Survey on test data generation tools

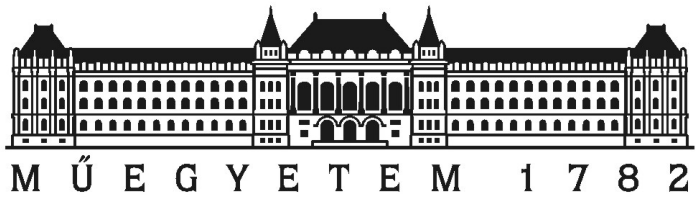
Presenter: Zoltan Karaffy

Based on:

An evaluation of white- and gray-box testing tools for C#, C++, Eiffel, and Java

by Stefan J. Galler · Bernhard K. Aichernig

Int J Softw Tools Technol Transfer (2014) 16:727–751



Content of presentation

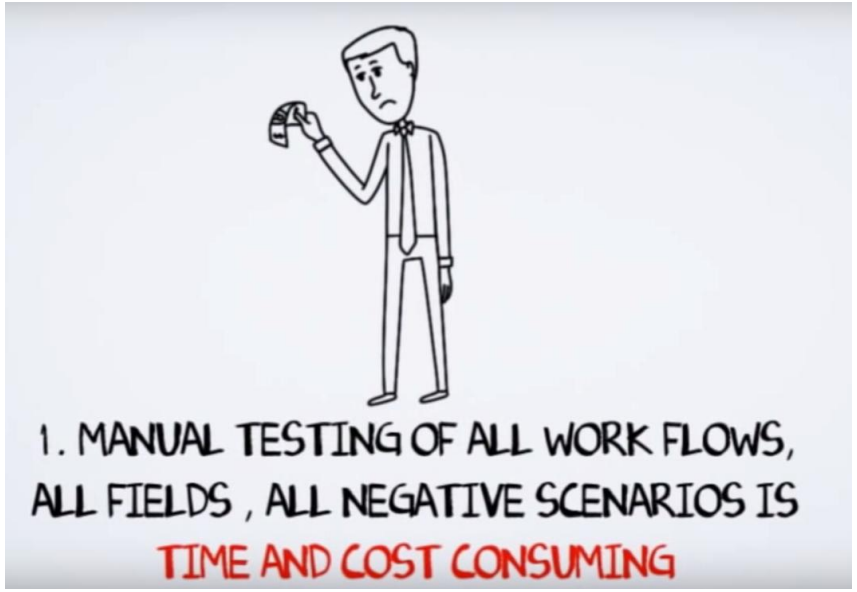
I. Introduction

II. Candidate tools

III. Evaluation criteria

IV. Conclusions

I. Introduction – Why automatize?

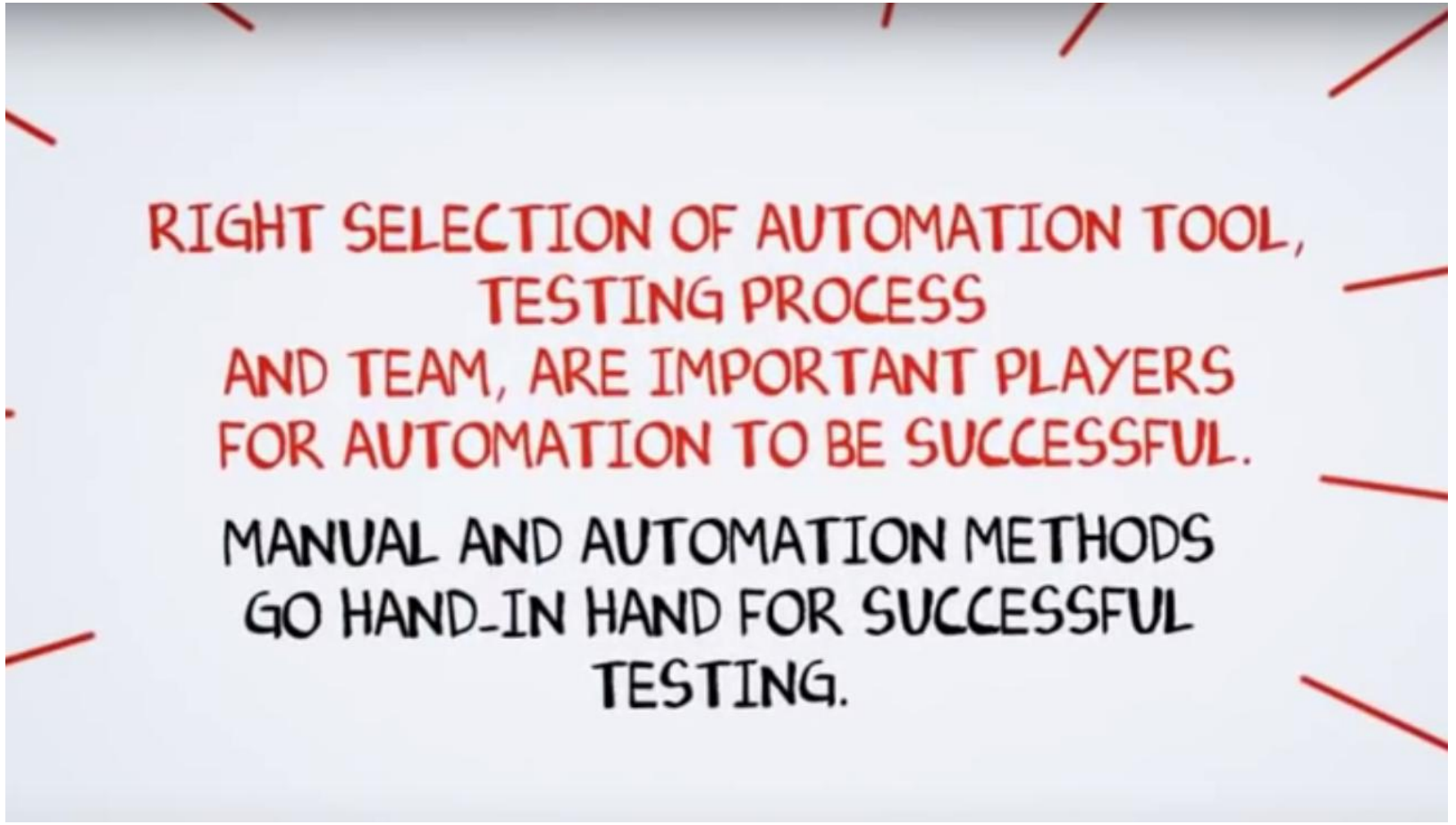


2. IT IS DIFFICULT TO TEST FOR MULTI LINGUAL SITES MANUALLY

3. AUTOMATION DOES NOT REQUIRE HUMAN INTERVENTION. YOU CAN RUN AUTOMATED TEST UNATTENDED (OVERNIGHT)

5. MANUAL TESTING CAN BECOME BORING AND HENCE ERROR PRONE.

I. Introduction



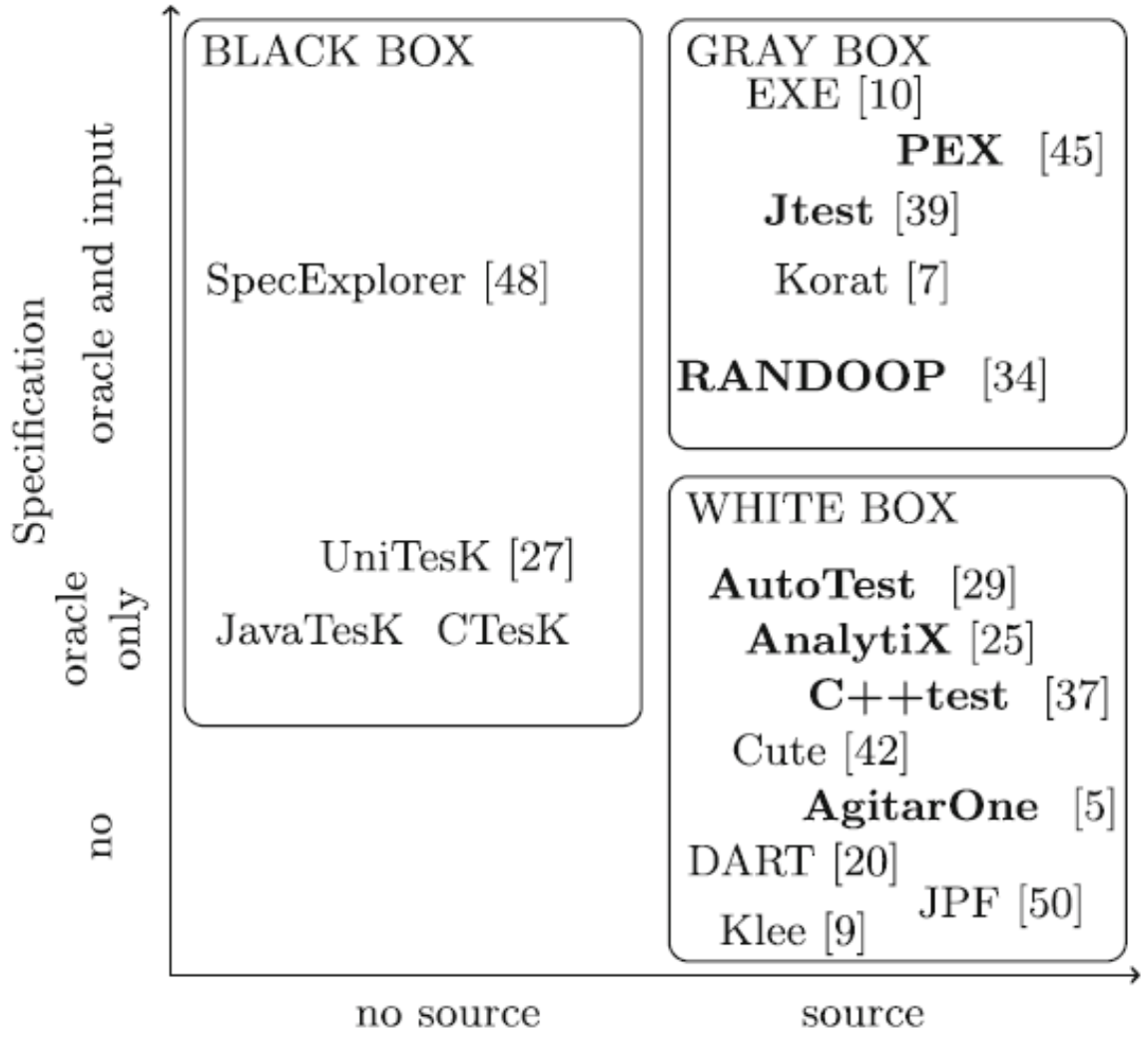
I. Introduction

Automating the test process still consists of multiple facets, ordered with respect to increasing complexity:
(a) Executing tests,
(b) generating empty test classes and methods,
(c) generating test cases, and
(d) generating test data

Focus:

- Current research efforts focus on **automatically generating test cases and test data**. The former automates the process of finding out which method sequence may reveal an error. The latter automates the generation of primitive values and especially non-primitive objects that can be used in test cases.
- The survey focuses on **white- and gray-box** techniques.

II. Candidate tools



II. Candidate tools

- *Availability*

Tools have to be publicly available. Either as free Download or as commercial tool.

- *Maturity*

Only tools that are already applied to industrial size applications are considered. We therefore rate all tools from 1 to 4:

1. commercial tool
2. applied to (at least one) industrial size case study
3. applied to (at least one) case study
4. no information about case studies available

- *Activity*

Tools have to be maintained. In otherwords, only tools updated within the last 3 years (i.e., since 2009) are considered.

- *Citation*

The amount of (scientific) publications that include references to the tool. Figures are extracted from Google scholar in December 2011. The delta value in brackets shows the amount of additional citations since October 2010.

Tool	Avail.	Mat.	Act.	Citations
AgitarOne [5]	✓	1	2010	64 (+14)
AnalytiX [25]	✓	1	2010	0 (+0)
AutoTest [29]	✓	1	2010	39 (+13)
Check'n'Crash [15]	✓	4	2005	121 (+22)
C++test [37]	✓	1	2011	0 (+0)
Cute [43]	✓	3	2006	519 (+124)
DART [20]	✗	3	2005	765 (+196)
Eclat [33]	✓	3	2005	144 (+26)
EXE [10]	✗	3	2008	389 (+124)
Klee [9]	✓	3	2009	271 (+94)
Jcrasher [14]	✓	n/a	2007	210 (+31)
JPF [50][a]	✓	3	2010	271 (+73)
Jtest [39]	✓	1	2011	0 (+0)
Korat [7]	✓	3	2007	419 (+37)
PEX [45]	✓	1	2010	199 (+88)
RANDOOP [36]	✓	1	2010	199 (+63)

AgitarOne, CodePro AnalytiX, AutoTest, C++test, Jtest, RANDOOP, and PEX

III. Evaluation

- *General description (input, ouptut, supported programming and specifcation languages, lincensing information)*
- *Data generation techniques incorporated in the tool*
- *Benchmark tests*

Type support tests

Type	Specification (arithmetic expression)			
	Constant	Simple linear arithmetic	Simple non-linear arithmetic	Inequality
Boolean	$a = true$	$a = b$	–	a != $false$
Character	$a =' b'$	–	–	$a >' b'$
Integer	$a = 3$	$a = 3 + 5$	$a = 3 * 5$	$a > 5$
Float	$a = 3.2f$	$a = 3.2f + 5.1f$	$a = 3.2f * 5.1f$	$a > 5.1f$
Double	$a = 3.2d$	$a = 3.2d + 5.1d$	$a = 3.2d * 5.1d$	$a > 3.2d$
String	$a = "abcd"$	a != $null$ && $a.matches("[a-z][0-9]+")$	–	a != $null$&& a != $""$ && a != $"abcd"$

Each of the given specifications is used as precondition for a method
The evaluation result tables for each approach show for what type of specification the approach was able to generate satisfying data

- *Dependencies between parameters* One parameter depends in any attribute from the value of another parameter.
- *Requested null objects* Explicitly requesting a *null* parameter.
- *Object type parameters* Explicitly requesting an object in a given state.
- *Triangle example* The specification for a scalene triangle.

Structural tests

Test	Parameters	Specification
Parameter dependencies	a: int, b: String	b != $null$ && $b.Length = a$ && $a > 32$
Null object	a: Stack	$a = null$
Object type	a: Stack	a != $null$ && $a.Count >= 2$
Array type	a: int[]	a != $null$ && $a.Length > 2$ && $a[1] = 1$
Forall quantifier	a: List0 $< String >$	a != $null$ && $a.Count = 2$ && $\forall s \in a : s =" abc"$
Exists quantifier	a: List$< String >$	a != $null$ && $a.Count = 2$ && $\exists s \in a : s =" abc"$
Scalene triangle example [32]	a,b,c: double	$a + b > c$ && $b + c > a$ && $a + c > b$ && a != b && a != c && b != c

Each of the given specifications is used as precondition for a method
The evaluation result tables for each approach show for what type of specification the approach was able to generate satisfying data

IV. Conclusions

Tool input and output

	Required input	Optional input	Output
AgitarOne	Source		JUnit tests
AnalytiX	Source	Assertions	JUnit tests
AutoTest	Source	Eiffel specifications	Eiffel tests
C++test	Source + binary		Unit tests
Jtest	Source	Jcontract specification	JUnit tests
RANDOOP	Assembly	RANDOOP contracts/filters	JUnit tests
PEX	Source	Assertions, code contracts	Visual studio unit tests, NUnit tests, Mb unit tests, XUnit.net tests

Summary of required and optional input as well as the output of each evaluated tool

IV. Conclusions

Primitive data type benchmark results

	AgitarOne Sect. 3.3	AnalytiX Sect. 3.5	AutoTest Sect. 3.4	C++test Sect. 3.2	Jtest Sect. 3.1	RANDOOP Sect. 3.6	PEX Sect. 3.7
Boolean							
Constant	✓	✓	✓	✓	✓	✓	✓
Linear	✓	✓	✓	✓	✓	✓	✓
Non-linear	–	–	–	–	–	–	–
Inequality	✓	✓	✓	✓	✓	✓	✓
Character							
Constant	✓	✓	×	✓	✓	×	✓
Linear	–	–	–	–	–	–	–
Non-linear	–	–	–	–	–	–	–
Inequality	✓	✓	✓	✓	✓	×	✓
Integer							
Constant	✓	✓	✓	✓	✓	✓	✓
Linear	✓	✓	✓	✓	✓	×	✓
Non-linear	✓	✓	×	✓	✓	×	✓
Inequality	✓	✓	✓	✓	✓	✓	✓
Float							
Constant	✓	✓	×	✓	✓	×	✓
Linear	✓	✓	×	✓	✓	×	✓
Non-linear	✓	✓	×	✓	✓	×	✓
Inequality	✓	✓	✓	✓	✓	✓	✓
Double							
Constant	✓	✓	×	✓	✓	×	✓
Linear	✓	✓	×	✓	✓	×	✓
Non-linear	✓	✓	×	✓	✓	×	✓
Inequality	✓	✓	✓	✓	✓	✓	✓
String							
Constant	✓	✓	×	✓	✓	×	✓
Linear	✓	✓	–	–	✓	×	✓
Non-linear	–	–	–	–	–	–	–
Inequality	✓	✓	✓	✓	✓	✓	✓

IV. Conclusions

Object type benchmark results

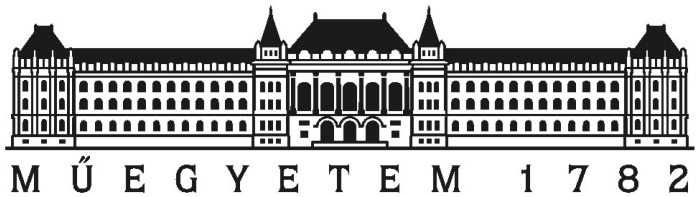
	AgitarOne Sect. 3.3	AnalytiX Sect. 3.5	AutoTest Sect. 3.4	C++test Sect. 3.2	Jtest Sect. 3.1	RANDOOP Sect. 3.6	PEX Sect. 3.7
Parameter dependencies	✓	✓	×	×	✓	×	✓
Null object	✓	✓	✓	×	✓	✓	✓
Object type	✓	×	×	✓	✓	✓	✓
Array type	✓	✓	×	×	✓	✓	✓
Forall quantifier	✓	×	×	×	×	×	✓
Exists quantifier	✓	×	×	×	×	×	✓
Scalene triangle example	✓	×	×	×	×	×	✓

IV. Conclusions

Tool generation techniques

	AgitarOne Sect. 3.3	AnalytiX Sect. 3.5	AutoTest Sect. 3.4	C++test Sect. 3.2	Jtest Sect. 3.1	RANDOOP Sect. 3.6	PEX Sect. 3.7
Primitive types							
Manual	✓	✓	✓	✓			
Pre-defined values		✓	✓	✓	✓	✓	
Random			✓	✓			
Constants extraction	✓	✓		✓	✓		
Constants extraction + manip.	✓						
Combinatorial testing		✓		✓	✓		
Constraint logic	✓						✓
Object types							
Null objects	✓	✓	✓		✓	✓	✓
Manual objects			✓	✓		✓	
Pre-defined objects							
Random constructor	✓	✓	✓	✓	✓	✓	
Random constructor + manip.	✓		✓			✓	
Mock/stub generation	✓	✓			✓		
Constraint logic							✓

This table lists all incorporated generation techniques of the evaluated tools with respect to primitive and object data types



Thank you for your attention!