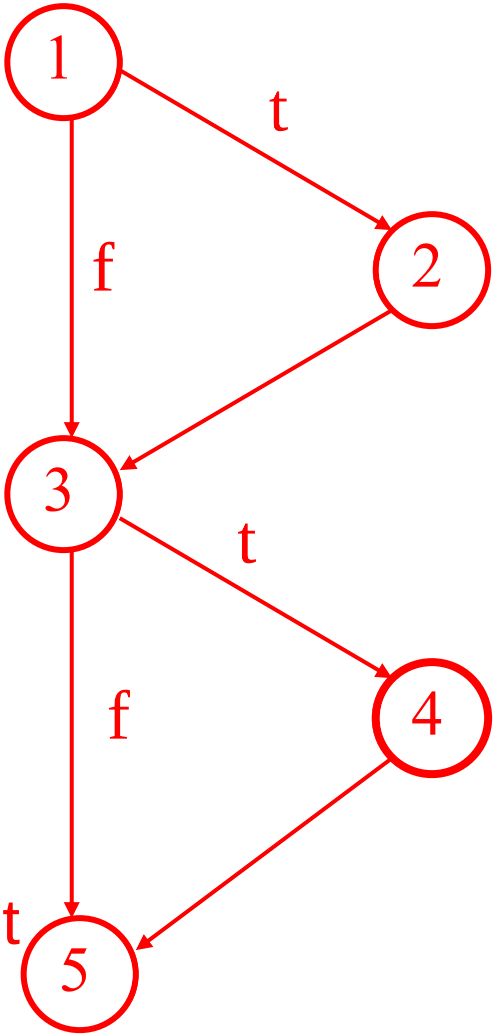
$i <= 16$

$j_2 = j_1 * i$

$j_2 > 8$

→(i = 10) -- Path 1-2-3-4-5

All-paths covered with three test data (i = 15448, i = 2, i = 10)

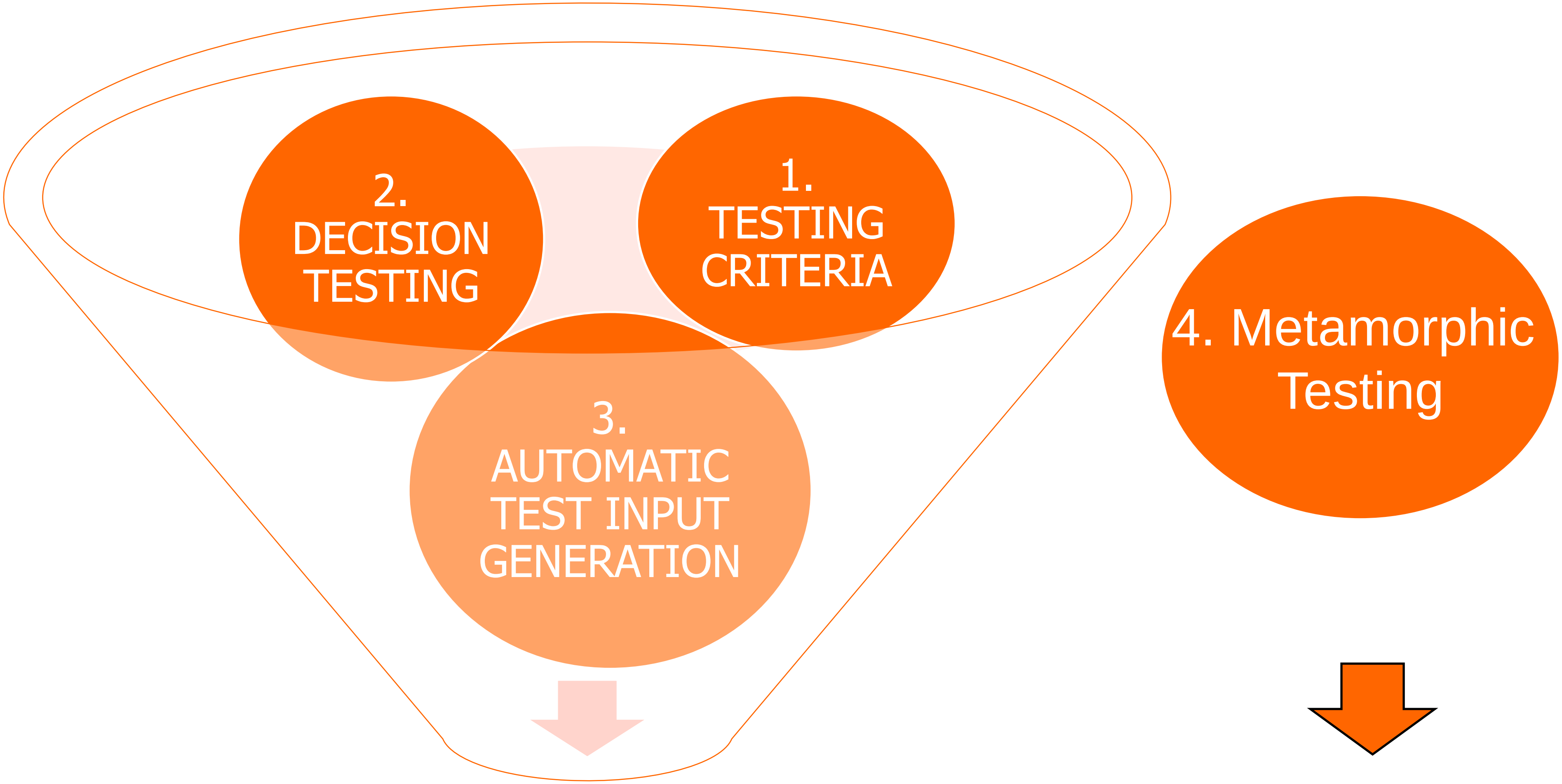


Dynamic Symbolic Execution: Discussion

- Requires to bound the number of iterations in loops
  → suitable for automatic test data generation
  for the **All-k-paths criterion**

- Performance of the method depends on the first initial random input

- Numerous extensions to handle pointers as input parameters, logical decisions, function calls, bit-to-bit operations



2. DECISION TESTING

1. TESTING CRITERIA

3. AUTOMATIC TEST INPUT GENERATION

4. Metamorphic Testing

Code-Based Testing

Oracle Problem

4. Metamorphic Testing

Non-testable programs

[Weyuker TSE 82]

← No (complete and correct) oracle available

Because

- No formal specifications, incomplete specifications;
- Expected results too difficult to compute;
- Inferred/generalized from a set of instances;
- Depending on the execution environment;

Typical examples:

Third-party library functions, RESTful APIs
Complex mathematical functions (using floating-point computations)
Trained ML models
Optimization programs (optimal planners, assignment, scheduling, etc.)
Reactive and self-adaptive programs

Metamorphic Testing [Chen *et al.* 98]

Metamorphic Relation (MR) of a program P:

User-specified input-output relation about P

Let's start with a trivial example:

P : a program that implements the *gcd* of 2 integers

Problem: P(1309, 693) = ?

MR: $\forall$u, $\forall$v, *gcd*(u, v) = *gcd*(v, u)

Hence, if P(1309, 693) $\neq$ P(693, 1309) then verdict = Fail

* Note that many other programs than gcd satisfy P(u, v) = P(v, u) so,

MRs are necessary, but not sufficient to establish program correctness

** Note also that there are many other possible MRs

MR: $\forall$u, $\forall$v, *gcd*(p.u, p.v) = p. *gcd*(u, v) if p is a prime number

MR: $\forall$u, $\forall$v *gcd*(u, v) = *gcd*(v, u-v) if u > v

Graph Theory

How to test a program P that computes a shortest path in an undirected graph G?

shortestPath(G, a, b) = ?

if P(G, a, b) = a-e1-e2-e3-b and P(G, b, a) = b-g1-g2-a then verdict = Fail

<u>MR:</u> $\forall a, \forall b\ |shortestPath(G, a, b)| = |shortestPath(G, b, a)|$

* Note that MRs can be based on the usage of other functions (possibly under test)

** Note also that MRs can involve more than one additional computation

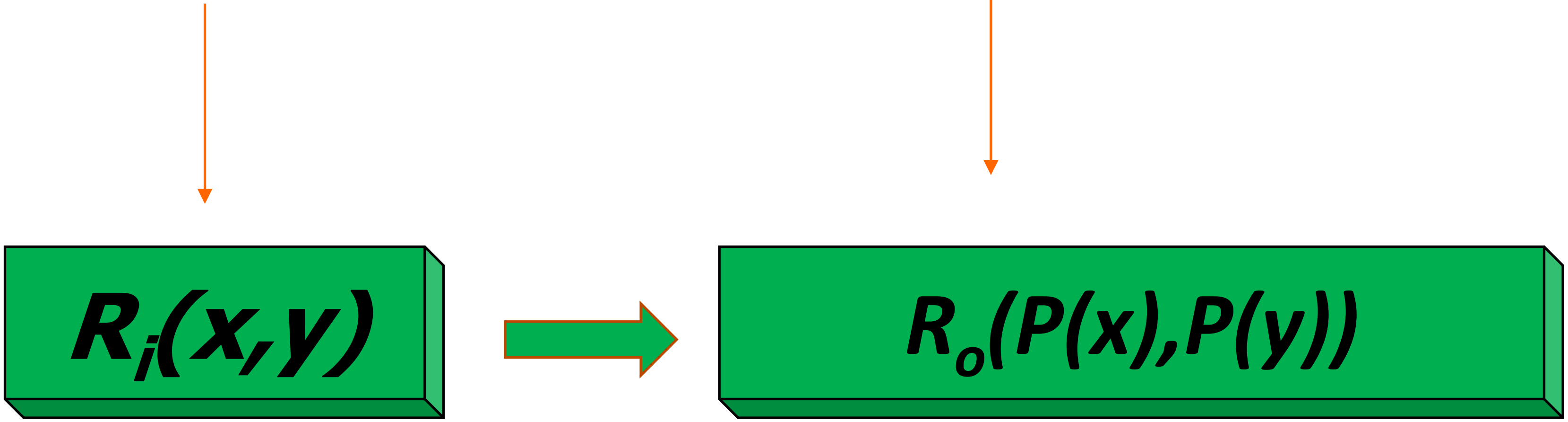
<u>MR:</u> $|shortestPath(G, a,b)| \leq |shortestPath(G, a, c)| + |shortestPath(G, c, b)|$

Search Engines

if search(“tom” OR ”jerry”) returns less items than search(“tom” AND ”jerry”)
then verdict = Fail

MR: $\forall$k1, $\forall$k2 |search(k1 OR k2)| ≥ |search(k1 AND k2)|

x: (k1 OR k2), y: (k1 AND k2) implies |search(x)| ≥ |search(y)|

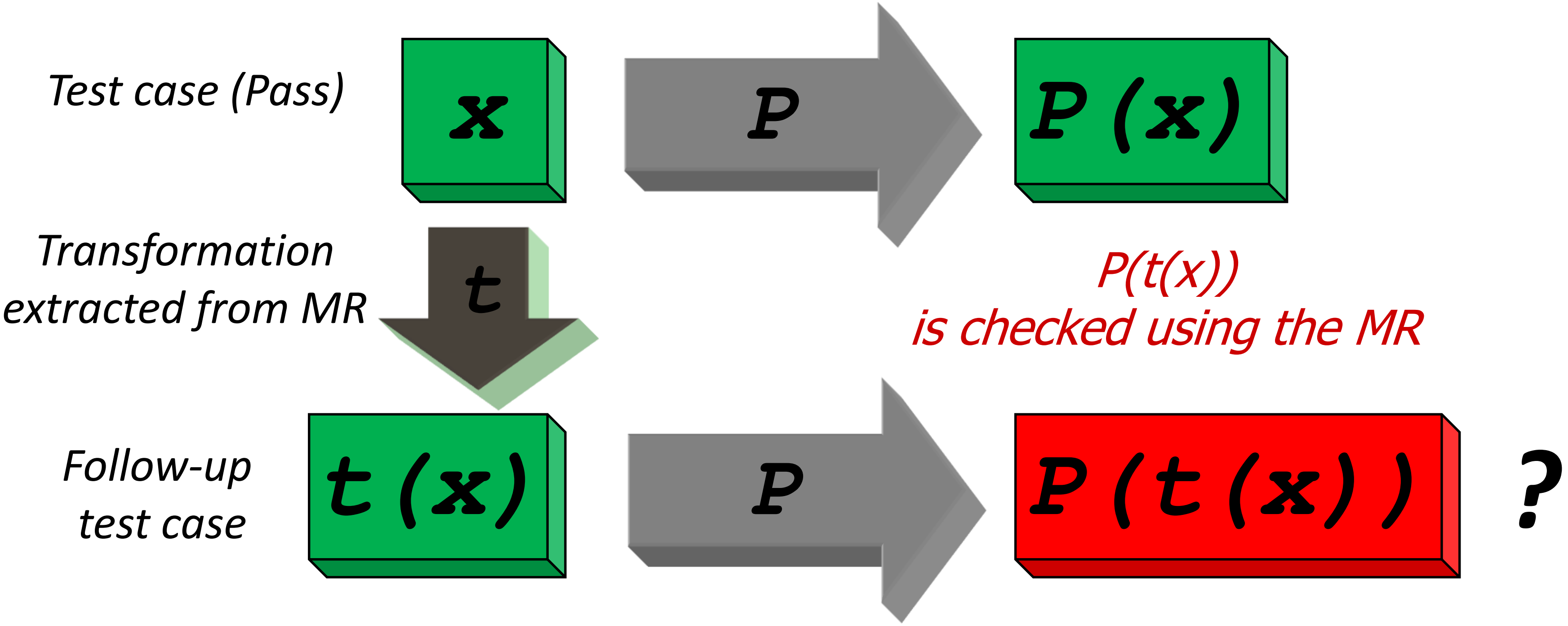


$R_i(x,y)$

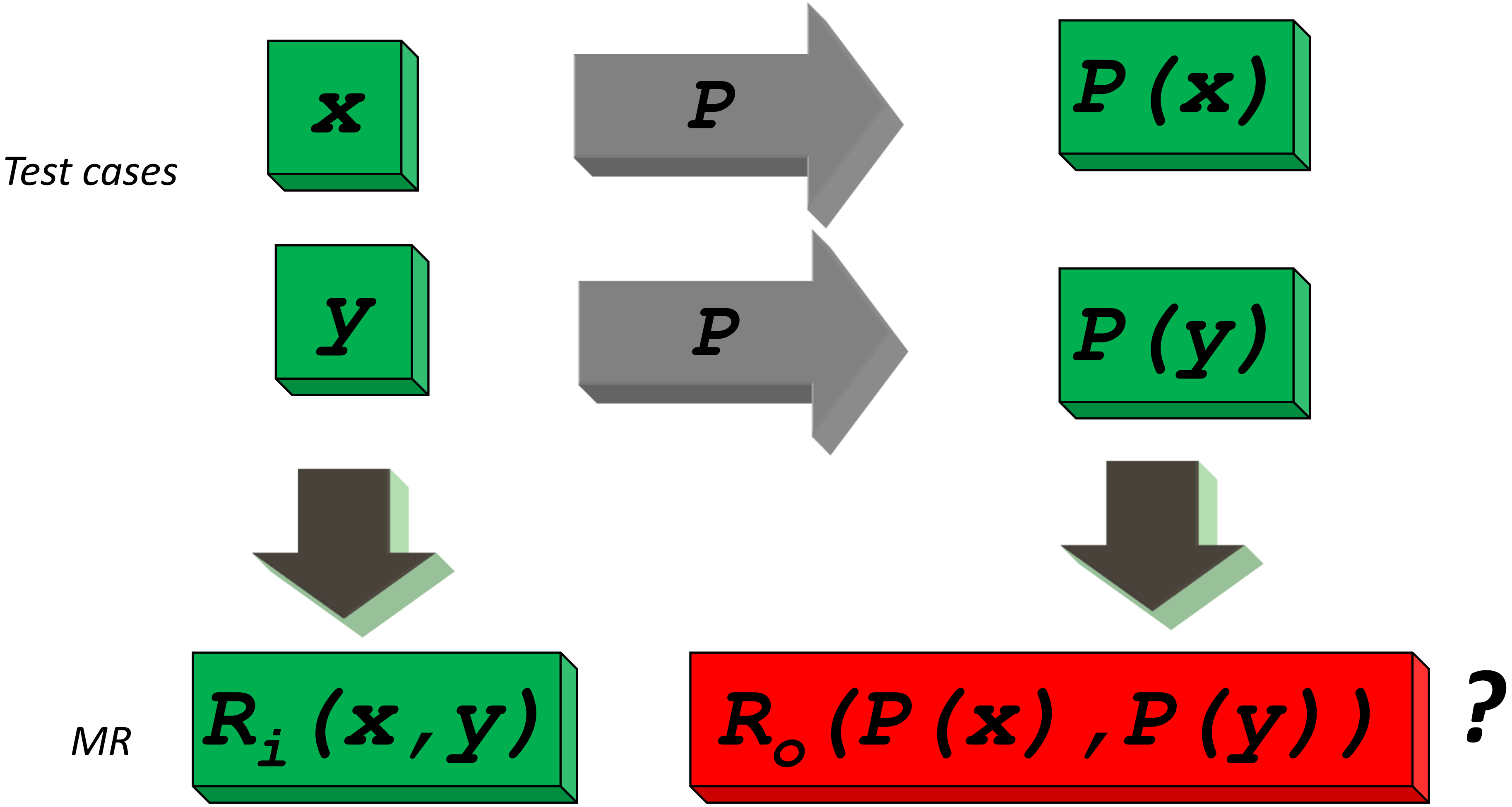
$R_o(P(x),P(y))$

Main Usages

1. To generate follow-up test cases



2. To create partial oracles



Test cases

x

y

P

P

$P(x)$

$P(y)$

MR

$R_i(x,y)$

$R_o(P(x),P(y))$?

Strategies for Finding Metamorphic Relations

1) Driven by transformation over input-data

Which transformations t over the inputs x do not change the outcome of P?

i.e., **find t such that P(x) = P(t(x))**

Transformations t: add, remove or reorder elements, perturb inputs, shift or rotate images, …

2) Driven by output-relation

Given two executions of P, what kind of relations do exist between these executions ?

i.e., Given x,y, P(x), P(y), **find R_o(P(x), P(t(x))**

Relations R_o: less_or_equal, length, subset, equivalent,…

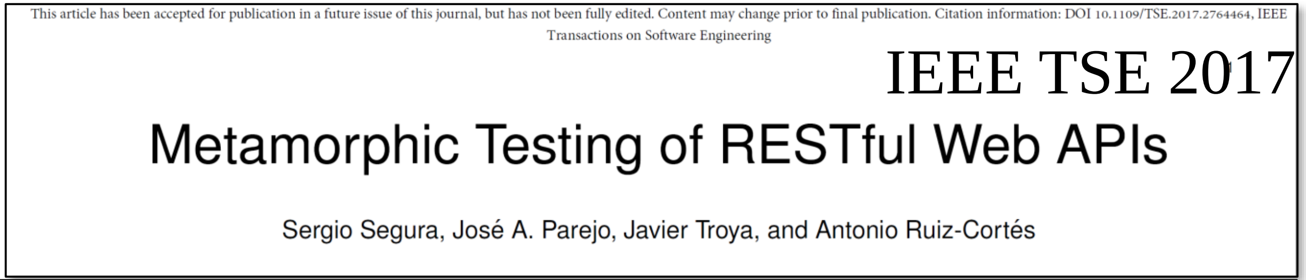
3) Driven by domain-knowledge

Which invariant properties P has to satisfy ?

4)…

Applications of MT (1/3)

Testing online search engines (Flickr, Youtube, Spotify,...)



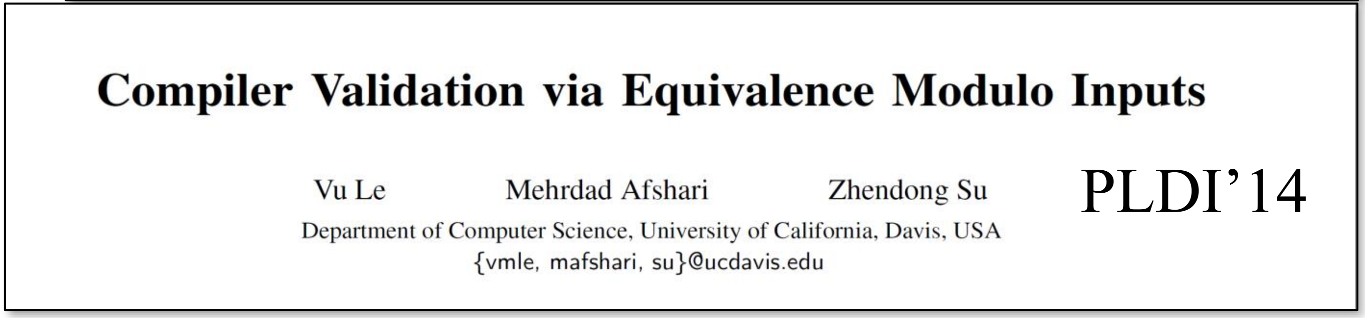
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2017.2764464, IEEE Transactions on Software Engineering

IEEE TSE 2017

Metamorphic Testing of RESTful Web APIs

Sergio Segura, José A. Parejo, Javier Troya, and Antonio Ruiz-Cortés

Testing compilers



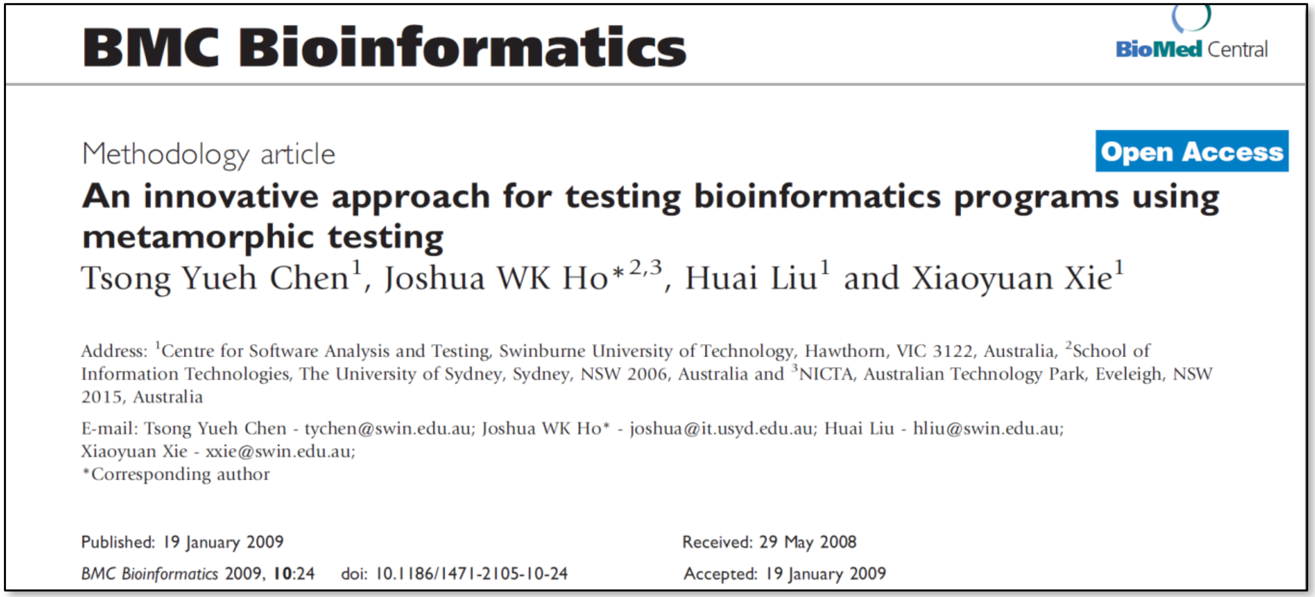
Compiler Validation via Equivalence Modulo Inputs

Vu Le Mehrdad Afshari Zhendong Su

Department of Computer Science, University of California, Davis, USA
{vmle, mafshari, su}@ucdavis.edu

PLDI'14

Testing bioinformatics programs (Genes Regulat. Net. simulation)



BMC Bioinformatics

BioMed Central

Methodology article

Open Access

An innovative approach for testing bioinformatics programs using metamorphic testing

Tsong Yueh Chen[1], Joshua WK Ho*[2,3], Huai Liu[1] and Xiaoyuan Xie[1]

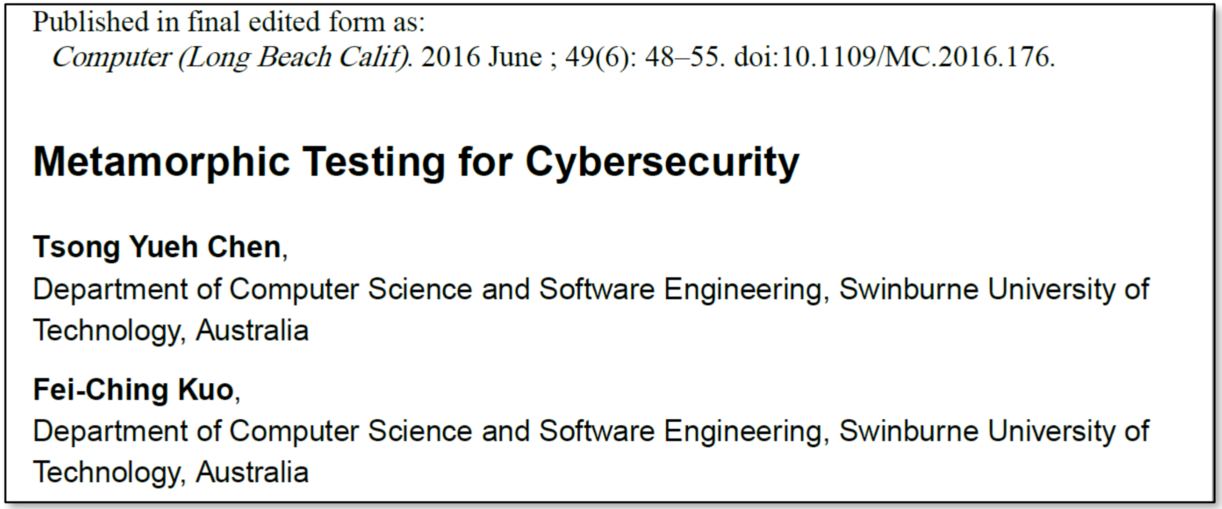
Address: [1]Centre for Software Analysis and Testing, Swinburne University of Technology, Hawthorn, VIC 3122, Australia, [2]School of Information Technologies, The University of Sydney, Sydney, NSW 2006, Australia and [3]NICTA, Australian Technology Park, Eveleigh, NSW 2015, Australia

E-mail: Tsong Yueh Chen - tychen@swin.edu.au; Joshua WK Ho* - joshua@it.usyd.edu.au; Huai Liu - hliu@swin.edu.au; Xiaoyuan Xie - xxie@swin.edu.au;
*Corresponding author

Published: 19 January 2009 Received: 29 May 2008
BMC Bioinformatics 2009, **10**:24 doi: 10.1186/1471-2105-10-24 Accepted: 19 January 2009

Applications of MT (2/3)

Testing code obfuscators, testing web interfaces, penetration testing



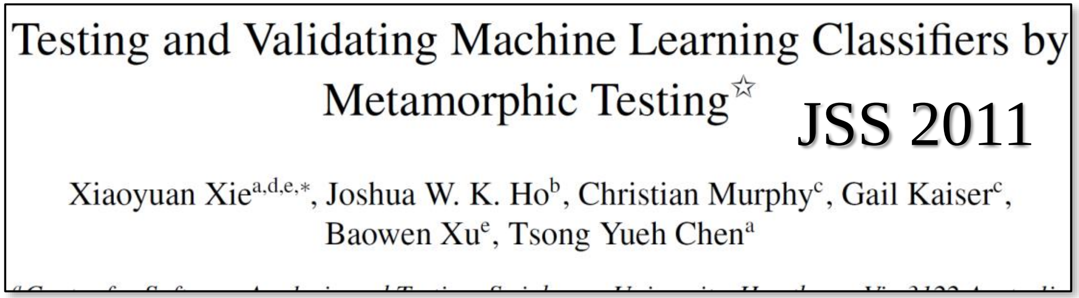
Published in final edited form as:
Computer (Long Beach Calif). 2016 June ; 49(6): 48–55. doi:10.1109/MC.2016.176.

Metamorphic Testing for Cybersecurity

Tsong Yueh Chen,
Department of Computer Science and Software Engineering, Swinburne University of Technology, Australia

Fei-Ching Kuo,
Department of Computer Science and Software Engineering, Swinburne University of Technology, Australia

Testing simple ML models



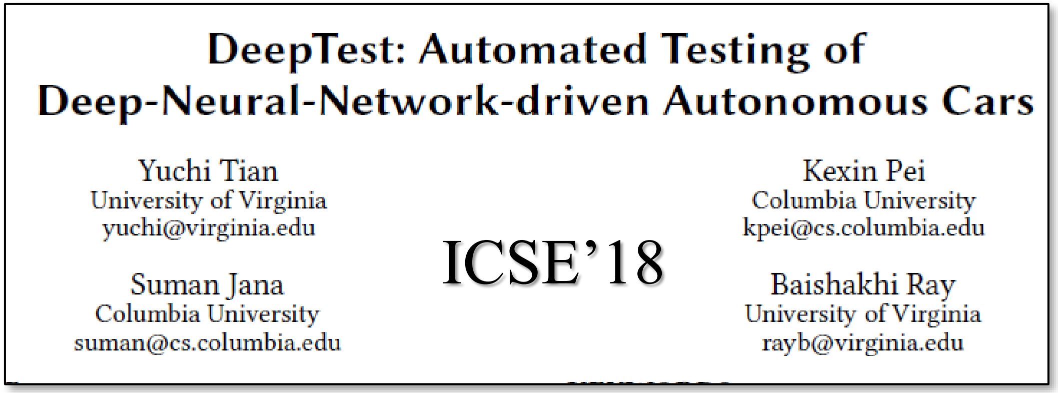
Testing and Validating Machine Learning Classifiers by Metamorphic Testing☆

JSS 2011

Xiaoyuan Xie[a,d,e,*], Joshua W. K. Ho[b], Christian Murphy[c], Gail Kaiser[c],
Baowen Xu[e], Tsong Yueh Chen[a]

Applications of MT (3/3)

Testing DNNs in self-driving cars



DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars

Yuchi Tian
University of Virginia
yuchi@virginia.edu

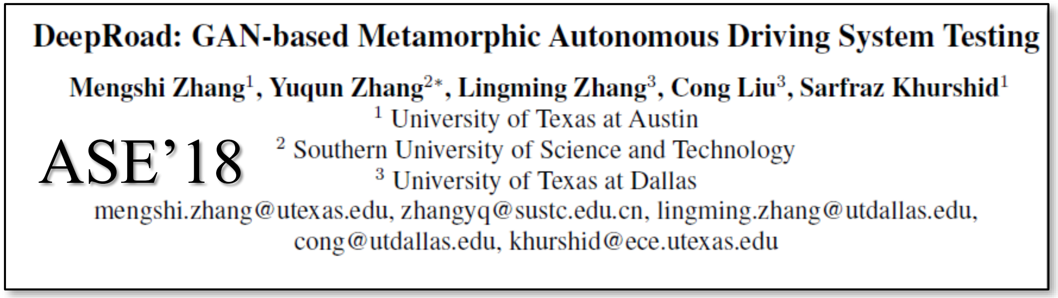
Kexin Pei
Columbia University
kpei@cs.columbia.edu

Suman Jana
Columbia University
suman@cs.columbia.edu

Baishakhi Ray
University of Virginia
rayb@virginia.edu

ICSE'18

Generating driving scenes

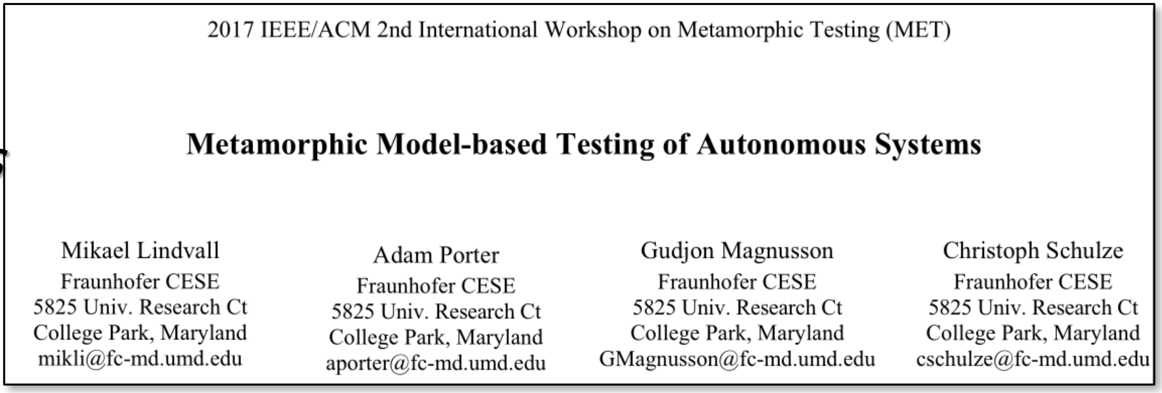


DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing

Mengshi Zhang[1], Yuqun Zhang[2*], Lingming Zhang[3], Cong Liu[3], Sarfraz Khurshid[1]
[1] University of Texas at Austin
[2] Southern University of Science and Technology
[3] University of Texas at Dallas
mengshi.zhang@utexas.edu, zhangyq@sustc.edu.cn, lingming.zhang@utdallas.edu,
cong@utdallas.edu, khurshid@ece.utexas.edu

ASE'18

Testing autonomous drones



2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)

Metamorphic Model-based Testing of Autonomous Systems

Mikael Lindvall
Fraunhofer CESE
5825 Univ. Research Ct
College Park, Maryland
mikli@fc-md.umd.edu

Adam Porter
Fraunhofer CESE
5825 Univ. Research Ct
College Park, Maryland
aporter@fc-md.umd.edu

Gudjon Magnusson
Fraunhofer CESE
5825 Univ. Research Ct
College Park, Maryland
GMagnusson@fc-md.umd.edu

Christoph Schulze
Fraunhofer CESE
5825 Univ. Research Ct
College Park, Maryland
cschulze@fc-md.umd.edu

MT: Pros/Cons

+ Automated powerful testing method

+ Multiple MRs can be combined altogether

+ Lightweight method, easy to setup and deploy (once MRs have been identified)

+ Successful in testing ML models

- Designing MRs often require domain knowledge

- MRs have different fault-revealing capabilities

- Shallow underlying theory, lack of foundations

- Not yet used for systematically testing critical programs

Remaining Challenges

- Lack of foundational theory
- Need for automatic finding and selection of MRs
- MT for performance (execution time, energy consumption) is not yet sufficiently developed
- MT of Collaborative Robots

First Synthesis

- In the industrial world, software systems are mostly validated with **software testing** (no model checking, no correction proof)

- Code-based testing (**Testing criteria**, **MCDC**) has a long-term tradition and it has been popularized with dynamic symbolic execution (**DSE**) which combine coverage and SE

- **Metamorphic Testing** is crucial and fruitful technique to deal with the oracle problem

- Numerous tools, methods and approaches exist. That background cannot be ignored when engaging new research works

- Still, open challenges remain…

Course Overview

- Software Testing Introduction
- Code-based Testing
- Testing of Autonomous Systems
- Open Challenges in Software Testing

Autonomous Software-Systems

- Systems which have a certain degree of self-decision capabilities,
  e.g., self-driving cars, industrial robots, smart transportation systems,…

- Systems with increased capabilities of planning (what, how), scheduling (when, who) and executing complex functions, with limited human intervention, managing unexpected events, such as faults or hazards

- Not equal to automated systems, which have limited capacity to learn and adapt to unexpected events

- In robotics and automated driving, the main focus for autonomy is to complement human's capacity to take decisions based on vast amounts of uncertain raw data

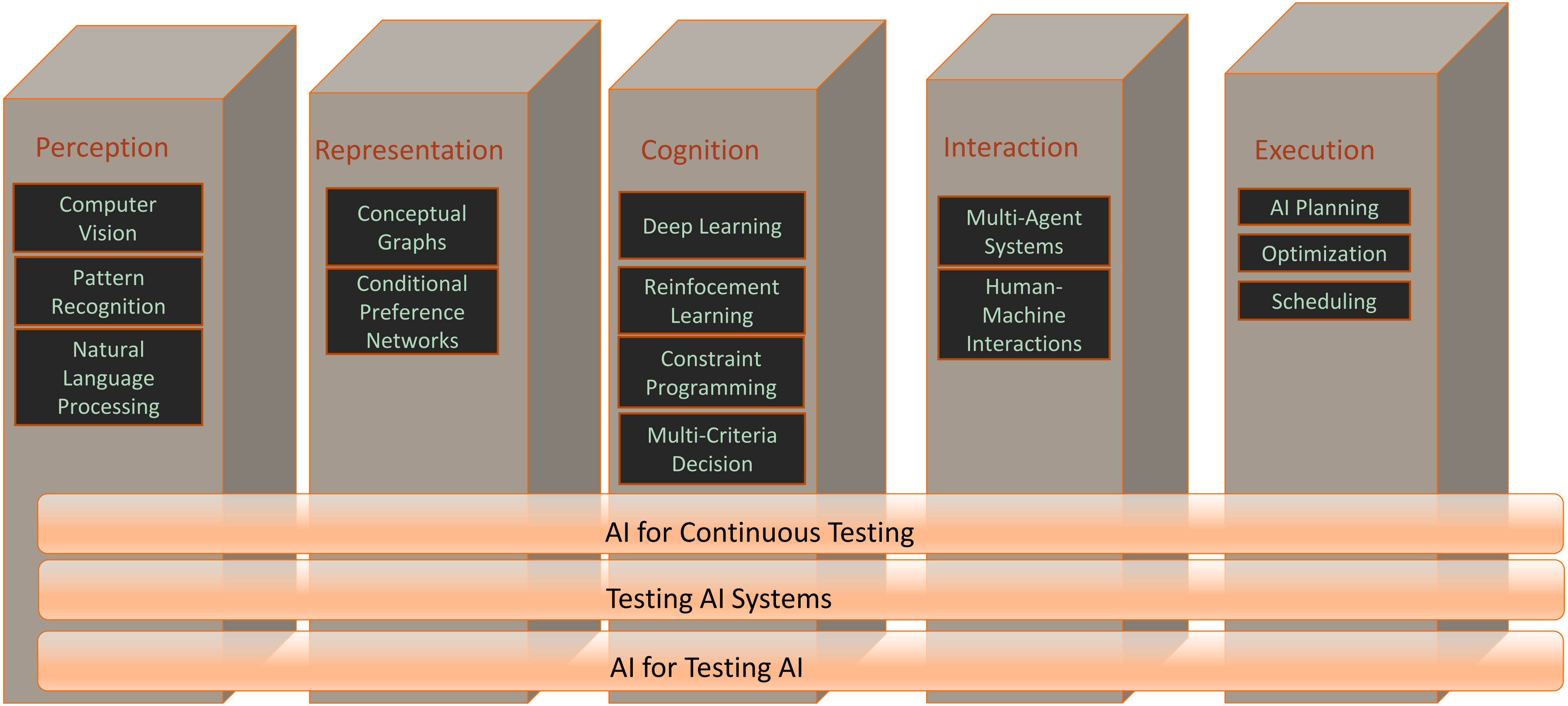

IEEE Spectrum – Self-driving car


Universal Robot – UR3
Cobot


ABB Robotics – YUMI


Kongsberg Maritime – Yara Birkeland

AI in the 5 Pilars of Autonomous Systems



Perception

Computer Vision

Pattern Recognition

Natural Language Processing

Representation

Conceptual Graphs

Conditional Preference Networks

Cognition

Deep Learning

Reinforcement Learning

Constraint Programming

Multi-Criteria Decision

Interaction

Multi-Agent Systems

Human-Machine Interactions

Execution

AI Planning

Optimization

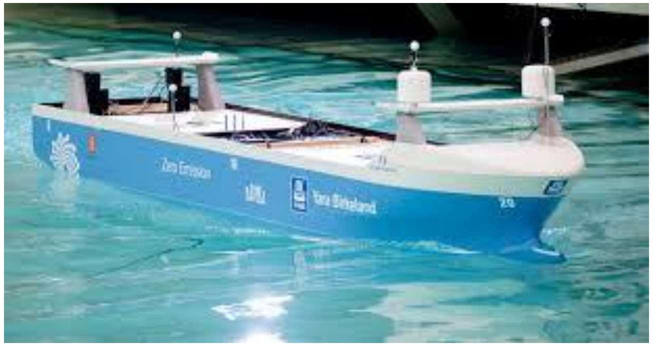
Scheduling

AI for Continuous Testing

Testing AI Systems

AI for Testing AI

Norwegian Yara Birkeland



This electrical autonomous cargo vessel will transport fertiliser from Yara's Porsgrunn plant via inland waterways to the deep-sea ports of Larvik and Brevik (31 nautical miles). Removing up to 40,000 truck journeys annually.

Norwegian shore

The system is based on a seven-axis robotic arm that takes the mooring ropes with loops and wraps them around bollards on the dock. The mooring system has redundant kinematics, with built-in movement compensation and track planning.
The vessel’s position against the quay will inform the robotic arm where each bollard is located, and the track planning is automatically generated by the control system.

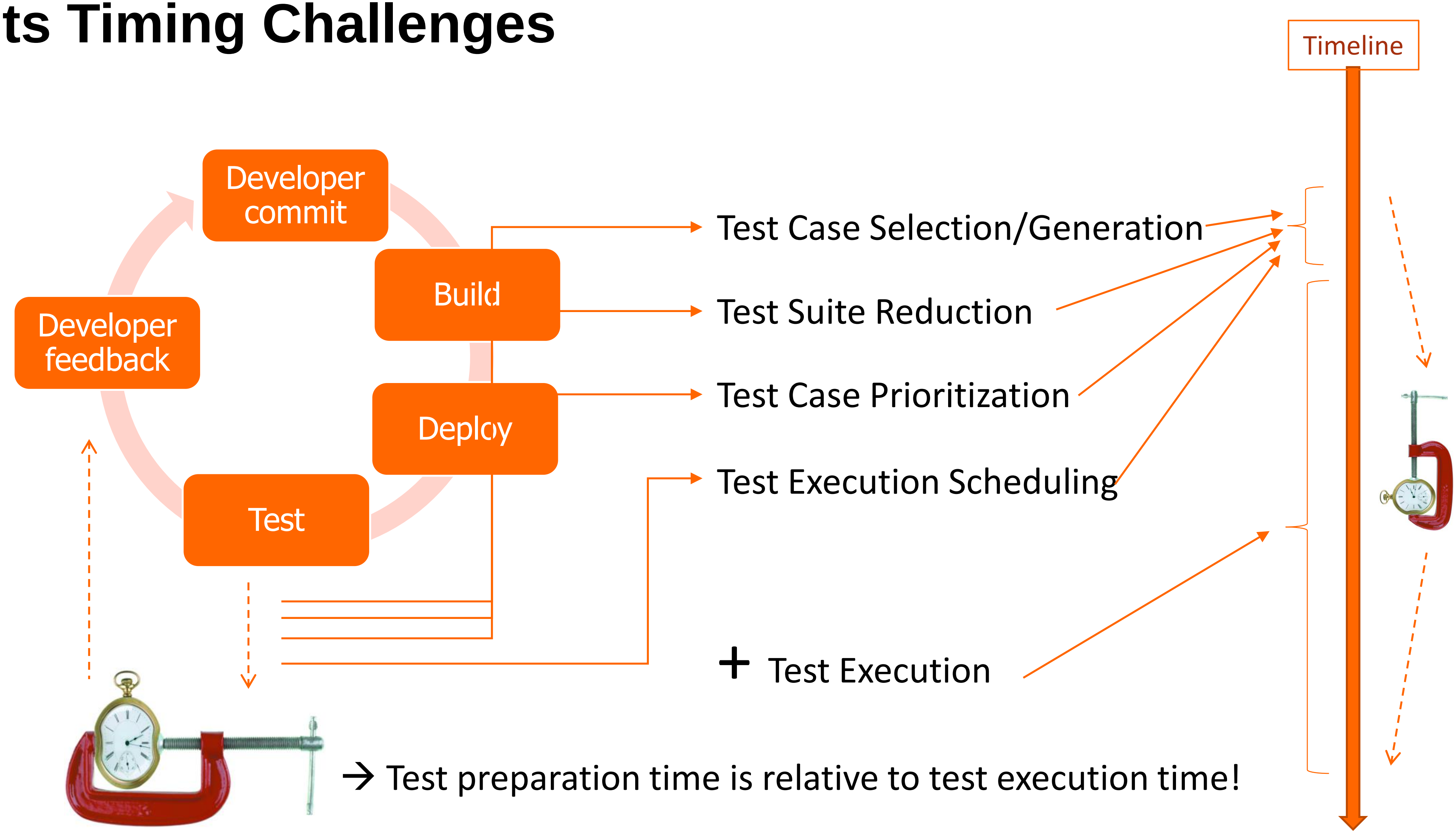
Automated Mooring System

Source: MacGregor Inc.

Testing Non-testable Autonomous Systems

- Testing perception systems needs to generate tests with (environment) hazards
- Test coverage over high-dimensional inputs is limited
- Non-linear motion planning involves solving complex constraint models
- Validation of learning systems needs test oracles which can hardly be defined
- Continuous testing is key but needs high control and more diversity

An Ideal Cycle of Continuous Integration and its Timing Challenges



Timeline

Developer commit

Build

Deploy

Test

Developer feedback

Test Case Selection/Generation

Test Suite Reduction

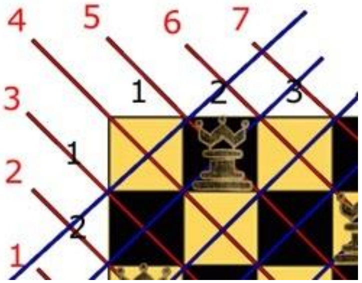
Test Case Prioritization

Test Execution Scheduling

+ Test Execution

→ Test preparation time is relative to test execution time!

Deployment of “Intelligent” Continuous Testing

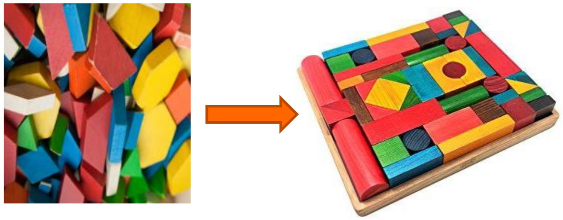


Constraint Programming

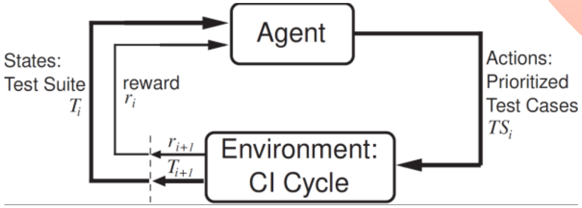
1. Test Suite Reduction

2. Test Execution Scheduling

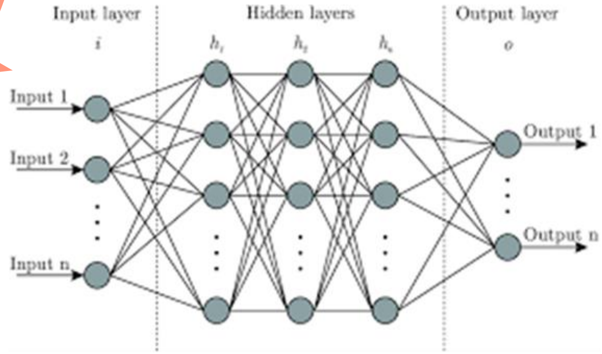
3. ML for testing autonomous Systems



Constraint-based Scheduling



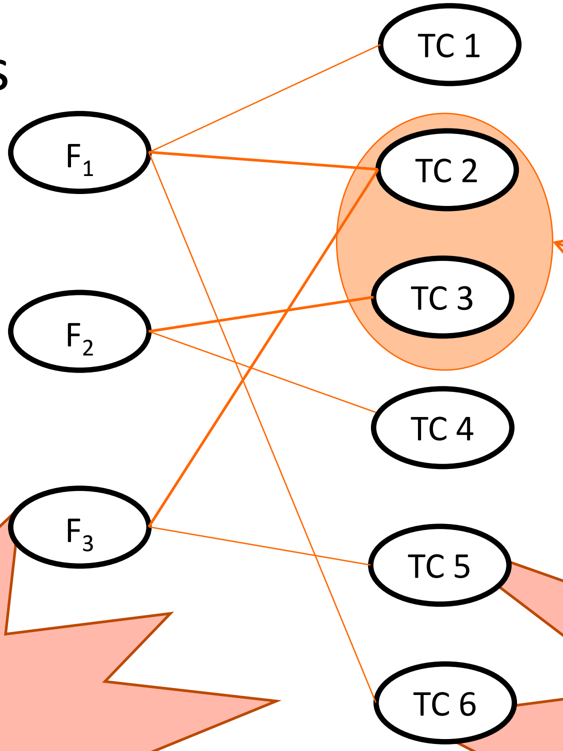
Reinforcement Learning



Artificial Neural Networks

Optimal Test Suite Reduction

F_i: Requirements
TC: Test Cases

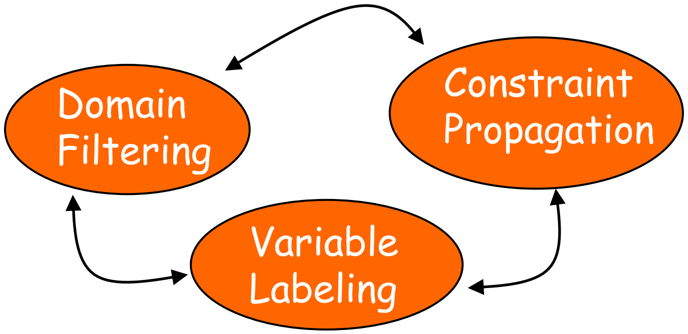


Optimally Reduced
Test Suite

NP-hard
problem!

Similar to the Vertex
Cover problem in a
bipartite graph

Constraint Programming (CP)



- Routinely used in Validation & Verification, **CP** handles efficiently hundreds of thousands of constraints and variables

- CP is versatile: user-defined constraints, dedicated solvers, programming search heuristics **but it is not a silver bullet** (developing efficient CP models and heuristics requires expertise)

→ **Global constraints**: relations over a non-fixed number of variables, implementing dedicated filtering algorithms

The **nvalue** global constraint

[Pachet Roy 1999, Beldiceanu 01]

nvalue(N, V)

Where:

N is a finite-domain variable

$V = [V_1, \ldots, V_k]$ is a vector of variables

nvalue(N, V) holds iff $N = card(\{V_i\}_{i \text{ } in \text{ } 1..k})$

nvalue(N, [3, 1, 3]) entails N = 2

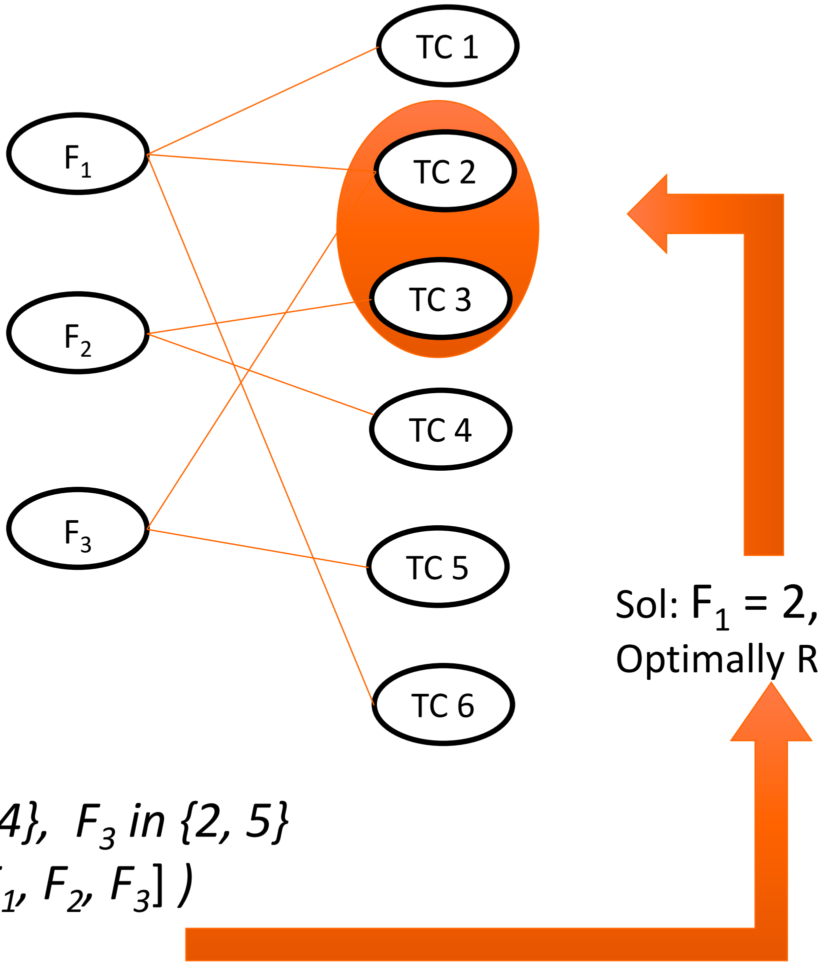
nvalue(3, $[X_1, X_2]$) fails

nvalue(1, $[X_1, X_2, X_3]$) entails $X_1 = X_2 = X_3$

N in 1..2**, nvalue**(N, $[4, 7, X_3]$) entails X_3 in {4,7}, N=2

Optimal Test Suite Reduction with nvalue

However,
only F_1, F_2, F_3
are available
for labeling!



Sol: $F_1 = 2$, $F_2 = 3$, $F_3 = 2$
Optimally Reduced Test Suite

F_1 in {1, 2, 6}, F_2 in {3, 4}, F_3 in {2, 5}
nvalue(*MaxNvalue*, *[F_1, F_2, F_3]*)
Minimize(*MaxNvalue*)

The global_cardinality constraint (gcc)

[Regin AAAI'96]

gcc(T, d, V)

Where

$T = [T_1, \ldots, T_N]$ is a vector of N variables

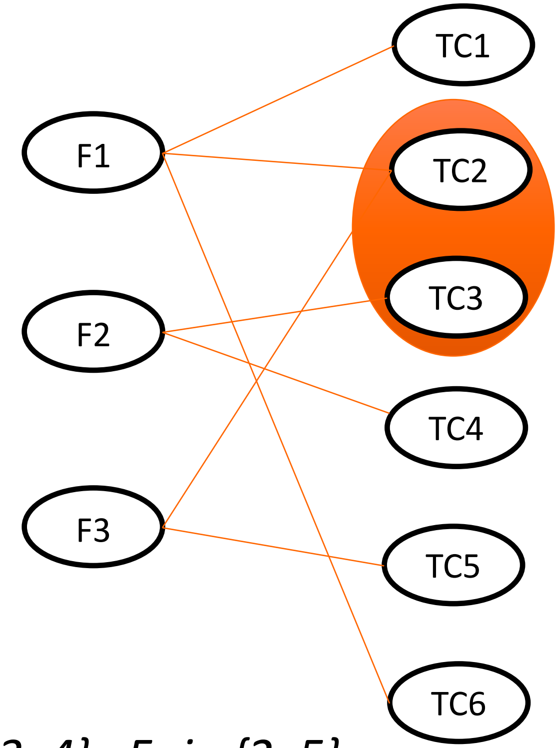
$d = [d_1, \ldots., d_k]$ is a vector of k values

$V = [V_1, \ldots, V_k]$ is a vector of k variables

gcc(T, d, V) holds iff $\forall i \;\; in \;\; 1..k,$ $V_i = card(\{j \mid T_j = di\})$

Filtering algorithms for **gcc** are based on max-flow computations

Mixt model using gcc and nvalue



F_1 in {1, 2, 6}, F_2 in {3, 4}, F_3 in {2, 5}

gcc([F_1, F_2, F_3], [1,2,3,4,5,6], [V_1, V_2, V_3, V_4, V_5, V_6])

nvalue(MaxNvalue, [F_1, F_2, F_3])

Minimize(MaxNvalue)

Model pre-processing

F_1 in {1, 2, 6} → $F_1 = 2$

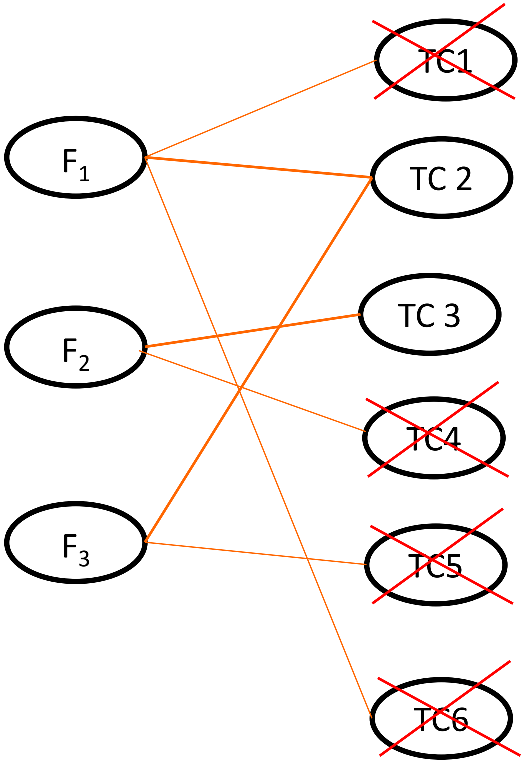
as $cov(TC_1) \subset cov(TC_2)$ and $cov(TC_6) \subset cov(TC_2)$

withdraw TC_1 and TC_6

F_3 is covered → withdraw TC_5

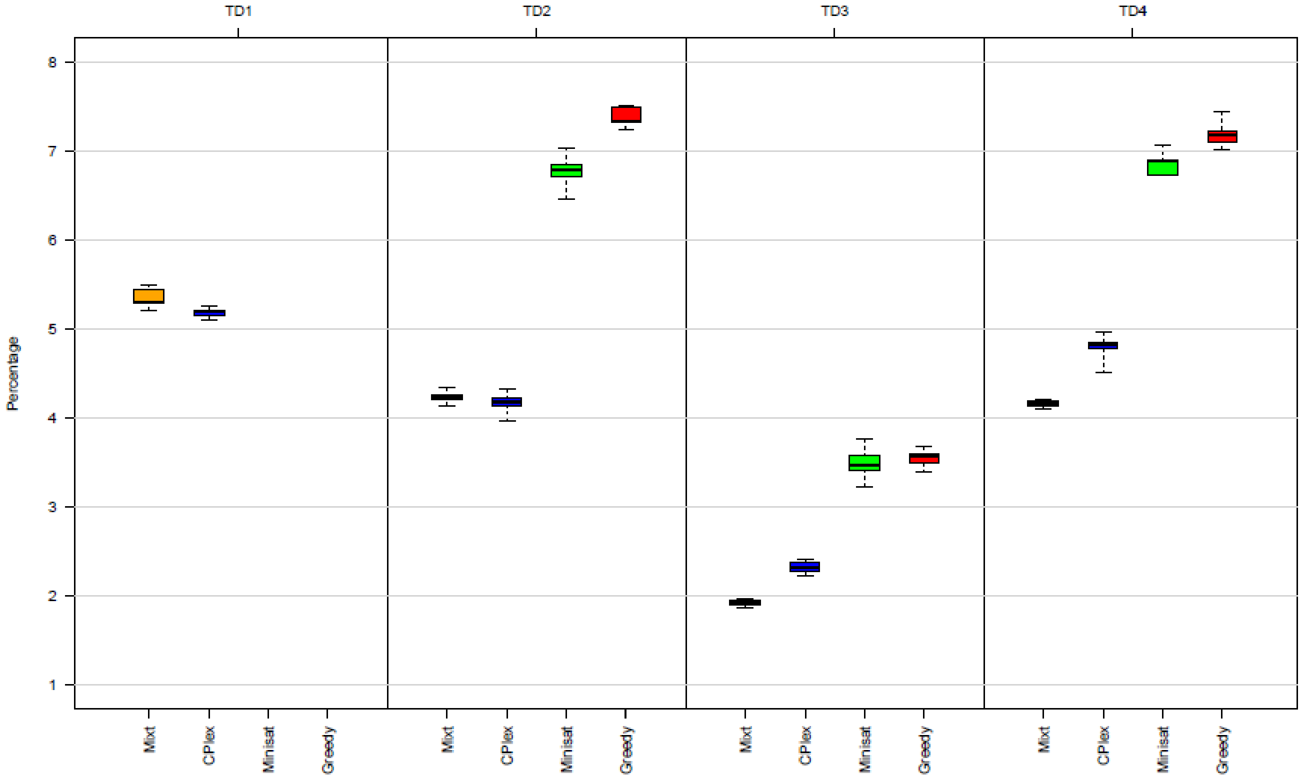
F_2 in {3,4} → e.g., $F_2 = 3$, withdraw TC_4

Pre-processing rules can be expressed once and then applied iteratively



Comparison with CPLEX, MiniSAT, Greedy (uniform costs)

(Reduced Test Suite percentage in 60 sec)



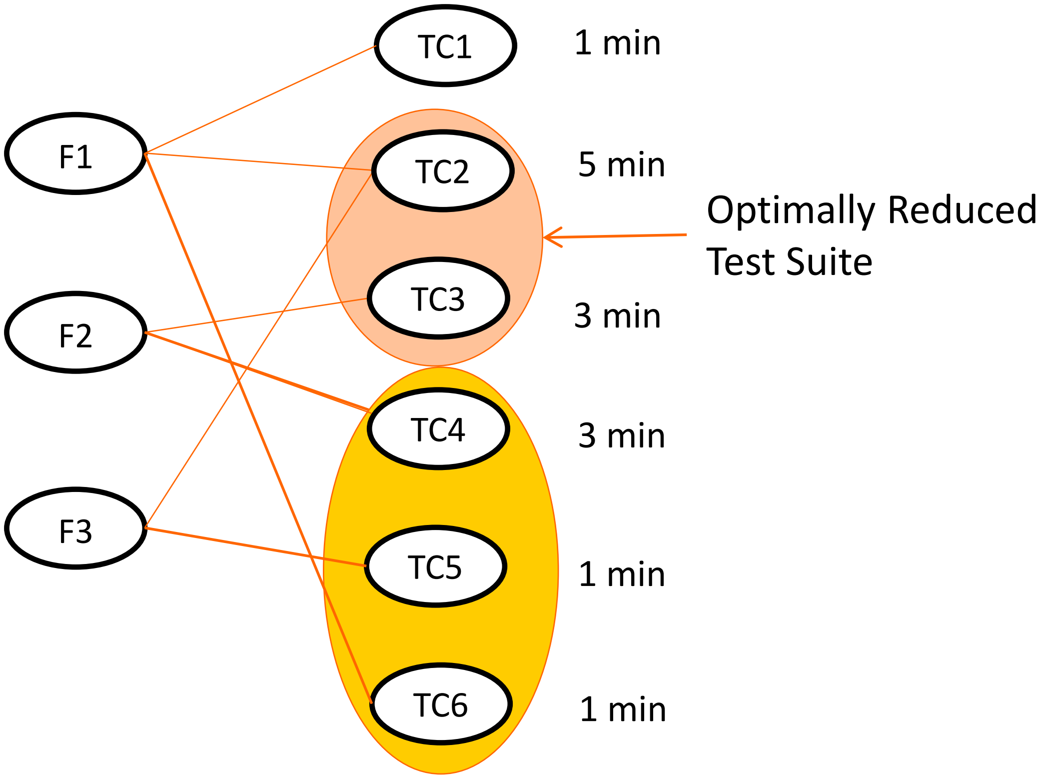
	TD1	TD2	TD3	TD4
Requirements	1000	1000	1000	2000
Test cases	2000	5000	5000	5000
Density	20	7	20	20

A. Gotlieb and D. Marijan - ***FLOWER: Optimal Test Suite Reduction As a Network Maximum Flow*** *– ACM Int. Symp. on Soft. Testing and Analysis (ISSTA'14), San José, CA, Jul. 2014.*

A. Gotlieb and D. Marijan - **Using Global Constraints to Automate Regression Testing -** *AI Magazine* 38, no. Spring (2017).

Other Criteria to Minimize

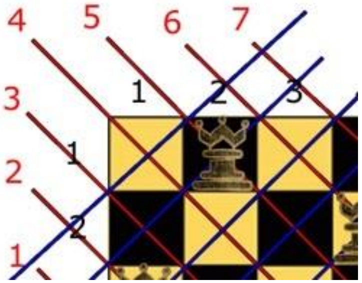
Requirement coverage
is always a prerequiste



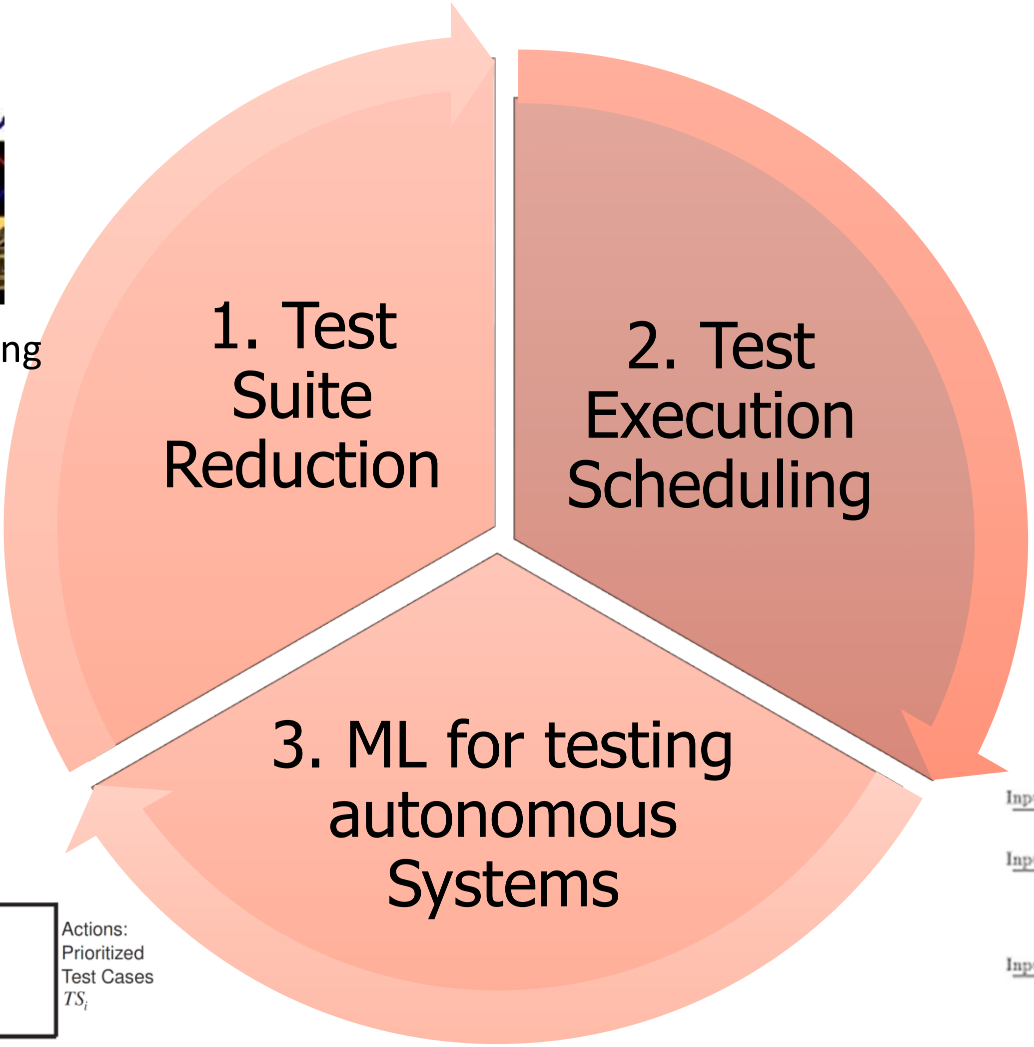
Execution time!

A Gotlieb, M Carlsson, D Marijan, A Petillon. ***A New Approach to Feature-based Test Suite Reduction in Software Product Line Testing****. ICSOFT-EA 2016. Best paper award. Scitepress.org*

Deployment of “Intelligent” Continuous Testing



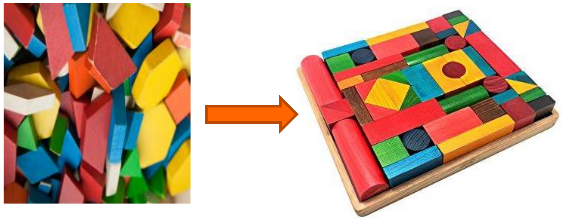
Constraint Programming



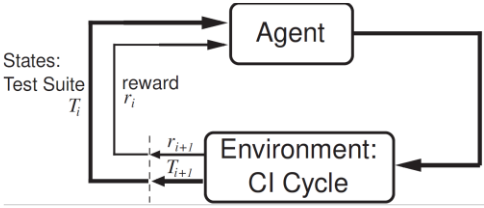
1. Test Suite Reduction

2. Test Execution Scheduling

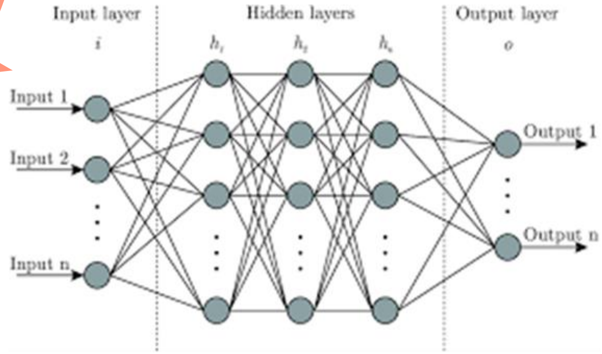
3. ML for testing autonomous Systems



Constraint-based Scheduling

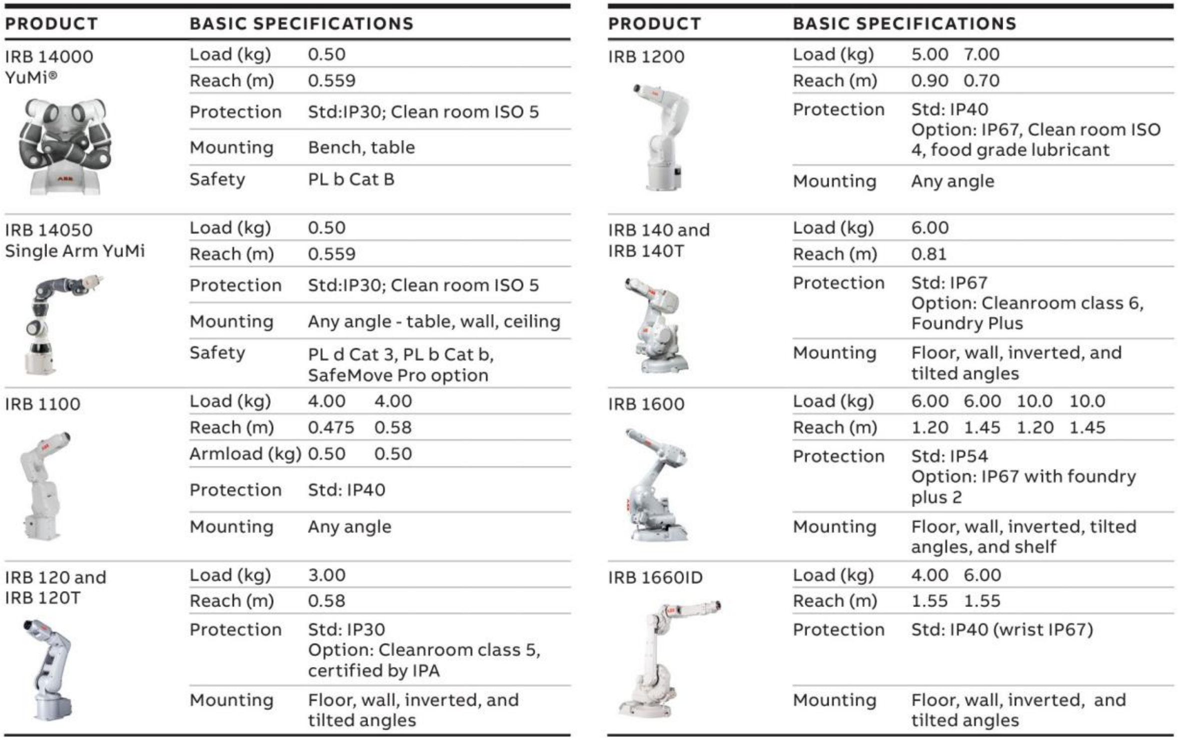


Reinforcement Learning



Artificial Neural Networks

Test Selection and Test Suite Reduction: An Example at ABB Robotics



PRODUCT	BASIC SPECIFICATIONS	
IRB 14000 YuMi®	Load (kg)	0.50
	Reach (m)	0.559
	Protection	Std:IP30; Clean room ISO 5
	Mounting	Bench, table
	Safety	PL b Cat B
IRB 14050 Single Arm YuMi	Load (kg)	0.50
	Reach (m)	0.559
	Protection	Std:IP30; Clean room ISO 5
	Mounting	Any angle - table, wall, ceiling
	Safety	PL d Cat 3, PL b Cat b, SafeMove Pro option
IRB 1100	Load (kg)	4.00 4.00
	Reach (m)	0.475 0.58
	Armload (kg)	0.50 0.50
	Protection	Std: IP40
	Mounting	Any angle
IRB 120 and IRB 120T	Load (kg)	3.00
	Reach (m)	0.58
	Protection	Std: IP30 Option: Cleanroom class 5, certified by IPA
	Mounting	Floor, wall, inverted, and tilted angles

PRODUCT	BASIC SPECIFICATIONS	
IRB 1200	Load (kg)	5.00 7.00
	Reach (m)	0.90 0.70
	Protection	Std: IP40 Option: IP67, Clean room ISO 4, food grade lubricant
	Mounting	Any angle
IRB 140 and IRB 140T	Load (kg)	6.00
	Reach (m)	0.81
	Protection	Std: IP67 Option: Cleanroom class 6, Foundry Plus
	Mounting	Floor, wall, inverted, and tilted angles
IRB 1600	Load (kg)	6.00 6.00 10.0 10.0
	Reach (m)	1.20 1.45 1.20 1.45
	Protection	Std: IP54 Option: IP67 with foundry plus 2
	Mounting	Floor, wall, inverted, tilted angles, and shelf
IRB 1660ID	Load (kg)	4.00 6.00
	Reach (m)	1.55 1.55
	Protection	Std: IP40 (wrist IP67)
	Mounting	Floor, wall, inverted, and tilted angles

From a concrete set up:

Test Case Repository:
~10,000 Test Cases (TC)
~25 distinct Test Robots
~500 distinct features

10..30 code changes per day

→ Select, schedule and execute about 150 TC per CI cycle

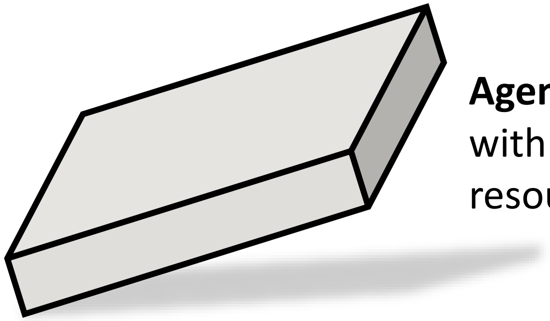
Constraint-Based Scheduling

Tasks
with distinct
characteristics

Schedule

Assignment of **Tasks to Agents** such that:

1. Task execution is not interrupted or paused
2. Agents are *maximally* occupied
3. Tasks sharing a global resource are not executed at the same time
4. Diversity of assignment of tasks to agents is ensured



Agents
with limited time or
resources capacity

Goal:
Schedule as much tasks as possible on available agents such that the overall execution time is minimized

Test Case Execution Scheduling

(T, M, G, d, g, f)

***T**: a set of Test Cases*
***M**: a set of Machines, e.g., robots*
***G**: a set of (non-shareable) resources*

d: $T \rightarrow N$ *estimated duration*
g: $T \rightarrow 2^G$ *usage of global resources*
f: $T \rightarrow 2^M$ *possible machines*

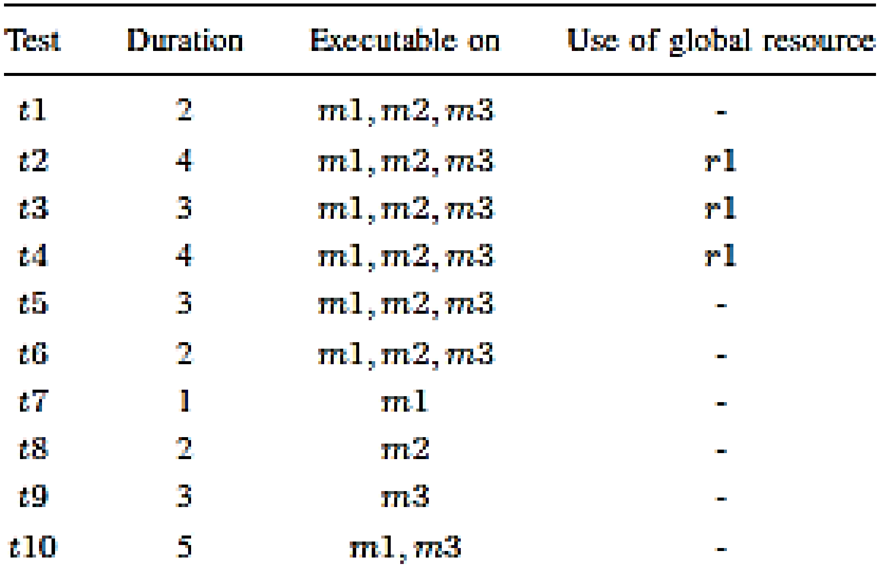
Disjunctive scheduling, non-preemptive,
non-shareable resources,
machine-independant execution time

Function to optimize:
TimeSpan: the overall duration of test execution T_E
(in order to minimize the round-trip time)

In practice, global optimality is desired but not mandatory, it’s more important to control T_S w.r.t T_E
→ Time-contract global optimization

A simple example

d *f* *g*



Test	Duration	Executable on	Use of global resource
$t1$	2	$m1, m2, m3$	-
$t2$	4	$m1, m2, m3$	$r1$
$t3$	3	$m1, m2, m3$	$r1$
$t4$	4	$m1, m2, m3$	$r1$
$t5$	3	$m1, m2, m3$	-
$t6$	2	$m1, m2, m3$	-
$t7$	1	$m1$	-
$t8$	2	$m2$	-
$t9$	3	$m3$	-
$t10$	5	$m1, m3$	-

Test Cases*: t1, t2, t3, t4, t5, t6, t7, t8, t9, t9, t10*

r1

m3

m2

m1

The **CUMULATIVE** global constraint

[Aggoun & Beldiceanu AAAI'93]

CUMULATIVE(t, d, r, m)

Where

$t = (t_1, \ldots, t_N)$ is a vector of tasks, each t_i in $S_i .. E_i$

$d = (d_1, \ldots., d_N)$ is a vector of task duration

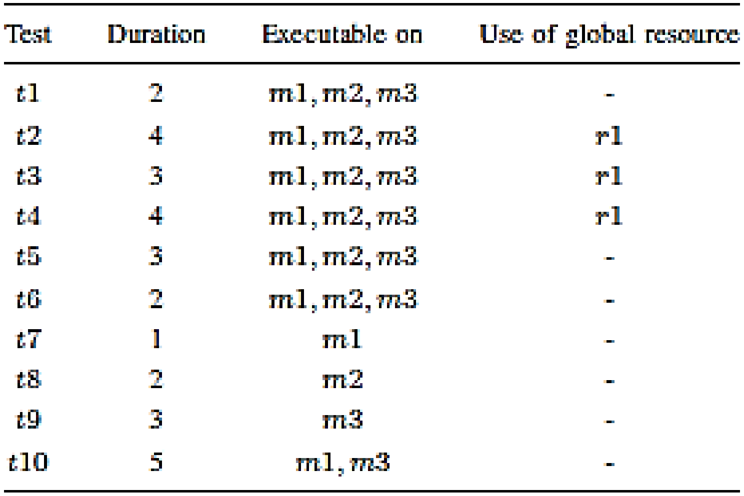
$r = (r_1, \ldots, r_N)$ is a vector of resource consumption rates

m is a scalar

CUMULATIVE ***(t, d, r, m)*** holds iff

$$\sum_{\substack{i=1 \\ t_i \leq t \leq t_i + d_i}}^{N} r_i \leq m$$

Using the global constraint **CUMULATIVE**



Test	Duration	Executable on	Use of global resource
$t1$	2	$m1, m2, m3$	-
$t2$	4	$m1, m2, m3$	$r1$
$t3$	3	$m1, m2, m3$	$r1$
$t4$	4	$m1, m2, m3$	$r1$
$t5$	3	$m1, m2, m3$	-
$t6$	2	$m1, m2, m3$	-
$t7$	1	$m1$	-
$t8$	2	$m2$	-
$t9$	3	$m3$	-
$t10$	5	$m1, m3$	-

CUMULATIVE$((t_1,..,t_{10}), (d_1,..,d_{10}), (1, ..,1), 3),$
$M_1,..,M_6$ in $1..3,$
$M_7 = 1, M_8 = 2, M_9 = 3, M_{10}$ in $\{1,3\},$
$(E_2 \leq S_3$ or $E_3 \leq S_2), (E_2 \leq S_4$ or $E_4 \leq S_2),$
$(E_3 \leq S_4$ or $E_4 \leq S_3),$
MAX$(MaxSpan, (E_1, \ldots, E_{10})),$
LABEL(MINIMIZE$(MaxSpan), (S_1,..,S_{10}), (M_1,..,M_{10}))$

<u>An optimal solution:</u>

$S_1 = 0, S_2 = 4, S_3 = 8, S_4 = 0, S_5 = 4, S_6 = 7, S_7 = 2, S_8 = 9,$
$S_{10} = 3,$
$M_1 = 1, M_2 = 1, M_3 = 1, M_4 = 2, M_5 = 2, M_6 = 2, M_7 = 1,$
$M_8 = 2, M_9 = 3, M_{10} = 3$
$MaxSpan = 11$

M Mossige, A Gotlieb, H Spieker, H Meling. ***Time-aware test case execution scheduling for cyber-physical systems****. Principles and Practice of Constraint Programming, Melbourne, 2017*

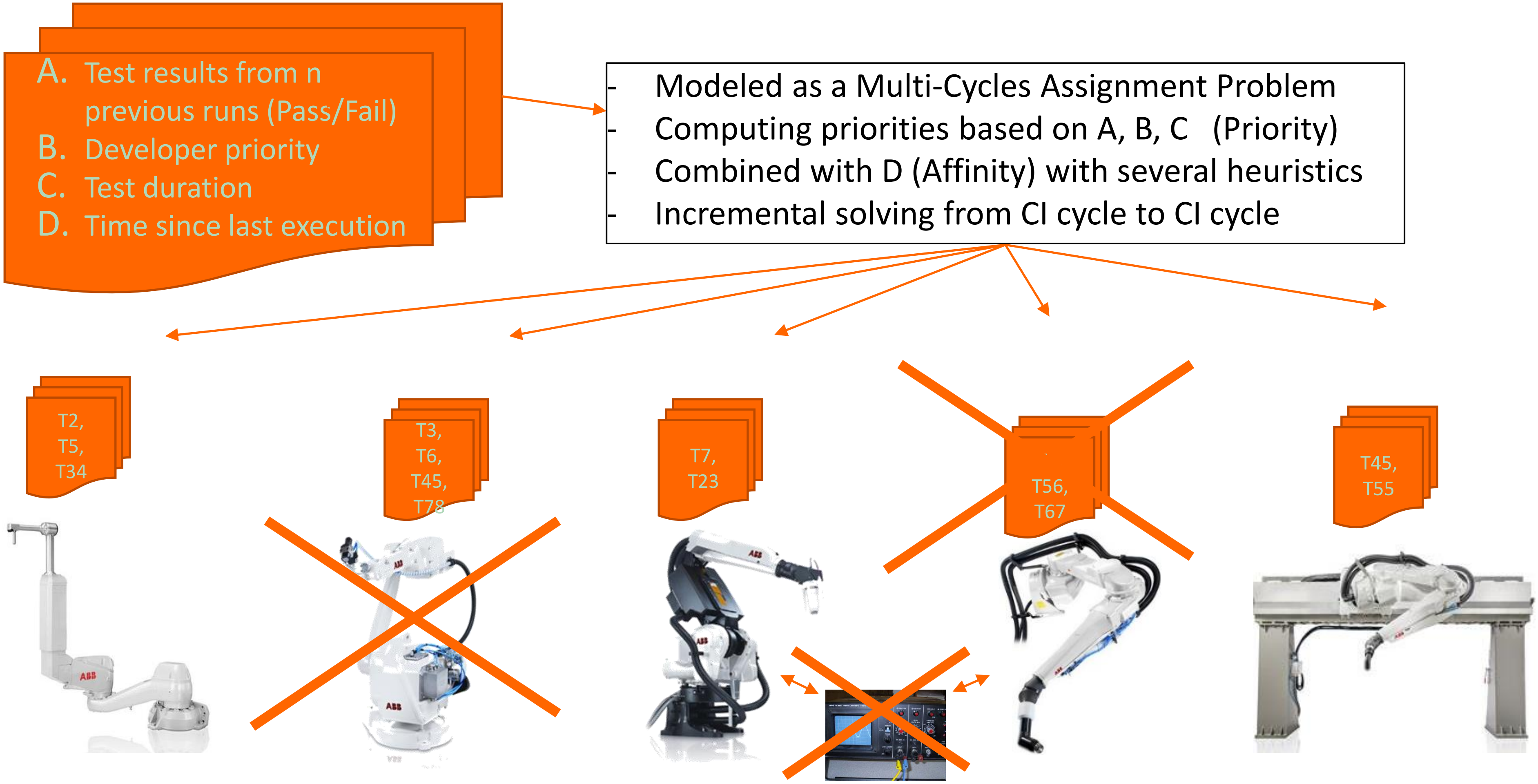
Limitations of this model

- Static model – In practice, robots and test cases are not necessarily available at each CI cycle → Need a more dynamic model!

- Historical data about test case success/failure is not taken into consideration!

- Diversity in scheduling among CI cycles is not handled

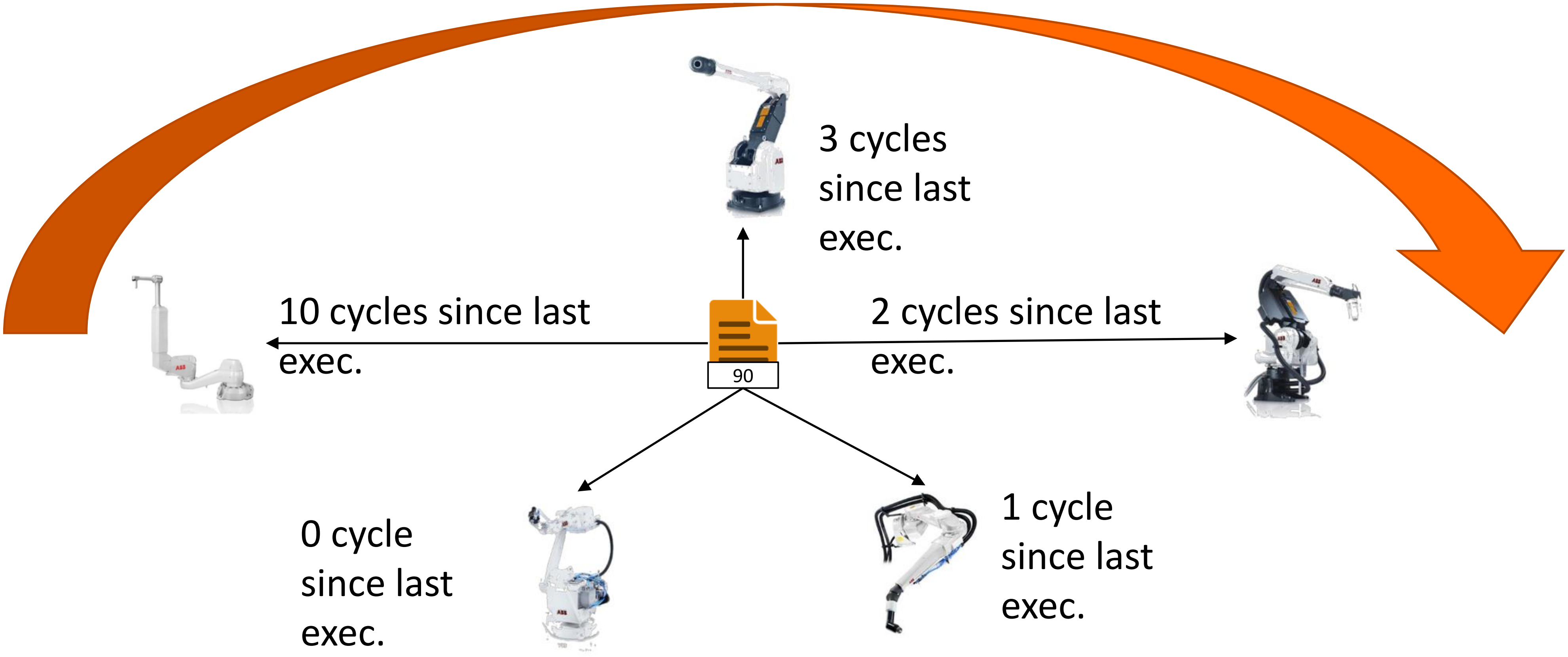
A New Approach Based on Priority and Affinity



A. Test results from n previous runs (Pass/Fail)
B. Developer priority
C. Test duration
D. Time since last execution

- Modeled as a Multi-Cycles Assignment Problem
- Computing priorities based on A, B, C (Priority)
- Combined with D (Affinity) with several heuristics
- Incremental solving from CI cycle to CI cycle

Affinity: more diversity in the test execution process



3 cycles since last exec.

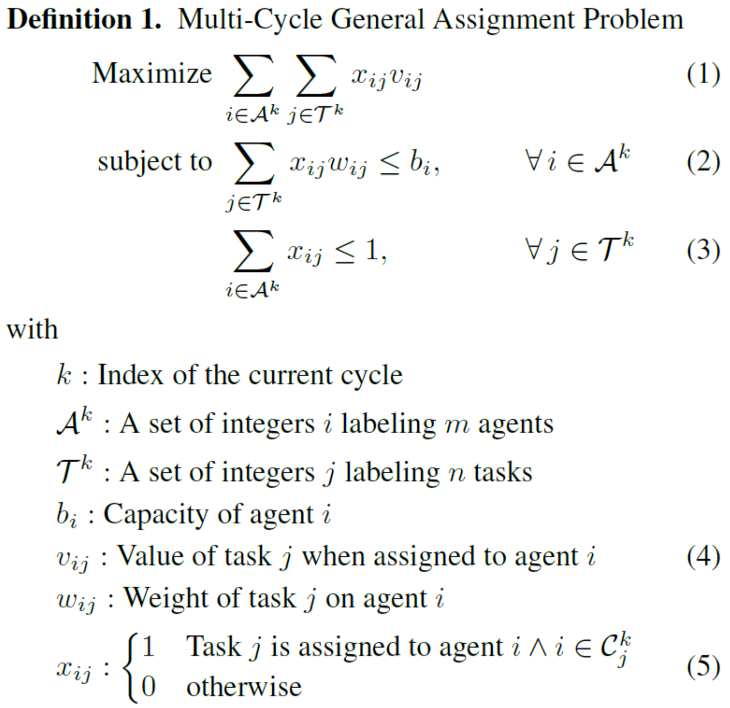
10 cycles since last exec.

2 cycles since last exec.

0 cycle since last exec.

1 cycle since last exec.

Rotational Diversity



Definition 1. Multi-Cycle General Assignment Problem

$$\text{Maximize} \sum_{i \in \mathcal{A}^k} \sum_{j \in \mathcal{T}^k} x_{ij} v_{ij} \tag{1}$$

$$\text{subject to} \sum_{j \in \mathcal{T}^k} x_{ij} w_{ij} \leq b_i, \qquad \forall i \in \mathcal{A}^k \tag{2}$$

$$\sum_{i \in \mathcal{A}^k} x_{ij} \leq 1, \qquad \forall j \in \mathcal{T}^k \tag{3}$$

with

k : Index of the current cycle

$\mathcal{A}^k$: A set of integers i labeling m agents

$\mathcal{T}^k$: A set of integers j labeling n tasks

b_i : Capacity of agent i

v_{ij} : Value of task j when assigned to agent i (4)

w_{ij} : Weight of task j on agent i

$$x_{ij} : \begin{cases} 1 & \text{Task } j \text{ is assigned to agent } i \wedge i \in \mathcal{C}_j^k \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Priority only (FOP) $v_{ij} \triangleq p_{ij}$

Affinity only (FOA) $v_{ij} \triangleq a_{ij}$

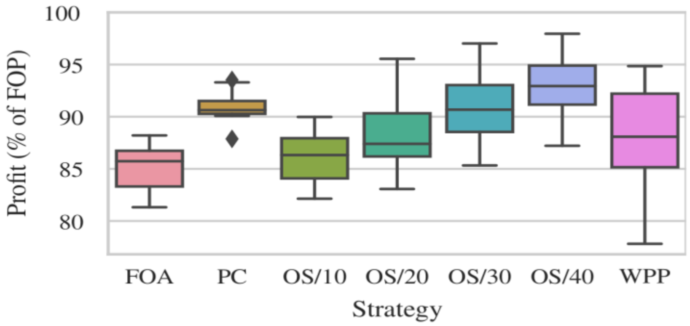
Objective Switch (OS)

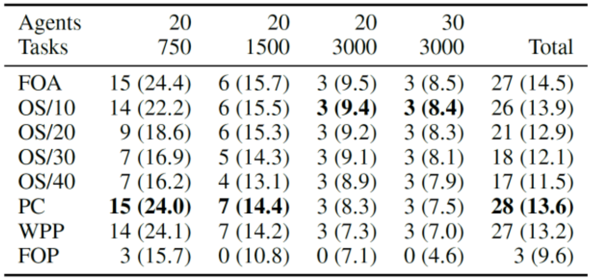
$$v_{ij} \triangleq \begin{cases} p_{ij} & \text{if } \gamma > \max_{j \in \mathcal{T}^k} \mathrm{AP}_j^k \\ a_{ij} & \text{otherwise} \end{cases}$$

Product Combination (PC) $v_{ij} \triangleq p_{ij}^{\alpha} \cdot a_{ij}^{\beta}$

Weighted Partial Profits (WPP)

$$v_{ij} \triangleq \lambda_j^k \cdot \frac{p_{ij}}{\max\limits_{i \in \mathcal{A}^k} \max\limits_{j \in \mathcal{T}^k} p_{ij}} + (1 - \lambda_j^k) \cdot \frac{a_{ij}}{\max\limits_{i \in \mathcal{A}^k} \max\limits_{j \in \mathcal{T}^k} a_{ij}}$$



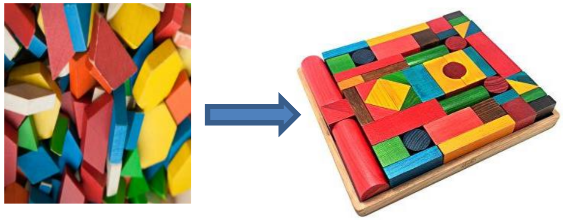


Agents Tasks	20 750	20 1500	20 3000	30 3000	Total
FOA	15 (24.4)	6 (15.7)	3 (9.5)	3 (8.5)	27 (14.5)
OS/10	14 (22.2)	6 (15.5)	**3 (9.4)**	**3 (8.4)**	26 (13.9)
OS/20	9 (18.6)	6 (15.3)	3 (9.2)	3 (8.3)	21 (12.9)
OS/30	7 (16.9)	5 (14.3)	3 (9.1)	3 (8.1)	18 (12.1)
OS/40	7 (16.2)	4 (13.1)	3 (8.9)	3 (7.9)	17 (11.5)
PC	**15 (24.0)**	**7 (14.4)**	3 (8.3)	3 (7.5)	**28 (13.6)**
WPP	14 (24.1)	7 (14.2)	3 (7.3)	3 (7.0)	27 (13.2)
FOP	3 (15.7)	0 (10.8)	0 (7.1)	0 (4.6)	3 (9.6)

(b) Diversity: Full rotations of all tasks (Avg. rotations per task)

SWMOD: Deployment of Time-aware Test Case Execution Scheduling at ABB Robotics

- ~1500 lines of SICStus Prolog Code with CP(FD)
- Fully integrated into the MS-TFS Continuous Integration
- Using the global constraint binpacking + rotational diversity
- Deployed at ABB since Feb. 2019



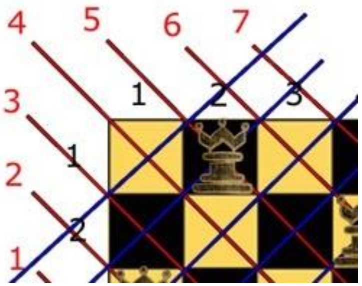
Constraint-based Scheduling

CP with **global constraints (cumulative, binpacking)** and rotational diversity can solve the test execution scheduling problem

"SWMOD deployed at ABB Robotics and used every day to schedule tests throughout several ABB centers in the world (Norway, Sweden, India, China)"

*H. Spieker, A. Gotlieb and M. Mossige - **Rotational Diversity in Multi-Cycle Assignment Problems** - In Proc. of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19). Feb. 2019.*

Deployment of “Intelligent” Continuous Testing

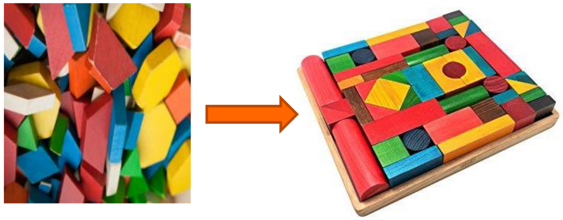


Constraint Programming

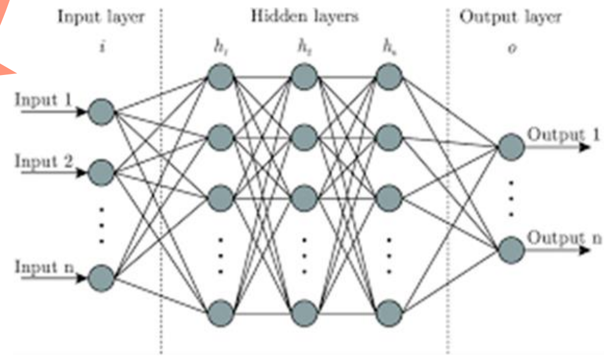
1. Test Suite Reduction

2. Test Execution Scheduling

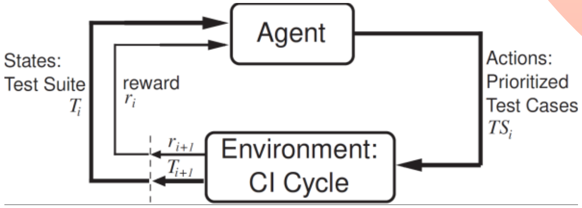
3. ML for testing autonomous Systems



Constraint-based Scheduling



Artificial Neural Networks



Reinforcement Learning

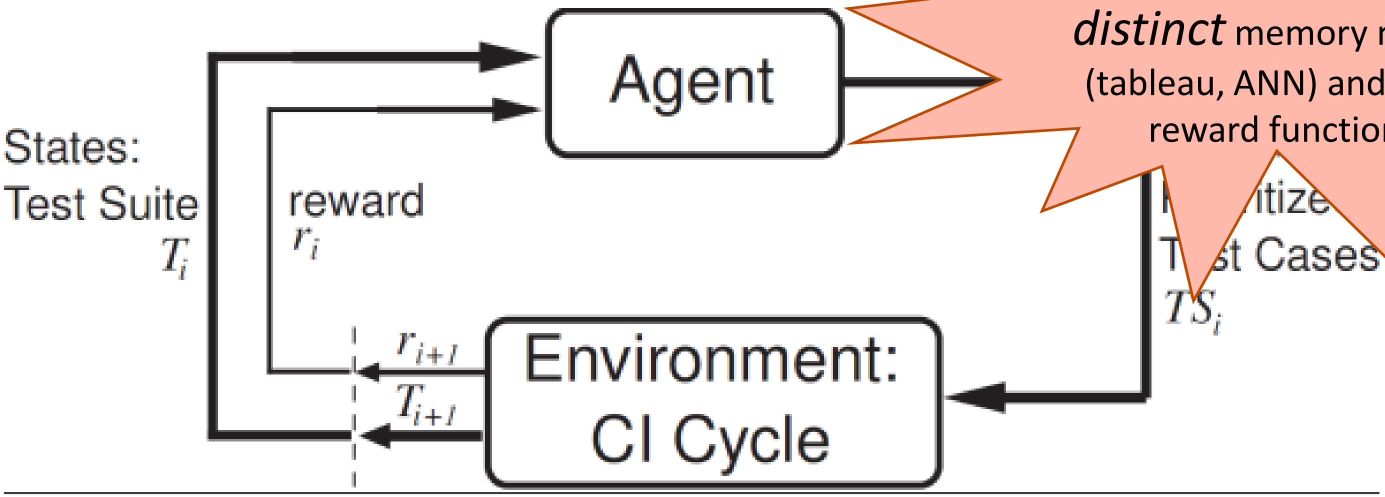
Test Prioritization: Learning from previous test runs

<u>Motivation:</u>

Adapting priorities to the most interesting test cases based on past test verdicts (from previous CI cycles)

- Considering test case meta-data only (test verdicts, execution time, ...)
- Limited memory of past executions / test verdicts

- *Using **Reinforcement Learning** for priorising test cases*



Implemented with *two distinct* memory models (tableau, ANN) and three reward functions

Reward Functions and Experimental Evaluation

3 Industrial data sets (1 year of CI cycles)
NAPFD: Normalized Average Percentage of Faults Detected

Reward Function 1. Failure Count Reward

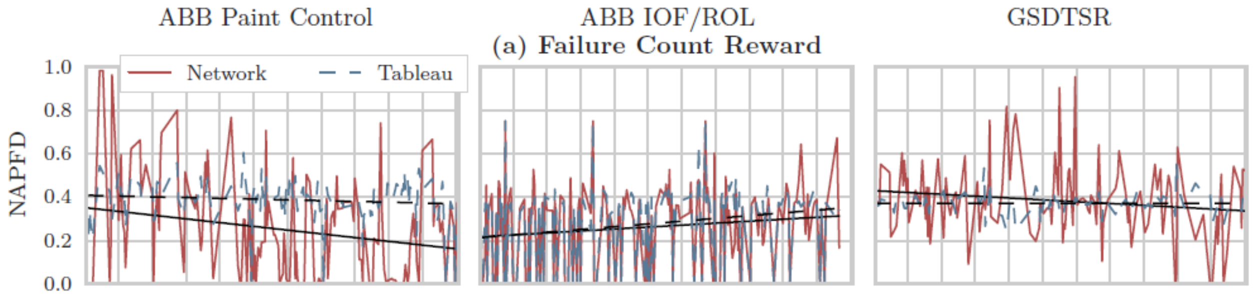
$$reward_i^{fail}(t) = |\mathcal{TS}_i^{fail}| \qquad (\forall\, t \in \mathcal{T}_i)$$

Reward Function 2. Test Case Failure Reward

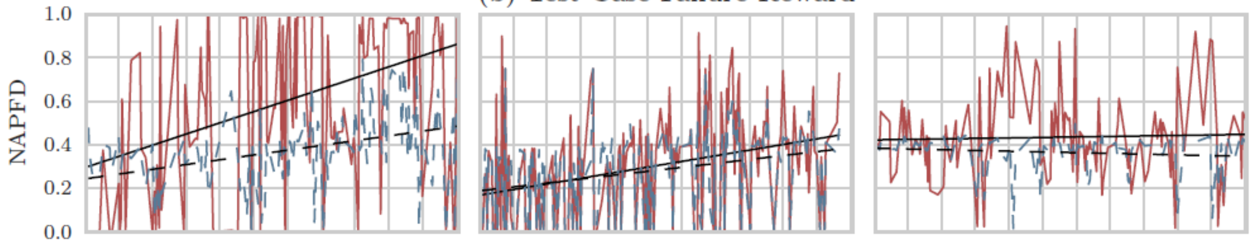
$$reward_i^{tcfail}(t) = \begin{cases} 1 - t.verdict_i & \text{if } t \in \mathcal{TS}_i \\ 0 & \text{otherwise} \end{cases}$$

Reward Function 3. Time-ranked Reward

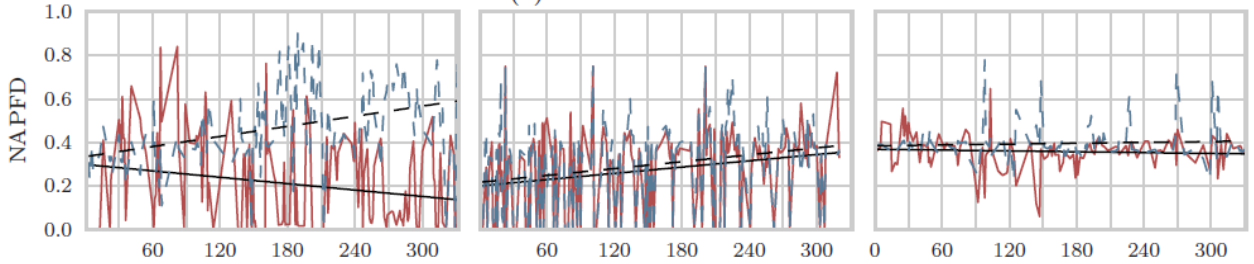
$$reward_i^{time}(t) = |\mathcal{TS}_i^{fail}| - t.verdict_i \times \sum_{\substack{t_k \in \mathcal{TS}_i^{fail} \wedge \\ rank(t) < rank(t_k)}} 1$$



(a) Failure Count Reward



(b) Test Case Failure Reward

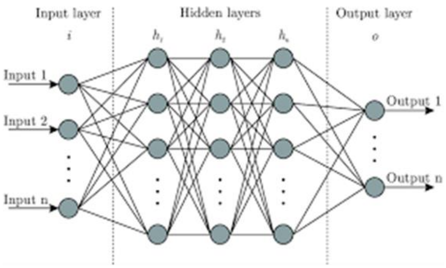


(c) Time-ranked Reward

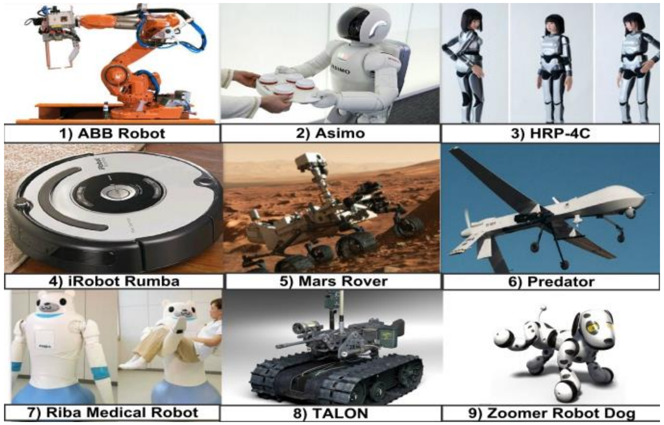
H. Spieker, A. Gotlieb, D. Marijan and M. Mossige **Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration.** In Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'17). New York, NY, USA: ACM, 2017.

Adaptive Metamorphic Testing

Motivation: Learning which *Metamorphic Relation* works best to test vision-based systems



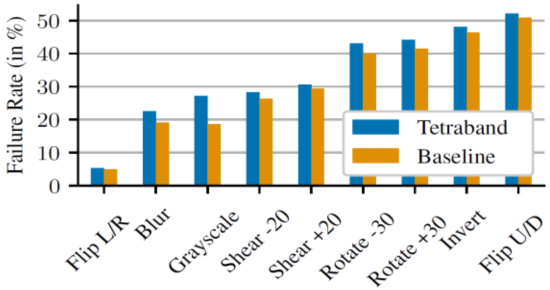
Artificial Neural Networks



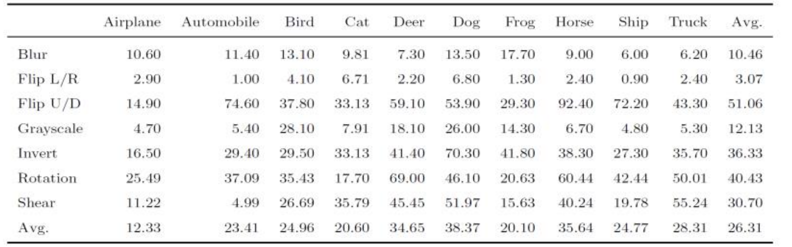
TensorFlow.org - Image classification – dataset of 10,000 images

Object Detection case study – MS COCO dataset of 5,000 images

Using Contextual Bandits (Reinforcement Learning) to learn how to select metamorphic relations
→ Adaptive Metamorphic Testing



(a) Image Classification



	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	Avg.
Blur	10.60	11.40	13.10	9.81	7.30	13.50	17.70	9.00	6.00	6.20	10.46
Flip L/R	2.90	1.00	4.10	6.71	2.20	6.80	1.30	2.40	0.90	2.40	3.07
Flip U/D	14.90	74.60	37.80	33.13	59.10	53.90	29.30	92.40	72.20	43.30	51.06
Grayscale	4.70	5.40	28.10	7.91	18.10	26.00	14.30	6.70	4.80	5.30	12.13
Invert	16.50	29.40	29.50	33.13	41.40	70.30	41.80	38.30	27.30	35.70	36.33
Rotation	25.49	37.09	35.43	17.70	69.00	46.10	20.63	60.44	42.44	50.01	40.43
Shear	11.22	4.99	26.69	35.79	45.45	51.97	15.63	40.24	19.78	55.24	30.70
Avg.	12.33	23.41	24.96	20.60	34.65	38.37	20.10	35.64	24.77	28.31	26.31

Table 1: CIFAR-10 dataset: Effects of MRs by the true class of the image. Each cell value shows the percentage of images in the class, which are wrongly classified after applying the MR. Every class contains 1000 images. Rotation and Shear are parameterized by 30 degrees.

---

*H. Spieker, A. Gotlieb – **Adaptive Metamorphic Testing with Contextual Bandits** – Journal of Systems and Software. 165: 110 (2020)*
A. Gotlieb, D. Marijan and H. Spieker**: Testing Industrial Robotic Systems: A New Battlefield!**
In Software Engineering for Robotics, 465p. Edited by A. Cavalcanti, B. Dongol, R. Hierons, J. Timmis, J. Woodcock ed. Springer Nature, 2021.

Take Away Message

- *Testing autonomous systems* brings new interesting challenges for software V&V research

- **Some AI techniques such as Constraint Programming (CP) and global constraints** are already very successful in test case generation, test suite reduction and test execution scheduling

- Testing autonomous systems such as collaborative robots or self-driving cars is challenging *as:*

  - **Expected behaviours** cannot be specified in advance
  - **Interactions with humans** involve more safety issues

Course Overview

- Software Testing Introduction
- Code-based Testing
- Testing of Autonomous Systems
- Open Challenges in Software Testing

Testing Neuro-Symbolic AI models

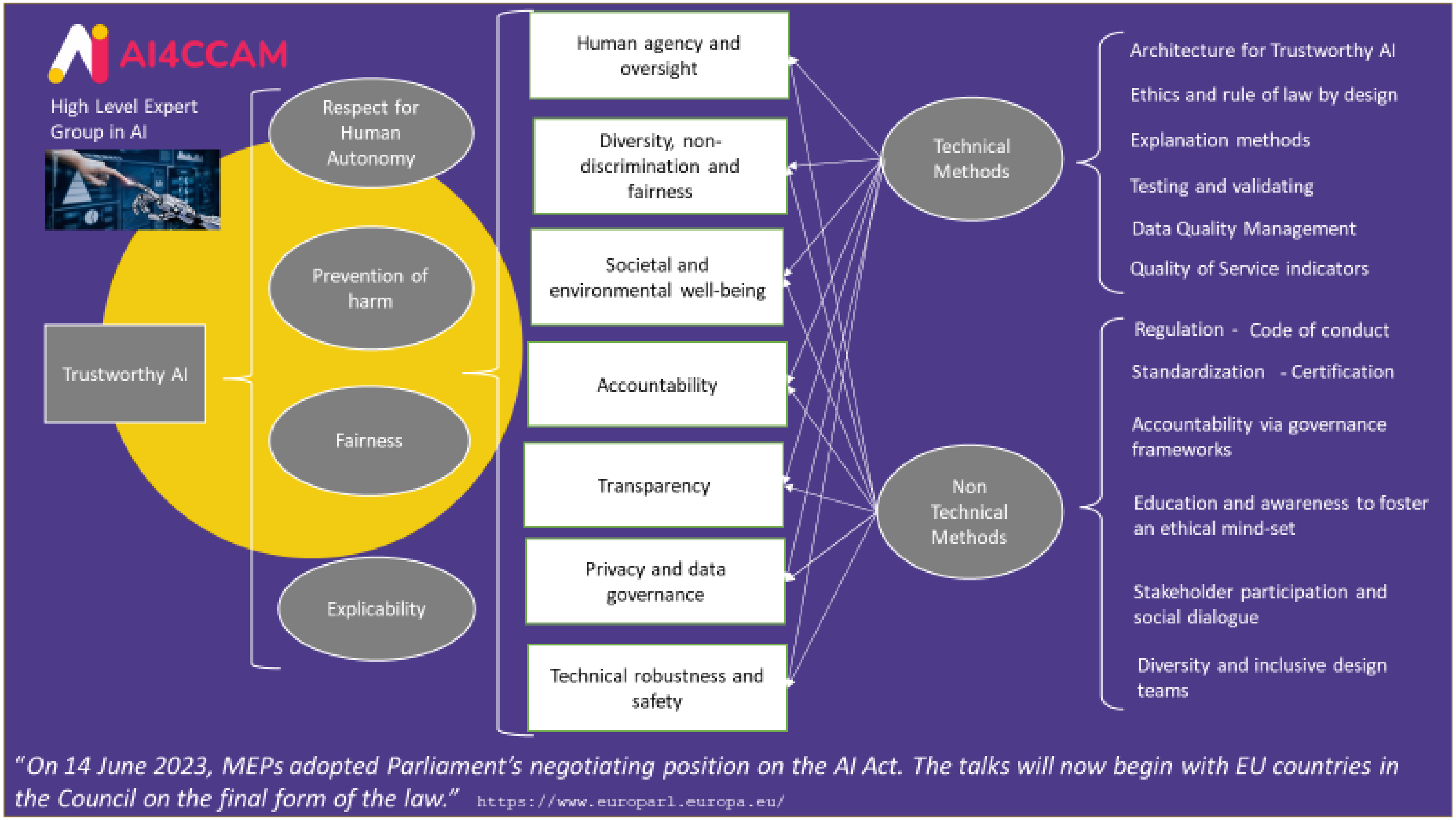
- Neuro-symbolic AI models combine NN (CNN, RNN, LSTM Transformers, etc.) with symbolic reasoning to improve
  1. The perf. of classification/regression models in ML
  2. The explicability of NN models

- Besides the oracle problem, testing these models is challenging as it it requires to quantify the benefice of each part (NN vs Symbolic)

- Testing the quality/interest of explanations is an open research question – An overall field has been created, the field of XAI

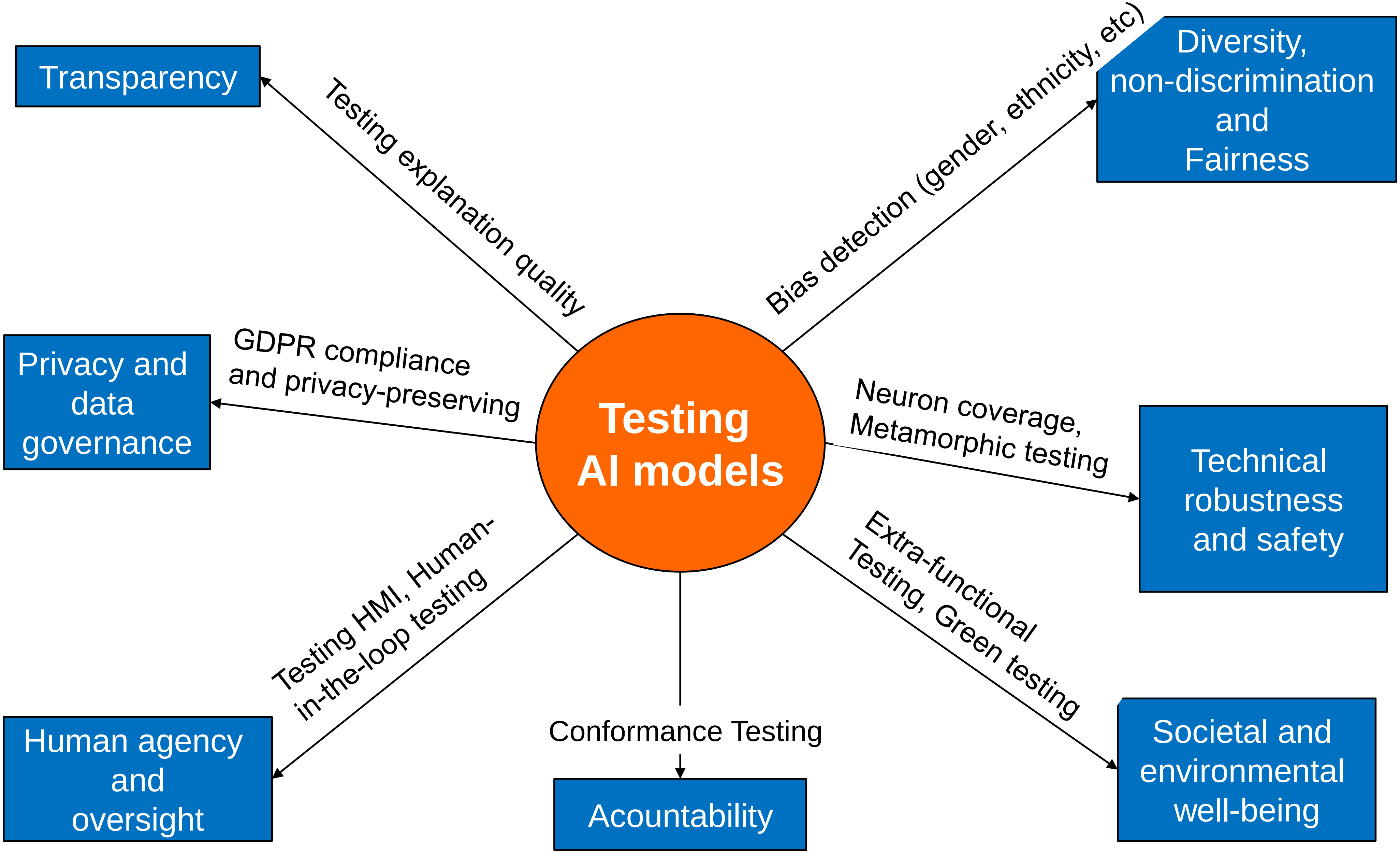
Testing AI model Trustworthiness (1)

- Need to adopt a definition of Trustworthy AI (e.g., EU HLEG AI)



"On 14 June 2023, MEPs adopted Parliament's negotiating position on the AI Act. The talks will now begin with EU countries in the Council on the final form of the law." https://www.europarl.europa.eu/

Testing AI model Trustworthiness: A Research Programme



Testing AI models

Transparency

Testing explanation quality

Diversity, non-discrimination and Fairness

Bias detection (gender, ethnicity, etc)

Privacy and data governance

GDPR compliance and privacy-preserving

Technical robustness and safety

Neuron coverage, Metamorphic testing

Human agency and oversight

Testing HMI, Human-in-the-loop testing

Acountability

Conformance Testing

Societal and environmental well-being

Extra-functional Testing, Green testing

simula

Thank You for Your Attention

VIAS Dept.

Validation Intelligence for Autonomous Software-Systems

Arnaud GOTLIEB

VIAS explores how to test the robustness, reliability, and transparency of software-systems (industrial robots, self-driving cars, navigation systems, etc.) with intelligent methods

1. **Trustworthy Artificial Intelligence for Autonomous Systems**
2. **Testing Intelligent Transport Systems**
3. **Learning and Reasoning for Data-Intensive Systems**

April 2023 (11 employees): 3 permanent researchers, 5 postdocs, 3 PhDs, 3 external PhDs + 2 ongoing recruitments

Funded by EC: **AI4CCAM** (HEU, Coordination, 2023-25), **TRANSACT** (ECSEL, 21-24), **MARS** (HEU, 23-26), **CERTIFAI** (HEU, 23-26)

Funded by RCN: **T-Largo** (2019-22), **T3AS** (19-22), **SMARTMED** (19-22), **TSAR** (19-23), **AutoCSP** (21-24)

RESIST_EA: 1st Inria-Simula Associate Team on Resilience of Software Systems (2021-2024)

simula Inria The Research Council of Norway HORIZON 2020 AI4EU