Advances in Automatic Software Testing: Test-Comp 2025

Dirk Beyer ✉

LMU Munich, Munich, Germany

Abstract. The 7th edition of the Competition on Software Testing (Test-Comp 2025) provides an overview and comparative evaluation of automatic test-suite generators for C programs. The experimental evaluation was performed on a benchmark set of 11 226 test-generation tasks for C programs. Each test-generation task consisted of a program and a test specification. The test specifications included error coverage (generate a test suite that exhibits a bug) and branch coverage (generate a test suite that executes as many program branches as possible). Test-Comp 2025 evaluated 20 software systems for test generation that are all freely available. This included 13 test-suite generators that participated with active support from teams led by 12 different representatives from 8 countries (actively maintained software systems, participation in competition jury). Test-Comp 2025 had 1 new participant (Sikraken new) and 2 re-entries (ESBMC-incr, ESBMC-kind). The evaluation included also 7 test-generation tools from previous years.

Keywords: Software Testing · Test-Case Generation · Competition · Program Analysis · Software Validation · Software Bugs · Test Validation · Test-Comp · Benchmarking · Test Coverage · Bug Finding · Test Suites · SV-Benchmarks · BenchExec · TestCov · CoVeriTeam

1 Introduction

In its 7th edition, the International Competition on Software Testing (Test-Comp, https://test-comp.sosy-lab.org, [10, 11, 12, 13, 14, 16, 17]) again compares automatic test-suite generators for C programs, in order to showcase the state of the art in the area of automatic software testing. This competition report is an update of the previous reports, referring to the rules and definitions, presents the competition results, and give some interesting data about the execution of the competition experiments. We use BenchExec [31] to execute the benchmark runs, BenchCloud [24] to distribute the execution to a large and elastic set of computers, FM-Weck [33] to

This report extends previous reports on Test-Comp [10, 11, 12, 13, 14, 16, 17] by providing new results, while the procedures and setup of the competition stay mainly unchanged. Reproduction packages are available on Zenodo (see Table 3).

✉ dirk.beyer@sosy.ifi.lmu.de

execute tools from previous years using the container with all their requirements fulfilled, and the FM-Tools [18] collection to look up all the information we need about the tools for test-case generation, including their versions, parameters, and jury representatives. The results are presented in tables and graphs, also on the competition web site (`https://test-comp.sosy-lab.org/2025/results`), and are available in the accompanying archives (see Table 3).

Competition Goals. In summary, the goals of Test-Comp are the following [11]:

- Establish *standards* for software test generation. This means, most prominently, to develop a standard for marking input values in programs, define an exchange format for test suites, agree on a specification language for test-coverage criteria, and define how to validate the resulting test suites.
- Establish a set of *benchmarks* for software testing in the community. This means to create and maintain a set of programs together with coverage criteria, and to make those publicly available for researchers to be used in performance comparisons when evaluating a new technique.
- Provide an overview of *available tools* for test-case generation and a snapshot of the state-of-the-art in software testing to the community. This means to compare, independently from particular paper projects and specific techniques, different test generators in terms of effectiveness and performance.
- Increase the visibility and credits that *tool developers* receive. This means to provide a forum for presentation of tools and discussion of the latest technologies, and to give the participants the opportunity to publish about the development work that they have done.
- Educate PhD students and other participants on how to set up performance experiments, package tools in a way that supports reproduction, and how to perform *robust and accurate research experiments*.
- Provide *resources* to development teams that do not have sufficient computing resources and give them the opportunity to obtain results from experiments on large benchmark sets.

Related Competitions. In the field of formal methods, competitions are respected as an important evaluation method and there are many competitions [8, 26]. We refer to the report from Test-Comp 2020 [11] for a more detailed discussion and give here only the references to the most related competitions: Competition on Software Verification (SV-COMP) [19], Competition on Search-Based Software Testing (SBST) [54], and the DARPA Cyber Grand Challenge [56]. For the techniques used for automatic software testing, we refer to the literature [5, 41].

2 Definitions, Formats, and Rules

Organizational aspects such as the classification (automatic, off-site, reproducible, jury, training) and the competition schedule is given in the initial competition definition [10]. In the following, we repeat some important definitions that are necessary to understand the results.

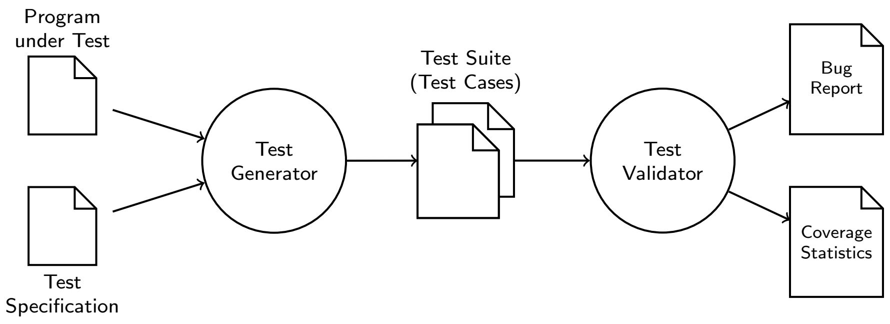

Fig. 1: Flow of the Test-Comp execution for one test generator (taken from [11])

Test-Generation Task. A *test-generation task* is a pair of an input program (program under test) and a test specification. A *test-generation run* is a non-interactive execution of a test generator on a single test-generation task, in order to generate a test suite according to the test specification. A *test suite* is a sequence of test cases, given as a directory of files according to the format for exchangeable test-suites.[1]

Execution of a Test Generator. Figure 1 illustrates the process of executing one test-suite generator on the benchmark suite. One test run for a test-suite generator gets as input (i) a program from the benchmark suite and (ii) a test specification (cover bug, or cover branches), and returns as output a test suite (i.e., a set of test cases). The test generator is contributed by a competition participant as a software archive in ZIP format on Zenodo, via a DOI entry of a version in the FM-Tools record of the test generator. All test runs are executed centrally by the competition organizer.

Execution of the Test Validator. The test-suite validator takes as input the test suite from the test generator and validates it by executing the program on all test cases: for bug finding it checks if the bug is exposed and for coverage it reports the coverage. We use the tool TestCov [30][2] as test-suite validator.

In Test-Comp 2025, we used TestCov in four configurations: (a) We use separate validations based on the compiler, with GCC and with Clang. The motivation for this is that the two different compilers use different choices for unspecified behavior, where the C standard leaves certain choices up to the compiler (for example, the unspecified order of evaluation of function arguments). (b) We use separate validations based on the formatting after instrumentation, with and without formatting. The motivation for this is that due to incompatibilities of the tools for formatting and coverage measurement, we would like to make sure to obtain the best possible coverage measurement by using those variants. For each test-validation run, the best of the four results is used to determine the score.

[1] `https://gitlab.com/sosy-lab/test-comp/test-format`

[2] `https://gitlab.com/sosy-lab/software/test-suite-validator`

Table 1: Coverage specifications used in Test-Comp 2025 (similar to 2019–2024)

Formula	Interpretation
`COVER EDGES(@CALL(reach_error))`	The test suite contains at least one test that executes function `reach_error`.
`COVER EDGES(@DECISIONEDGE)`	The test suite contains tests such that all branches of the program are executed.

Test Specification. The specification for testing a program is given to the test generator as input file (either `properties/coverage-error-call.prp` or `properties/coverage-branches.prp` for Test-Comp 2025).

The definition `init(main())` is used to define the initial states of the program under test by a call of function `main` (with no parameters). The definition `FQL(f)` specifies that coverage definition `f` should be achieved. The FQL (FShell query language [45]) coverage definition `COVER EDGES(@DECISIONEDGE)` means that all branches should be covered (typically used to obtain a standard test suite for quality assurance) and `COVER EDGES(@CALL(foo))` means that a call (at least one) to function `foo` should be covered (typically used for bug finding). A complete specification looks like: `COVER(init(main()), FQL(COVER EDGES(@DECISIONEDGE)))`.

Table 1 lists the two FQL formulas that are used in test specifications of Test-Comp 2025; there was no change from 2020 (except that special function `__VERIFIER_error` does not exist anymore).

Task-Definition Format 2.0. Test-Comp 2025 used again the task-definition format in version 2.0.

License and Qualification. The license of each participating test generator must allow its free use for reproduction of the competition results. The license for each tool is available in the FM-Tools entry for the tool, as well as in Table 4. Details on qualification criteria can be found in the competition report of Test-Comp 2019 [12].

3 Categories and Scoring Schema

Benchmark Programs. The input programs were taken from the largest and most diverse open-source repository of software-verification and test-generation tasks[3], which is also used by SV-COMP [19]. As in 2020 and 2021, we selected all programs for which the following properties were satisfied (see issue on GitLab[4] and report [12]):

[3] https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks

[4] https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/merge_requests/774

1. compiles with `gcc`, if a harness for the special methods[5] is provided,
2. should contain at least one call to a nondeterministic function,
3. does not rely on nondeterministic pointers,
4. does not have expected result 'false' for property 'termination', and
5. has expected result 'false' for property 'unreach-call' (only for category *Cover-Error*).

This selection yielded a total of 11 226 test-generation tasks, namely 1 215 tasks for category *Cover-Error* and 10 011 tasks for category *Cover-Branches*. The test-generation tasks are partitioned into categories, which are listed in Tables 6 and 7 and described in detail on the competition web site.[6] Figure 2 illustrates the category composition.

Category Cover-Error. The first category is to show the abilities to discover bugs. The benchmark set consists of programs that contain a bug. We produce for every tool and every test-generation task one of the following scores: 1 point, if the validator succeeds in executing the program under test on a generated test case that explores the bug (i.e., the specified function was called), and 0 points, otherwise.

Category Cover-Branches. The second category is to cover as many branches of the program as possible. The coverage criterion was chosen because many test generators support this standard criterion by default. Other coverage criteria can be reduced to branch coverage by transformation [44]. We produce for every tool and every test-generation task the coverage of branches of the program (as reported by TestCov [30]; a value between 0 and 1) that are executed for the generated test cases. The score is the returned coverage.

Max Over All Validators. As mentioned before, TestCov is executed four times on each test suite, using four different configurations. The score of a test suite is the maximum of the four computed scores.

Ranking. The ranking was decided based on the sum of points (normalized for meta categories). In case of a tie, the ranking was decided based on the run time, which is the total CPU time over all test-generation tasks. Opt-out from categories was possible and scores for categories were normalized based on the number of tasks per category (see competition report of SV-COMP 2013 [9], page 597).

4 Reproducibility

We followed the same competition workflow that was described in detail in the previous competition report (see Sect. 4, [13]). All major components that were used for the competition were made available in public version-control repositories. An overview of the components that contribute to the reproducible setup of Test-Comp is provided in Fig. 3, and the details are given in Table 2. We refer to the report of Test-Comp 2019 [12] for a thorough description of all

[5] https://test-comp.sosy-lab.org/2025/rules.php
[6] https://test-comp.sosy-lab.org/2025/benchmarks.php

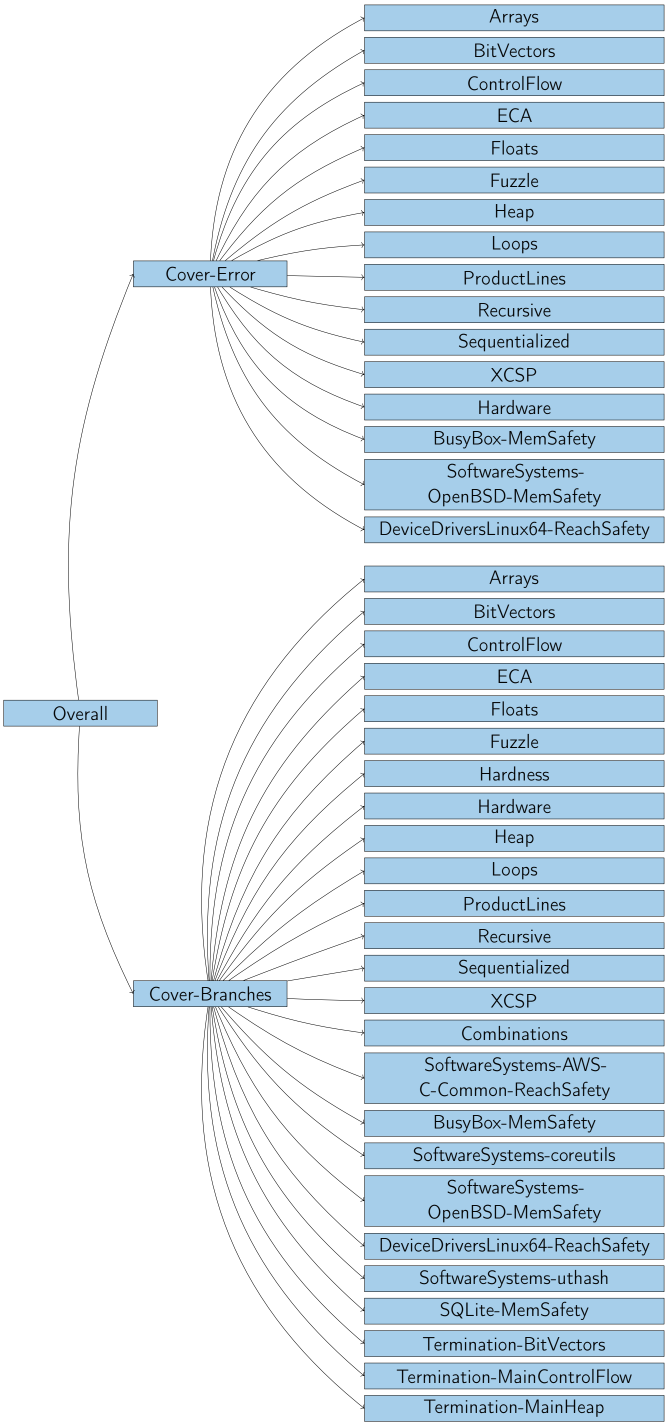

Fig. 2: Category structure for Test-Comp 2025

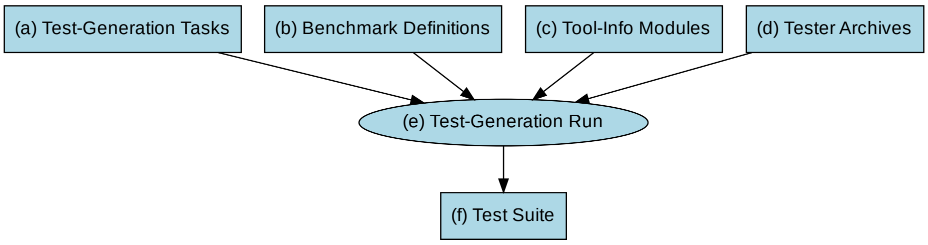

Fig. 3: Benchmarking components of Test-Comp and competition's execution flow (same as for Test-Comp 2020)

Table 2: Publicly available components for reproducing Test-Comp 2025

Component	Fig. 3	Repository at http://gitlab.com/sosy-lab/...	Version
Test-Generation Tasks	(a)	benchmarking/sv-benchmarks	testcomp25
Benchmark Definitions	(b)	test-comp/bench-defs	testcomp25
Tool-Info Modules	(c)	software/benchexec	3.29
Test-Generators	(d)	benchmarking/fm-tools	testcomp25
BenchExec (Benchmarking)	(e)	software/benchexec	3.29
BenchCloud (Distribution)	(e)	software/benchcloud	1.3.0
FM-Weck (Containers)	(e)	software/fm-weck	1.4.5
Test-Suite Format	(f)	test-comp/test-format	testcomp25
CoVeriTeam for CI		software/coveriteam	1.2.1
Processing Scripts		benchmarking/competition-scripts	testcomp25

Table 3: Artifacts published for Test-Comp 2025

Content	DOI	Reference
Test-Generation Tasks	10.5281/zenodo.15034421	[22]
Competition Results	10.5281/zenodo.15034433	[21]
Test-Suite Generators	10.5281/zenodo.15055359	[20]
Test Suites (Witnesses)	10.5281/zenodo.15034431	[23]
BenchExec	10.5281/zenodo.15007216	[61]
CoVeriTeam	10.5281/zenodo.11193690	[32]

components of the Test-Comp organization and how we ensure that all parts are publicly available for maximal reproducibility.

In order to guarantee long-term availability and immutability of the test-generation tasks, the produced competition results, and the produced test suites, we also packaged the material and published it at Zenodo (see Table 3).

The competition used CoVeriTeam [28][7] again to provide participants access to execution machines that are similar to actual competition machines. The competition report of SV-COMP 2022 provides a description on reproducing

[7] https://gitlab.com/sosy-lab/software/coveriteam

Table 4: Competition candidates with tool references and representing jury members; new indicates first-time participants, ∅ indicates inactive (hors concours) participation; licenses are abbreviated, see the hyperlink or tool page at FM-Tools for the specific version of the license; TestCov is the validator that computes the score for each test-suite

Tester	Ref.	License	Jury member	Affiliation
CetFuzz∅		Apache	–	–
CoVeriTest	[27, 47]	Apache	M.-C. Jakobs	LMU Munich, Germany
ESBMC-incr	[60]	Apache	C. Wei	U. of Manchester, UK
ESBMC-kind	[42, 60]	Apache	C. Wei	U. of Manchester, UK
FDSE	[62]	Apache	Z. Chen	National U. Defense Techn., China
Fizzer	[48, 49]	Zlib	M. Trtík	Masaryk U., Brno, Czechia
FuSeBMC	[3, 4]	MIT	K. Alshmrany	U. of Manchester, UK and Inst. Public Admin., Saudi Arabia
FuSeBMC-AI∅	[1, 2]	MIT	–	–
HybridTiger∅	[34, 55]	Apache	–	–
KLEEF	[53]	NCSA	A. Misonizhnik	Independent Researcher, Neutral
KLEE∅	[35, 36]	NCSA	–	–
Owi∅		AGPL	–	–
PRTest	[29, 51]	Apache	T. Lemberger	LMU Munich, Germany
Rizzer∅		Zlib	–	–
Sikraken new		LGPL	C. Meudec	South East Technological U., Ireland
Symbiotic	[37, 38]	MIT	M. Jonáš	Masaryk U., Brno, Czechia
TracerX	[40, 46]	Apache	J. Jaffar	National U. of Singapore, Singapore
TracerX-WP	[40, 46]	Apache	J. Jaffar	National U. of Singapore, Singapore
UTestGen	[6, 7]	LGPL	M. Barth	LMU Munich, Germany
WASP-C∅	[52]	Apache	–	–
TestCov	[30]	Apache	M. Kettl	LMU Munich, Germany

individual results and on trouble-shooting (see Sect. 3, [15]). A new component in Test-Comp 2025 was the use of the container solution FM-Weck [33], which makes it possible to include also older archives in the comparative evaluation, even if the tools were made for an older distribution of Ubuntu or use packages that are not available anymore. The tools can specify in their FM-Tools [18] entry a container in which they can run.

5 Results and Discussion

This section represents the results of the competition experiments. The report shall help to understand the state of the art and the advances in fully automatic test generation for whole C programs, in terms of effectiveness (test coverage, as accumulated in the score) and efficiency (resource consumption in terms of CPU time). All results mentioned in this article were inspected and approved by the participants.

Participating Test-Suite Generators. Table 4 provides an overview of the participating test generators and references to publications, as well as the team representatives of the jury of Test-Comp 2025. (The competition jury consists

Table 5: Technologies and features that the test generators used

Tester	**Algorithm Selection**	**Bit-Precise Analysis**	**Bounded Model Checking**	**CEGAR**	**Concurrency Support**	**Evolutionary Algorithms**	**Explicit-Value Analysis**	**Floating-Point Arithmetics**	**Guidance by Coverage Measures**	**Portfolio**	**Predicate Abstraction**	**Random Execution**	**Symbolic Execution**	**Targeted Input Generation**
CETFUZZ$^{\varnothing}$	✓					✓								
COVERITEST		✓		✓		✓	✓	✓		✓	✓	✓		
ESBMC-INCR		✓	✓		✓									
ESBMC-KIND		✓	✓		✓		✓	✓						
FDSE								✓	✓			✓	✓	
FIZZER		✓												
FUSEBMC			✓					✓	✓	✓				✓
FUSEBMC-AI$^{\varnothing}$			✓					✓	✓	✓				✓
HYBRIDTIGER$^{\varnothing}$				✓			✓	✓			✓			
KLEEF		✓						✓	✓				✓	✓
KLEE$^{\varnothing}$								✓					✓	✓
OWI$^{\varnothing}$		✓						✓				✓	✓	✓
PRTEST								✓				✓		
RIZZER$^{\varnothing}$		✓											✓	
SIKRAKEN new													✓	
SYMBIOTIC		✓			✓			✓	✓	✓			✓	✓
TRACERX			✓					✓					✓	✓
TRACERX-WP														
UTESTGEN				✓						✓				
WASP-C$^{\varnothing}$								✓				✓	✓	

of the chair and one member of each participating team.) An online table with information about all participating systems is provided on the competition web site.[8] Table 5 lists the features and technologies that are used in the test generators.

There are test generators that did not actively participate (tester archives taken from last year) and that are not included in rankings. Those are called *inactive* participation and the tools are labeled with a symbol ($^{\varnothing}$). In the past, we named those inactive tools 'hors concours', but since there could be other

[8] https://test-comp.sosy-lab.org/2025/systems.php

reasons for hors-concours participation (for example meta tools that consist of other participating tools), we now use the more specific term 'inactive'.

Computing Resources. The computing environment and the resource limits were the same as for Test-Comp 2024 [17], except for the upgraded operating system: Each test run was limited to 4 processing units (cores), 15 GB of memory, and 15 min of CPU time. The test-suite validation was limited to 2 processing units, 7 GB of memory, and 5 min of CPU time. The machines for running the experiments are part of a compute cluster that consists of 168 machines. Each machine had one Intel Xeon E3-1230 v5 CPU, with 8 processing units each, a frequency of 3.4 GHz, 33 GB of RAM, and a GNU/Linux operating system (x86_64-linux, Ubuntu 24.04 with Linux kernel 6.8). We used BenchExec [31] to measure and control computing resources (CPU time, memory, CPU energy), BenchCloud [24] to distribute, install, run, and clean-up test-case generation runs, and to collect the results, and FM-Weck [33] to prepare the correct container according to the tools' FM-Tools [18] entry. The values for CPU time are accumulated over all cores of the CPU. Further technical parameters of the competition machines are available in the repository which also contains the benchmark definitions. [9]

One complete test-generation execution of the competition consisted of 235 746 single test-generation run executions. The total CPU time was 3.7 years for one complete competition run for test generation (without validation). Test-suite validation consisted of 987 888 single test-suite validation runs. The total consumed CPU time was 0.95 years. Each tool was executed several times, in order to make sure no installation issues occur during the execution. Including preruns, the infrastructure managed a total of 968 364 test-generation runs (consuming 4.9 years of CPU time). The prerun test-suite validation consisted of 4 212 084 single test-suite validation runs (consuming 3.8 years of CPU time).

Quantitative Results. The quantitative results are presented in the same way as last year: Table 6 presents the quantitative overview of all tools and all categories. The head row mentions the category and the number of test-generation tasks in that category. The tools are listed in alphabetical order; every table row lists the scores of one test generator. We indicate the top three candidates by formatting their scores in bold face and in larger font size. An empty table cell means that the test generator opted-out from the respective main category (perhaps participating in subcategories only, restricting the evaluation to a specific topic). More information (including interactive tables, quantile plots for every category, and also the raw data in XML format) is available on the competition web site [10] and in the results artifact (see Table 3). Table 7 reports the top three test generators for each category. The consumed run time (column 'CPU Time') is given in hours and the consumed energy (column 'Energy') is given in kWh.

Score-Based Quantile Functions for Quality Assessment. We use score-based quantile functions [31] because these visualizations make it easier to understand the results of the comparative evaluation. The web site [10] and the results

[9] `https://gitlab.com/sosy-lab/test-comp/bench-defs/tree/testcomp25`
[10] `https://test-comp.sosy-lab.org/2025/results`

Table 6: Quantitative overview over all results; empty cells mark opt-outs; $^{\text{new}}$ indicates first-time participants, $^{\varnothing}$ indicates hors-concours participation

Participant	**Cover-Error** 1215 tasks	**Cover-Branches** 10011 tasks	**Overall** 11226 tasks
CetFuzz$^{\varnothing}$	323	2524	2906
CoVeriTest	552	4959	5333
ESBMC-incr	679	4380	5591
ESBMC-kind	680	4323	5565
FDSE	729	**5468**	6435
Fizzer	736	5429	**6446**
FuSeBMC	**994**	**5656**	**7763**
FuSeBMC-AI$^{\varnothing}$	853	4077	6228
HybridTiger$^{\varnothing}$	438	3866	4193
KLEE$^{\varnothing}$	804	3065	5434
KLEEF	**969**	**5734**	**7692**
Owi$^{\varnothing}$	281	2462	2677
PRTest	211	3191	2764
Rizzer$^{\varnothing}$	608		
Sikraken$^{\text{new}}$		2469	
Symbiotic	**743**	4207	5793
TracerX	390	3327	3667
TracerX-WP	349	3275	3447
UTestGen	439	4393	4492
WASP-C$^{\varnothing}$	554	2740	4094

artifact (Table 3) include such a plot for each category; as example, we show the plot for category *Overall* (all test-generation tasks) in Fig. 4. We had 18 test generators participating in category *Overall*, for which the quantile plot shows the overall performance over all categories (scores for meta categories are normalized [9]). A more detailed discussion of score-based quantile plots for testing is provided in the Test-Comp 2019 competition report [12].

Table 7: Overview of the top-three test generators for each category (measurement values for CPU time rounded to two significant digits, in hours)

Rank	Tester	Score	CPU Time
Cover-Error			
1	**FuSeBMC**	**994**	75
2	KLEEF	969	9.5
3	Symbiotic	743	5.5
Cover-Branches			
1	**KLEEF**	**5734**	1 500
2	FuSeBMC	5656	2 500
3	FDSE	5468	2 200
Overall			
1	**FuSeBMC**	**7763**	2 600
2	KLEEF	7692	1 500
3	Fizzer	6446	2 100

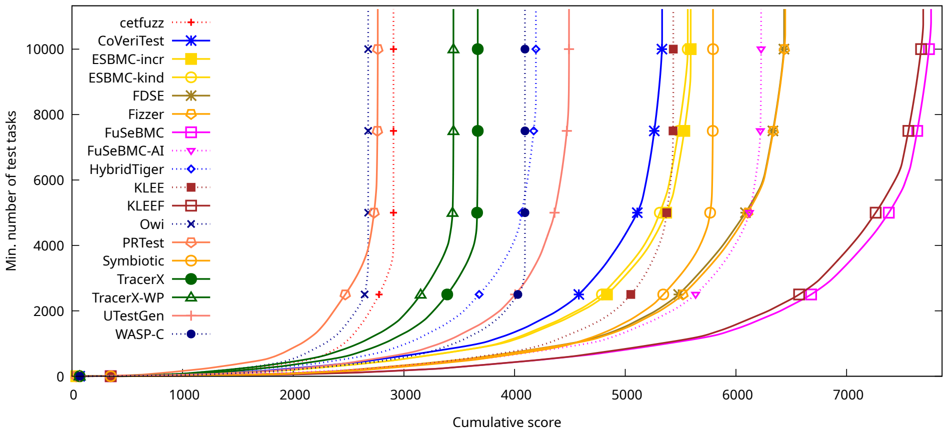

Fig. 4: Quantile functions for category *Overall*. Each quantile function illustrates the quantile (x-coordinate) of the scores obtained by test-generation runs below a certain number of test-generation tasks (y-coordinate). More details were given previously [12]. The graphs are decorated with symbols to make them better distinguishable without color.

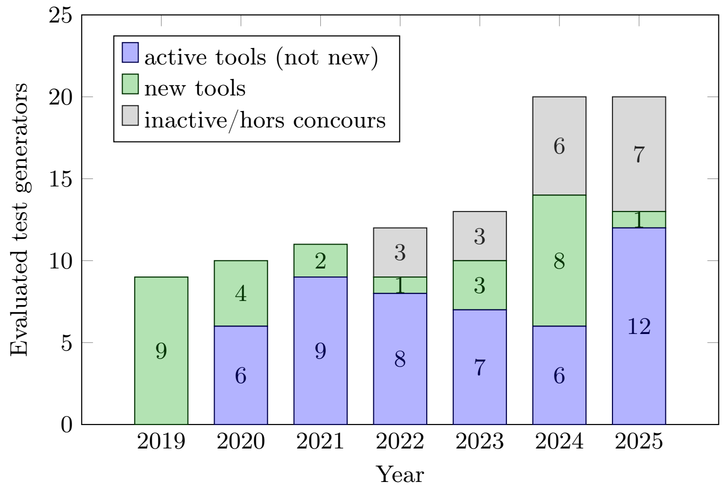

Fig. 5: Number of evaluated test generators for each year (blue/bottom: active participants from previous years, green/middle: number of first-time participants, gray/top: inactive participants from previous years)

6 Conclusion

The 7th Competition on Software Testing continues to provide an overview of fully-automatic test-generation tools for C programs. A total of 20 test-suite generators was compared (see Fig. 5 for the participation numbers and Table 4 for the details). This off-site competition uses a benchmark infrastructure that makes the execution of the experiments fully-automatic and reproducible. Transparency is ensured by making all components available in public repositories and have a jury (consisting of members from each team) that oversees the process. All test suites were validated by the test-suite validator TestCov [30] to measure the coverage. For the first time, the competition used several different validation runs for each test suite, in order to obtain the best possible coverage result, using different compiler backends and different formatting choices after instrumentation for coverage measurement. The results of the competition were presented at the 28th International Conference on Fundamental Approaches to Software Engineering (FASE) at ETAPS 2025 in Hamilton, Canada.

Data-Availability Statement. The test-generation tasks and results of the competition are published at Zenodo, as described in Table 3. All components and data that are necessary for reproducing the competition are available in public version repositories, as specified in Table 2. For easy access, the results are presented also online on the competition web site `https://test-comp.sosy-lab.org/2025/results`.

Funding Statement. This project was funded in part by the Deutsche Forschungsgemeinschaft (DFG) — 418257054 (Coop).

References

1. Aldughaim, M., Alshmrany, K.M., Gadelha, M.R., de Freitas, R., Cordeiro, L.C.: FuSeBMC_IA: Interval analysis and methods for test-case generation (competition contribution). In: Proc. FASE. pp. 324–329. LNCS 13991, Springer (2023). https://doi.org/10.1007/978-3-031-30826-0_18
2. Aldughaim, M., Alshmrany, K.M., Mustafa, M., Cordeiro, L.C., Stancu, A.: Bounded model checking of software using interval methods via contractors. arXiv/CoRR **2012**(11245) (December 2020). https://doi.org/10.48550/arXiv.2012.11245
3. Alshmrany, K., Aldughaim, M., Cordeiro, L., Bhayat, A.: FuSeBMC v.4: Smart seed generation for hybrid fuzzing (competition contribution). In: Proc. FASE. pp. 336–340. LNCS 13241, Springer (2022). https://doi.org/10.1007/978-3-030-99429-7_19
4. Alshmrany, K.M., Aldughaim, M., Bhayat, A., Cordeiro, L.C.: FuSeBMC: An energy-efficient test generator for finding security vulnerabilities in C programs. In: Proc. TAP. pp. 85–105. Springer (2021). https://doi.org/10.1007/978-3-030-79379-1_6
5. Anand, S., Burke, E.K., Chen, T.Y., Clark, J.A., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., McMinn, P.: An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software **86**(8), 1978–2001 (2013). https://doi.org/10.1016/j.jss.2013.02.061
6. Barth, M., Dietsch, D., Heizmann, M., Jakobs, M.C.: Ultimate TestGen: Test case generation with automata-based software model checking (competition contribution). In: Proc. FASE. pp. 326–330. LNCS 14573, Springer (2024). https://doi.org/10.1007/978-3-031-57259-3_20
7. Barth, M., Jakobs, M.C.: Test-case generation with automata-based software model checking. In: Proc. SPIN. Springer (2024). https://doi.org/10.1007/978-3-031-66149-5_14
8. Bartocci, E., Beyer, D., Black, P.E., Fedyukovich, G., Garavel, H., Hartmanns, A., Huisman, M., Kordon, F., Nagele, J., Sighireanu, M., Steffen, B., Suda, M., Sutcliffe, G., Weber, T., Yamada, A.: TOOLympics 2019: An overview of competitions in formal methods. In: Proc. TACAS (3). pp. 3–24. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_1
9. Beyer, D.: Second competition on software verification (Summary of SV-COMP 2013). In: Proc. TACAS. pp. 594–609. LNCS 7795, Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_43
10. Beyer, D.: Competition on software testing (Test-Comp). In: Proc. TACAS (3). pp. 167–175. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_11
11. Beyer, D.: Second competition on software testing: Test-Comp 2020. In: Proc. FASE. pp. 505–519. LNCS 12076, Springer (2020). https://doi.org/10.1007/978-3-030-45234-6_25
12. Beyer, D.: First international competition on software testing (Test-Comp 2019). Int. J. Softw. Tools Technol. Transf. **23**(6), 833–846 (December 2021). https://doi.org/10.1007/s10009-021-00613-3
13. Beyer, D.: Status report on software testing: Test-Comp 2021. In: Proc. FASE. pp. 341–357. LNCS 12649, Springer (2021). https://doi.org/10.1007/978-3-030-71500-7_17
14. Beyer, D.: Advances in automatic software testing: Test-Comp 2022. In: Proc. FASE. pp. 321–335. LNCS 13241, Springer (2022). https://doi.org/10.1007/978-3-030-99429-7_18

15. Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS (2). pp. 375–402. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_20
16. Beyer, D.: Software testing: 5th comparative evaluation: Test-Comp 2023. In: Proc. FASE. pp. 309–323. LNCS 13991, Springer (2023). https://doi.org/10.1007/978-3-031-30826-0_17
17. Beyer, D.: Automatic testing of C programs: Test-Comp 2024. Springer (2024)
18. Beyer, D.: Find, use, and conserve tools for formal methods. In: Proc. Festschrift Podelski 65th Birthday. Springer (2024), available online: https://www.sosy-lab.org/research/pub/2024-Podelski65.Find_Use_and_Conserve_-Tools_for_Formal_Methods.pdf
19. Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: Proc. TACAS (3). pp. 299–329. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_15
20. Beyer, D.: FM-Tools Release 2.2: Data set of metadata about tools for formal methods (SV-COMP 2025, Test-Comp 2025). Zenodo (2025). https://doi.org/10.5281/zenodo.15055359
21. Beyer, D.: Results of the 7th Intl. Competition on Software Testing (Test-Comp 2025). Zenodo (2025). https://doi.org/10.5281/zenodo.15034433
22. Beyer, D.: SV-Benchmarks: Benchmark set for software testing (Test-Comp 2025). Zenodo (2025). https://doi.org/10.5281/zenodo.15034421
23. Beyer, D.: Test suites from test-generation tools (Test-Comp 2025). Zenodo (2025). https://doi.org/10.5281/zenodo.15034431
24. Beyer, D., Chien, P.C., Jankola, M.: BenchCloud: A platform for scalable performance benchmarking. In: Proc. ASE. pp. 2386–2389. ACM (2024). https://doi.org/10.1145/3691620.3695358
25. Beyer, D., Chlipala, A.J., Henzinger, T.A., Jhala, R., Majumdar, R.: Generating tests from counterexamples. In: Proc. ICSE. pp. 326–335. IEEE (2004). https://doi.org/10.1109/ICSE.2004.1317455
26. Beyer, D., Hartmanns, A., Kordon, F.: TOOLympics Challenge 2023: Updates, Results, Successes of the Formal-Methods Competitions. LNCS 14550, Springer (2024). https://doi.org/10.1007/978-3-031-67695-6
27. Beyer, D., Jakobs, M.C.: CoVeriTest: Cooperative verifier-based testing. In: Proc. FASE. pp. 389–408. LNCS 11424, Springer (2019). https://doi.org/10.1007/978-3-030-16722-6_23
28. Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. pp. 561–579. LNCS 13243, Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_31
29. Beyer, D., Lemberger, T.: Software verification: Testing vs. model checking. In: Proc. HVC. pp. 99–114. LNCS 10629, Springer (2017). https://doi.org/10.1007/978-3-319-70389-3_7
30. Beyer, D., Lemberger, T.: TestCov: Robust test-suite execution and coverage measurement. In: Proc. ASE. pp. 1074–1077. IEEE (2019). https://doi.org/10.1109/ASE.2019.00105
31. Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: Requirements and solutions. Int. J. Softw. Tools Technol. Transfer **21**(1), 1–29 (2019). https://doi.org/10.1007/s10009-017-0469-y
32. Beyer, D., Wachowitz, H.: Coveriteam Release 1.2.1. Zenodo (2024). https://doi.org/10.5281/zenodo.11193690

33. Beyer, D., Wachowitz, H.: FM-Weck: Containerized execution of formal-methods tools. In: Proc. FM. pp. 39–47. LNCS 14934, Springer (2024). https://doi.org/10.1007/978-3-031-71177-0_3
34. Bürdek, J., Lochau, M., Bauregger, S., Holzer, A., von Rhein, A., Apel, S., Beyer, D.: Facilitating reuse in multi-goal test-suite generation for software product lines. In: Proc. FASE. pp. 84–99. LNCS 9033, Springer (2015). https://doi.org/10.1007/978-3-662-46675-9_6
35. Cadar, C., Dunbar, D., Engler, D.R.: Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. OSDI. pp. 209–224. USENIX Association (2008)
36. Cadar, C., Nowack, M.: Klee symbolic execution engine in 2019 (competition contribution). Int. J. Softw. Tools Technol. Transf. **23**(6), 867 – 870 (December 2021). https://doi.org/10.1007/s10009-020-00570-3
37. Chalupa, M., Novák, J., Strejček, J.: Symbiotic 8: Parallel and targeted test generation (competition contribution). In: Proc. FASE. pp. 368–372. LNCS 12649, Springer (2021). https://doi.org/10.1007/978-3-030-71500-7_20
38. Chalupa, M., Strejček, J., Vitovská, M.: Joint forces for memory safety checking. In: Proc. SPIN. pp. 115–132. Springer (2018). https://doi.org/10.1007/978-3-319-94111-0_7
39. Cok, D.R., Déharbe, D., Weber, T.: The 2014 SMT competition. JSAT **9**, 207–242 (2016)
40. Dutta, A., Maghareh, R., Jaffar, J., Godboley, S., Yu, X.L.: TracerX: Pruning dynamic symbolic execution with deletion and weakest precondition interpolation (competition contribution). In: Proc. FASE. pp. 320–325. LNCS 14573, Springer (2024). https://doi.org/10.1007/978-3-031-57259-3_19
41. Fraser, G., Wotawa, F., Ammann, P.: Testing with model checkers: A survey. STVR **19**(3), 215–261 (2009). https://doi.org/10.1002/stvr.402
42. Gadelha, M.Y., Ismail, H.I., Cordeiro, L.C.: Handling loops in bounded model checking of C programs via k-induction. Int. J. Softw. Tools Technol. Transf. **19**(1), 97–114 (February 2017). https://doi.org/10.1007/s10009-015-0407-9
43. Godefroid, P., Sen, K.: Combining model checking and testing. In: Handbook of Model Checking, pp. 613–649. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_19
44. Harman, M., Hu, L., Hierons, R.M., Wegener, J., Sthamer, H., Baresel, A., Roper, M.: Testability transformation. IEEE Trans. Softw. Eng. **30**(1), 3–16 (2004). https://doi.org/10.1109/TSE.2004.1265732
45. Holzer, A., Schallhart, C., Tautschnig, M., Veith, H.: How did you specify your test suite. In: Proc. ASE. pp. 407–416. ACM (2010). https://doi.org/10.1145/1858996.1859084
46. Jaffar, J., Murali, V., Navas, J.A., Santosa, A.E.: Tracer: A symbolic execution tool for verification. In: Proc. CAV. pp. 758–766. LNCS 7358, Springer (2012). https://doi.org/10.1007/978-3-642-31424-7_61
47. Jakobs, M.C., Richter, C.: CoVeriTest with adaptive time scheduling (competition contribution). In: Proc. FASE. pp. 358–362. LNCS 12649, Springer (2021). https://doi.org/10.1007/978-3-030-71500-7_18
48. Jonáš, M., Strejček, J., Trtík, M.: Fizzer with local space fuzzing (competition contribution). In: Proc. FASE. LNCS , Springer (2025)
49. Jonáš, M., Strejček, J., Trtík, M., Urban, L.: Fizzer: New gray-box fuzzer (competition contribution). In: Proc. FASE. pp. 309–313. LNCS 14573, Springer (2024). https://doi.org/10.1007/978-3-031-57259-3_17

50. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976). https://doi.org/10.1145/360248.360252
51. Lemberger, T.: Plain random test generation with PRTest (competition contribution). Int. J. Softw. Tools Technol. Transf. **23**(6), 871–873 (December 2021). https://doi.org/10.1007/s10009-020-00568-x
52. Marques, F., Santos, J.F., Santos, N., Adão, P.: Concolic execution for webassembly (artifact). Dagstuhl Artifacts Series **8**(2), 20:1–20:3 (2022). https://doi.org/10.4230/DARTS.8.2.20
53. Misonizhnik, A., Morozov, S., Kostyukov, Y., Kalugin, V., Babushkin, A., Mordvinov, D., Ivanov, D.: KLEEF: Symbolic execution engine (competition contribution). In: Proc. FASE. pp. 314–319. LNCS 14573, Springer (2024). https://doi.org/10.1007/978-3-031-57259-3_18
54. Panichella, S., Gambi, A., Zampetti, F., Riccio, V.: SBST tool competition 2021. In: Proc. SBST. pp. 20–27. IEEE (2021). https://doi.org/10.1109/SBST52555.2021.00011
55. Ruland, S., Lochau, M., Jakobs, M.C.: HybridTiger: Hybrid model checking and domination-based partitioning for efficient multi-goal test-suite generation (competition contribution). In: Proc. FASE. pp. 520–524. LNCS 12076, Springer (2020). https://doi.org/10.1007/978-3-030-45234-6_26
56. Song, J., Alves-Foss, J.: The DARPA cyber grand challenge: A competitor's perspective, part 2. IEEE Security and Privacy **14**(1), 76–81 (2016). https://doi.org/10.1109/MSP.2016.14
57. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A cross-community infrastructure for logic solving. In: Proc. IJCAR, pp. 367–373. LNCS 8562, Springer (2014). https://doi.org/10.1007/978-3-319-08587-6_28
58. Sutcliffe, G.: The CADE ATP system competition: CASC. AI Magazine **37**(2), 99–101 (2016). https://doi.org/10.1609/aimag.v37i2.2620
59. Visser, W., Păsăreanu, C.S., Khurshid, S.: Test-input generation with Java PathFinder. In: Proc. ISSTA. pp. 97–107. ACM (2004). https://doi.org/10.1145/1007512.1007526
60. Wei, C., Wu, T., Menezes, R.S., Shmarov, F., Aljaafari, F., Godboley, S., Alshmrany, K., de Freitas, R., Cordeiro, L.: Esbmc v7.7: Automating branch-coverage analysis using CFG-based instrumentation and smt solving (competition contribution). In: Proc. FASE. LNCS , Springer (2025)
61. Wendler, P., Beyer, D.: sosy-lab/benchexec: Release 3.29. Zenodo (2025). https://doi.org/10.5281/zenodo.15007216
62. Zhang, G., Shuai, Z., Ma, K., Liu, K., Chen, Z., Wang, J.: FDSE: Enhance symbolic execution by fuzzing-based pre-analysis (competition contribution). In: Proc. FASE. pp. 304–308. LNCS 14573, Springer (2024). https://doi.org/10.1007/978-3-031-57259-3_16