



UMASS
AMHERST

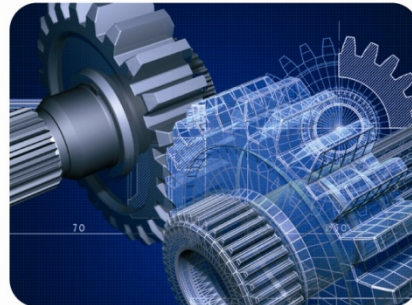
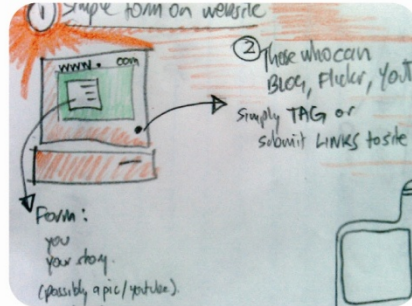


UNIVERSIDAD
POLITECNICA
DE VALENCIA



Centro de Investigación en Métodos
de Producción de Software

TOOL COMPETITION SESSION



Unit Testing Tool Competition Round Four

Urko Rueda, René Just,
Juan P. Galeotti, Tanja E. J. Vos


The 9th International Workshop on Search-Based Software Testing



1. About the Tool competition
2. The Tools
3. The Methodology
4. The Results
5. Lessons learned



Benchmarked Java unit testing at the class level

Unit Testing Tool Competition	FITTEST crest.cs.ucl.ac.uk/fittest	Coverage metrics	Mutation metrics	CUTs / Projects / Tools	Tools SBST & nonSBST
2012 ICST'13	✓	Cobertura	Javalanche	77 / 5 / 2	Manual & Randoop - baselines
2013 Round Two FITTEST'13		JaCoCo	PITest	63 / 9 / 4	1 st + T3 & Evosuite
				63 / 9 / 8	
2014 Round Three SBST'15		✗			2 nd + Commercial & GRT & jTexPert & Mosa(Evosuite)
2015 Round Four SBST'16	✗	Defects4J: github.com/rjust/defects4j + Real fault finding metric		68 / 5 / 4	Randoop - baseline & T3 & Evosuite & jTexPert



- Why?
 - Towards testing field maturity – this is just Java ...
 - Tools improvements, future developments insight
- What is new in the 4th edition?
 - Benchmark infrastructure – split into
 - Test generation
 - Test execution & Test assessment (Defects4J)
 - Benchmark subjects (from Defects4J dataset)
 - Time budgets (1, 2, 4 & 8 minutes)
 - Flaky tests (non compliant, non reliable pass)



- SBST and non-SBST tools
- Command line tools
- Fully automated – no human intervention

Tool	Technique	Static analysis	Edition			
			2012	2013	2014	2015
Randoop (baseline)	Random	x	✓	✓	✓	✓
T3		x	x	✓	✓	✓
jTexPert	Random (guided)	✓	x	x	✓	✓
Evosuite	Evolutionary algorithm	✓	x	✓	✓	✓



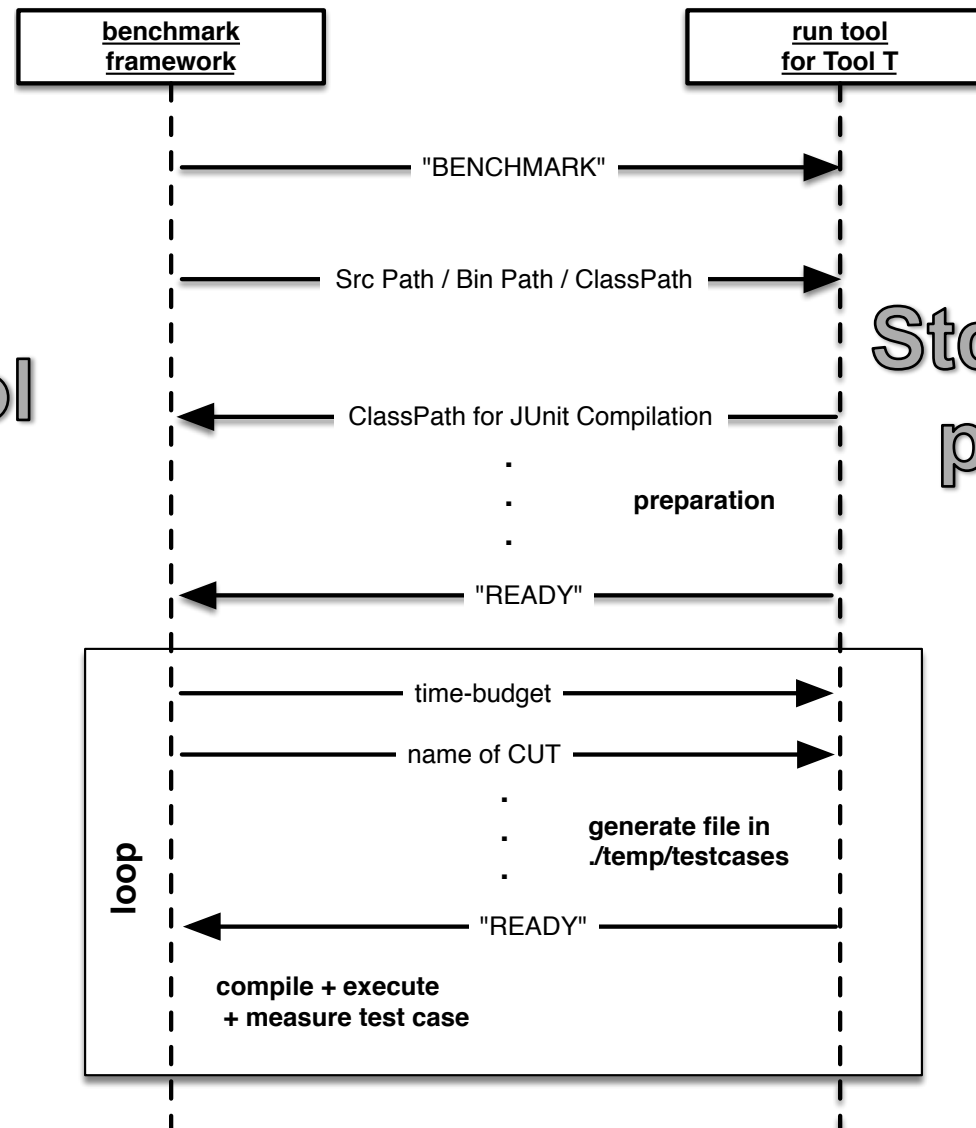
- Tool deployment
 - Installation – Linux environment
 - Wrapper implementation – runtool script
 - Std. IN/OUT communication protocol
 - 4th edition has a time budget
- Tune-up cycle – setup, run, resolve issues
 - Benchmark infrastructure
 - Defects4J integration
 - Decoupling test generation from test execution/assessment
 - Tool – run over non contest benchmark samples

The Methodology



runtool

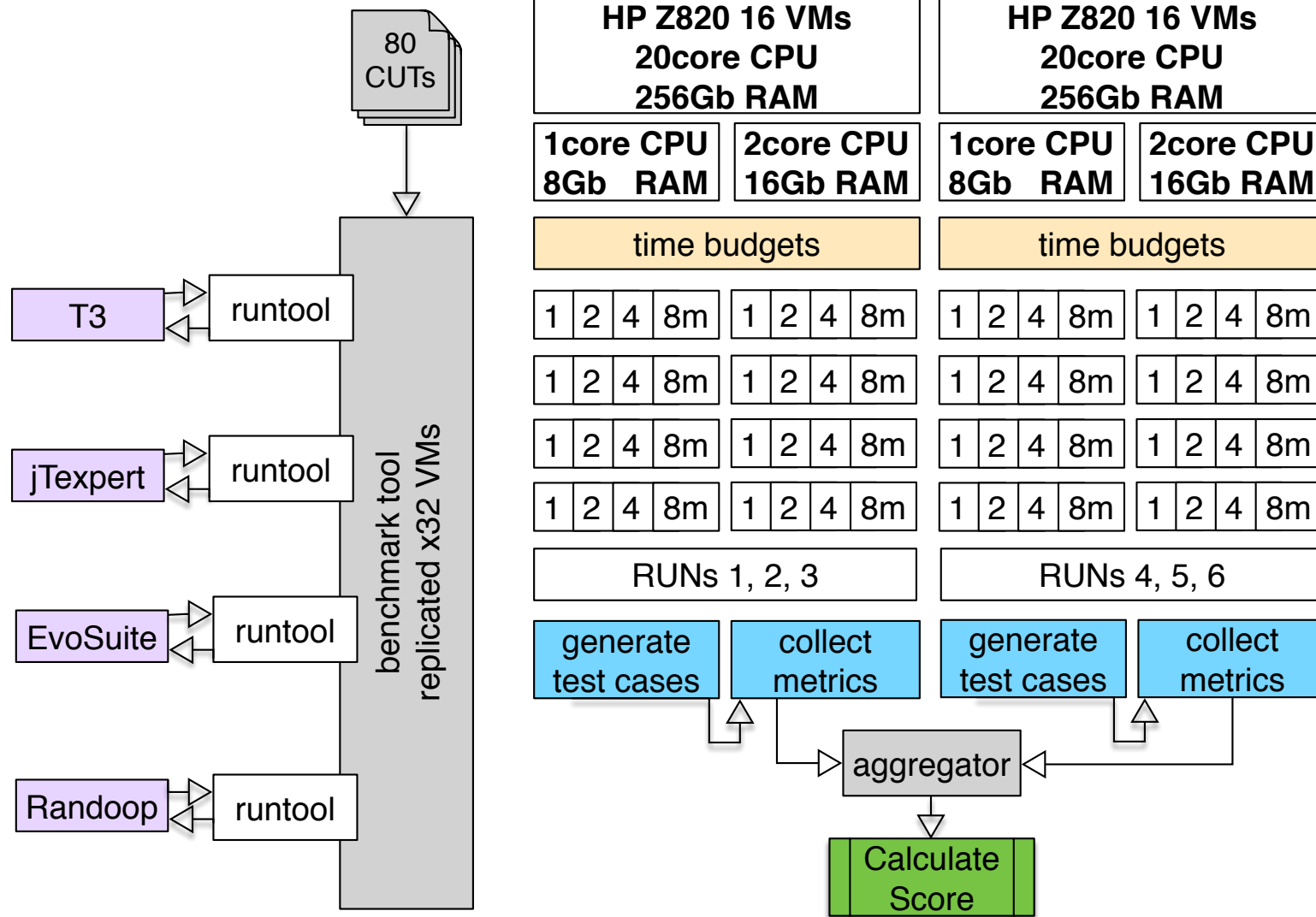
Std. IN/OUT
protocol



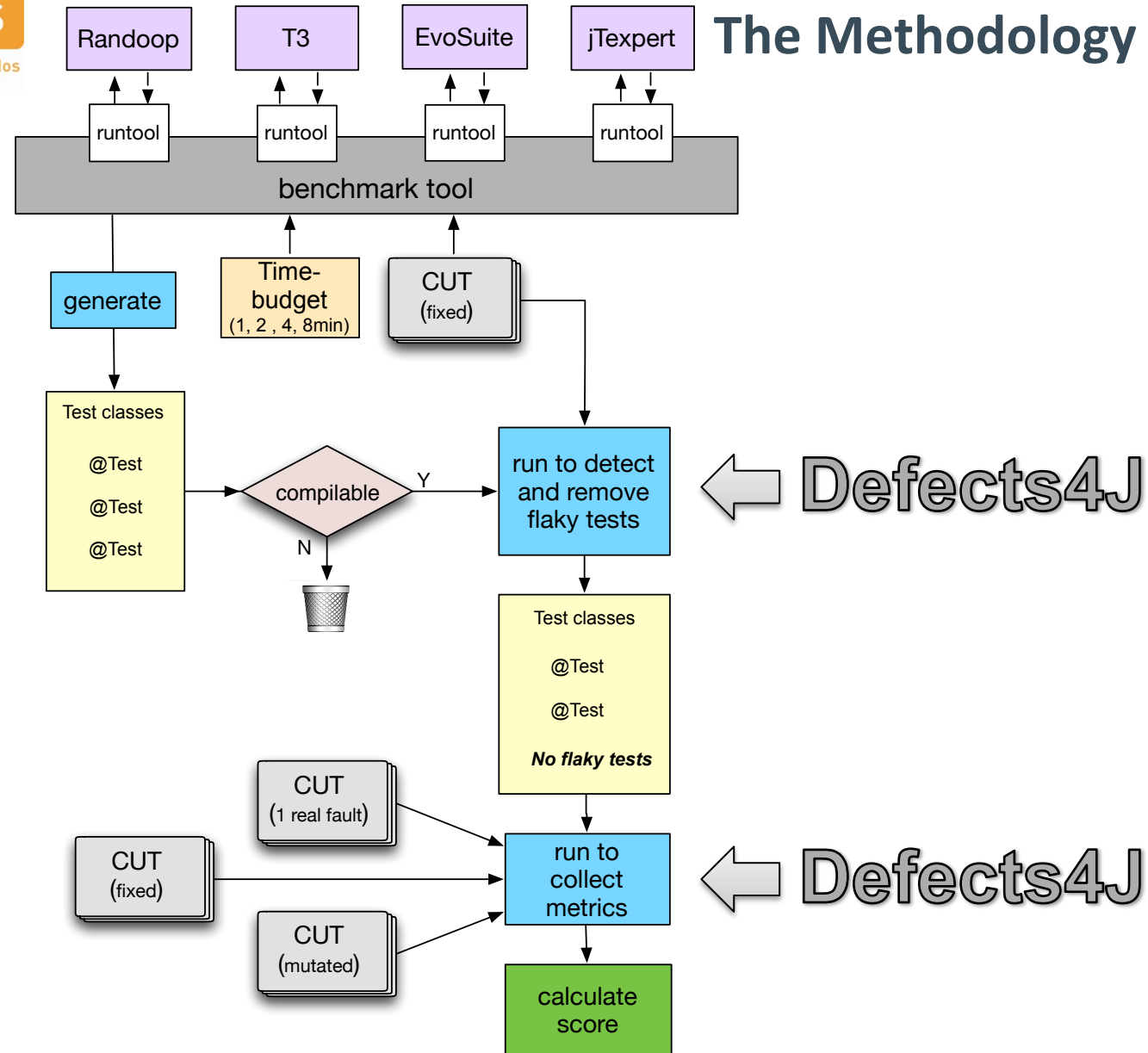


- Benchmark infrastructure
 - Two HP Z820 workstations – each:
 - 2 CPU sockets for a total of 20 cores
 - 256Gb RAM
 - 32 virtual machines (16 per workstation)
 - Test generation
 - 1 core – control tool multi-threading capability
 - 8GB RAM
 - Test execution/assessment (tool independent)
 - 2 cores
 - 16Gb RAM – resolves out of memory issues

The Methodology



The Methodology





- Flaky tests
 - Passes during generation
 - But, might Fail during execution/assessment
 - False-positive warnings
 - Non reliable fault-detection
 - Non reliable mutation analysis
- Defects4J flaky tests sanity
 - Non compiling test classes
 - Failing tests over 5 executions (fixed CUT versions)



- The Metrics – Test effectiveness
 - Code coverage (fixed benchmark versions)
 - Defects4J <- *Cobertura*
 - Statement coverage
 - Condition coverage
 - Mutation score
 - Defects4J <- *Major framework* (all mutation operators)
 - Real fault detection (buggy benchmark versions)
 - 1 real fault per benchmark
 - 0 or 1 score, independent of how many tests reveal it



- The Scoring formula

$$\text{covScore}_{(T,L,C,r)} := w_i \cdot \text{cov}_i + w_b \cdot \text{cov}_b + w_m \cdot \text{cov}_m + \\ (\text{real fault found ? } w_f : 0)$$

T = Tool; **L** = Time budget; **C** = CUT; **r** = RUN (1..6)

Coverages: **cov_i** = statement; **cov_b** = condition

cov_m = mutants kill ratio

Weights: **w_i** = 1; **w_b** = 2; **w_m** = 4; **w_f** = 4



- The Scoring formula – time penalty

$$tScore_{\langle T, L, C, r \rangle} := covScore_{\langle T, L, C, r \rangle} \cdot \min\left(1, \frac{L}{genTime}\right)$$

- Test generation slot: $L \dots 2 \cdot L$
- No penalty if $genTime \leq L$
- Penalty for Extra time taken ($genTime - L$)
 - Half $covScore$ if the Tool must be killed ($> 2 \cdot L$)



- The Scoring formula – tests penalty

$$penalty_{\langle T, L, C, r \rangle} := \begin{cases} 2 & \text{if no compilable test classes} \\ \frac{\#uClasses}{\#Classes} + \frac{\#fTests}{\#Tests} & \text{otherwise} \end{cases}$$

#Classes = generated test classes; **#uClasses** = uncompileable

#Tests = test cases; **#fTests** = flaky



- The Scoring formula – Tool score

$$\text{Score}(T, L, C, r) := \text{tScore}(T, L, C, r) - \text{penalty}(T, L, C, r)$$

$\text{Score}(T, L, C) := \text{avg}(\text{Score}(T, L, C, r) \text{ for all } r \text{ executions})$

$$\text{score}_T := \sum_{L, C} \text{Score}_{\langle T, L, C \rangle}$$



- Conclusion validity
 - Reliability of treatment implementation
 - Tool deployment instructions EQUAL for all participants
 - Reliability of measures
 - Efficiency: wall clock time by Java `System.currentTimeMillis()`
 - Effectiveness: Defects4J
 - Tools non-deterministic nature: 6 runs (HW Capacity)



- Internal validity
 - CUTs from Defects4J (uniform and arbitrary selection from 5 open source projects)
 - Tools and benchmark infrastructure Tune-up samples
 - Contest benchmarks
 - Wrappers *runtool*: implemented by Tools side
- Construct validity
 - Scoring formula weights – quality indicators value
 - Empirical studies – correlation of proxy metrics for:
Test effectiveness and Fault finding capability

Table 3: Overall scores for all tools.

Tool	Budget	Score	Std.dev
EvoSUITE	*	1127	136.96
T3	*	978	86.17
JTEXPERT	*	931	137.03
RANDOOOP	*	747	40.31

The Results



Table 4: Scores for all time budgets.

(OPTIMAL gives the maximum score and DEVELOPER the score achieved by the developer-written test suites).

Tool	Budget	Score	Std.dev
T3	1min	220	27.836
EvoSUITE	1min	209	33.564
JTEXPERT	1min	179	38.811
RANDOOOP	1min	155	11.500
EvoSUITE	2min	259	45.679
T3	2min	241	30.649
JTEXPERT	2min	231	41.199
RANDOOOP	2min	179	12.553
EvoSUITE	4min	318	58.304
T3	4min	253	27.687
JTEXPERT	4min	251	48.000
RANDOOOP	4min	197	16.254
EvoSUITE	8min	341	57.720
JTEXPERT	8min	270	47.830
T3	8min	263	27.687
RANDOOOP	8min	216	20.708
OPTIMAL	—	748	—
DEVELOPER	—	611	—

Contest run for ~1week

Test generation,
execution and
assessment

x32 VMs

A single virtual
machine would use
8 CPU months!

No time budgets



- Testing Tools improvements
 - Automation, Test effectiveness, Comparability
- Benchmarking infrastructure improvements
 - Decoupling Test gen. from execution/assessment
 - Flaky tests identification and sanity
 - Fault finding capability measurement
 - Test effectiveness due to Test generation time
 - What next?
 - Automated parallelization of the benchmark contest
 - More Tools, new languages? (i.e. C#?)



Universidad Politécnica de Valencia, ES

urueda@pros.upv.es, tvos@dsic.upv.es

Open Universiteit Heerlen, NL

tanja.vos@ou.nl

University of Massachusetts Amherst, MA, USA

rjust@cs.umass.edu

University of Buenos Aires, Argentina

jgaleotti@dc.uba.ar

web: <http://sbstcontest.dsic.upv.es/>