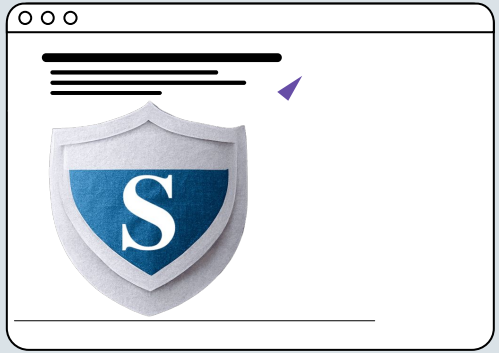
VALIDATION

OF CODE ANALYZERS

How to benchmark tools from system programming



ABOUT ME

Andrey Kuleshov

https://github.com/**akuleshov7**

intel. Deutsche Bank HUAWEI

---

save-cli Public

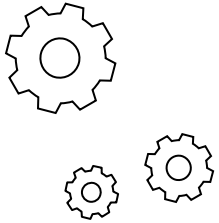
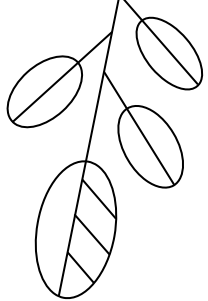
Universal test framework for cli tools [mainly for code analyzers and compilers]

Kotlin ☆ 33 4

save-cloud Public

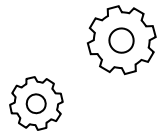
Cluster-based cloud mechanism for running SAVE framework

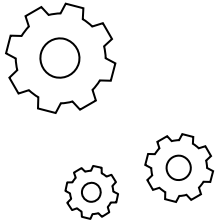
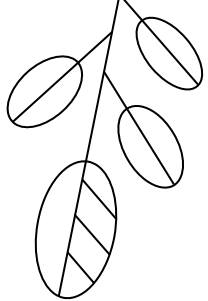
Kotlin ☆ 29 1

DISCLAIMER:

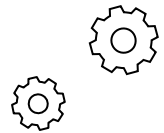
No testing «philosophy»
No marketing
Only personal experience
And some open-source projects

SCOPE OF THIS TALK

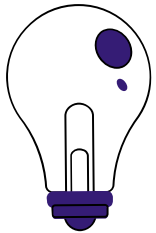
System programming tools
Compilers, code analyzers
A lot about existing benchmarks
Functional testing (90%)

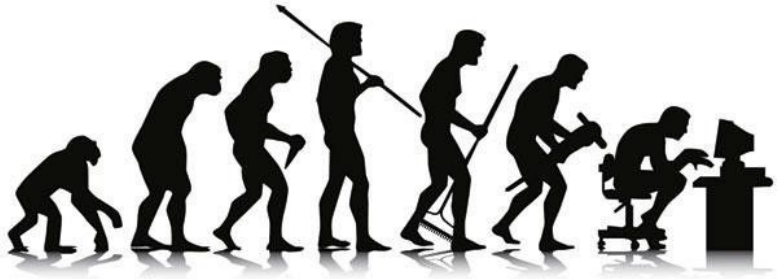
The price of the **mistake** of
system programming tools is extremely high

01 PROBLEM

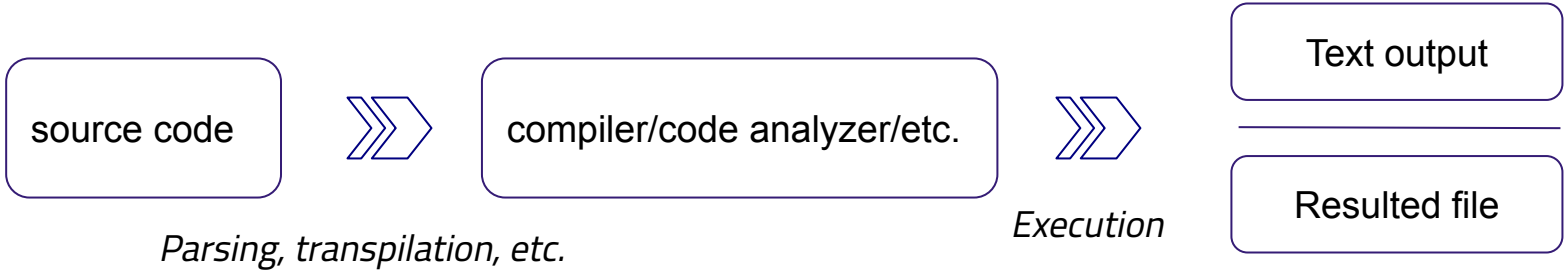
MOTIVATION OF "SAVE"



Let’s start from the origins…



HOW THESE TOOLS USUALLY WORK



JUST AN EXAMPLE

https://github.com/saveourtool/diktat

Test.kt:

```
fun main() {
println("Hello world")
}
```

```
$ ./ktlint -R diktat.jar -F Test.kt
$ ./ktlint -R diktat.jar Test.kt
```

```
Test.kt:1:1: [HEADER_MISSING]
Test.kt:1:1: [PACKAGE_NAME_MISSING]
Test.kt:2:1: [WRONG_INDENTATION]
Test.kt:2:5: [DEBUG_PRINT]
```

```
package com.huawei.test

fun main() {
    println("Hello world")
}
```

COMPILER'S EXAMPLE

https://clang.llvm.org/

test.c:

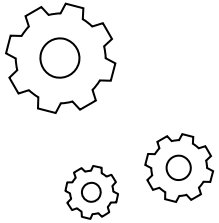
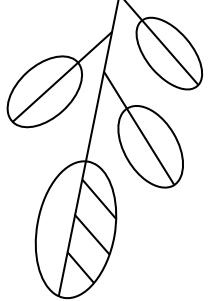
```c
#include <stdio.h>
int main(int argc, char **argv) {
    printf("hello world\n");
}
```

```
$ clang test.c
$ ./a.out

$ clang -fsyntax-only test.c

$ clang test.c -S -emit-llvm -o -
```

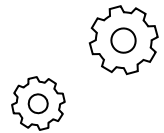
```
hello world
```

```
test.c:2:17: warning:
```

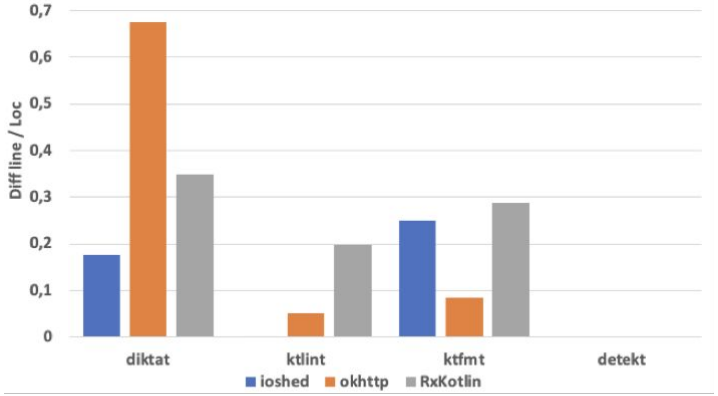
```
. . .
define i32 @main(i32 %0, i8** %1) #0 {
  %3 = alloca i32, align 4
  %4 = alloca i8**, align 8
. . .
```

So we need a very simple, but universal cli-framework.
What else?

BENCHMARKS AND COMPARISON



A figure from our paper for ISSRE conference comparing opensource code fixers for Kotlin

A Comparison of Static Analysis Tools for Vulnerability Detection in C/C++ Code

Andrei Arusoaie*, Ștefan Ciobâcă*, Vlad Crăciun*†, Dragoș Gavriluț†, Dorel Lucanu*
*Faculty of Computer Science, Alexandru Ioan Cuza University, Iași
Email: arusoaie.andrei,stefan.ciobaca,dorel.lucanu@...
†Bitdefender, Iași
Email: vcraciun,dgavrilut@bitde...

2009 Third International Conference on Emerging Security Information, Systems and Technologies

Probing into Code Analysis Tools

A Comparison of C# Supporting Static Code Analyzers

Rida Shaukat[1], Arooba Shahoor[2], Aniqa Ur...
Department of Software Engineering, Fatima Jin... Rawalpindi, ...
ridashaukat1996@gmail.com[1], arooba.shahoor@...

Abstract—Essence of a software system lies in the quality of source code and the degree to which it follows the underlying coding standard. The widely-used code analysis techniques focus upon examining the programs without actually executing them. The major purpose is the detection of complex code constructs and the potential defects that result in the decrement in quality of the code. The code complexity is often assumed to inflate the maintenance cost and lead to the unexpected system behaviour. A number of code analyzing tools are available and currently many researches are being conducted to improve software quality, bring down software complexity without affecting its external behaviour. A brief review of the existing code analyzers is presented in this paper; however the major focus is upon the tools that analyze the source code written in C#. Researchers round the globe have identified that the potential problems in the source code are; code smells, code clones, anti-detect pattern etc that often lead to the increase in system complexity and hence amplification in the system response time. The code analyzing

With th... comes th... efficiency...

It has ... issues are a... software des... realize that s... whole new a... within our so... analyzing tools ... transferred to ... programmers fro... failure that the lo... detecting the origin... Software Develop... maintainability.

Comparison of Static Code Analysis Tools

Matti Mantere and Ilkka Uusitalo
VTT Technical Research Centre of Finland
Kaitoväylä 1, Oulu, Finland
firstname.lastname@vtt.fi

Röning
Secure Programming group
ty of Oulu, Finland
iing@oulu.fi

valuable resource if applied with skill [5, p. 47-70].

In this paper we discuss the possible benefits of incorporating source code analysis tools into the software development process. We consider three different static analysis tools and demonstrate their use by analysing a demonstration code with intentionally implemented errors. These

ols.
ield
ased
ased

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Benchmarking static code analyzers

Jörg Herter[a,*], Daniel Kästner[a], Christoph Mallon[a], Reinhard Wilhelm[b]

[a] AbsInt GmbH, Science Park 1, Saarbrücken D-66123, Germany
[b] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

ARTICLE INFO

Keywords:
Static code analysis
Sound semantic analysis
C code
Safety-critical code
Benchmarking
Test case design
Abstract interpretation
Functional safety
Tool evaluation

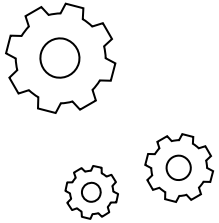
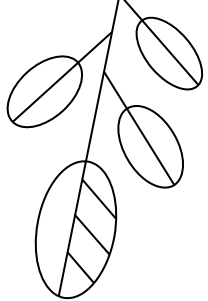
ABSTRACT

We show that a widely used benchmark set for the comparison of static"=analysis tools exhibits an impressive number of weaknesses, and that the internationally accepted quantitative"=evaluation metrics may lead to useless results. The weaknesses in the benchmark set were identified by applying a sound static analysis to the programs in this set and carefully interpreting the results. We propose how to deal with weaknesses of the quantitative metrics and how to improve such benchmarks and the evaluation process, in particular for *external* evaluations, in which an ideally neutral institution does the evaluation, whose results potential clients can trust. We also show that sufficiently high quality of the test cases makes an automatic result evaluation possible.

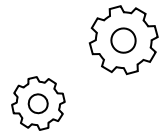
1. Introduction

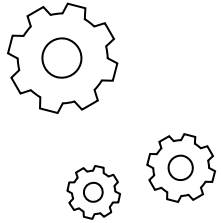
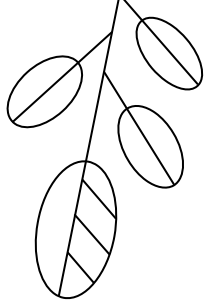
In the past, static analysis most of the time meant manual review of programs. Nowadays, automatic static"=analysis tools have gained popularity in software development as they offer a tremendous increase in productivity by automatically checking the code under a wide range of criteria. Many software development projects are developed according to coding guidelines, such as MISRA C, aiming at a programming style that improves clarity and reduces the risk of introducing bugs. For safety"=critical software projects complying to such coding best fitting tool is a challenging task. The first problem is that the term static analysis is used for a wide range of techniques that are conceptually very different. They all have in common that they compute their results just from the program code, without actually executing the program under analysis. They can be categorized in three main groups:

Syntax checkers They are limited to investigating the program syntax. Most of the algorithmically checkable MISRA C rules can

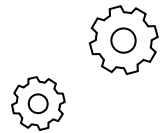
So we need a bunch of benchmarks, standards and test suites

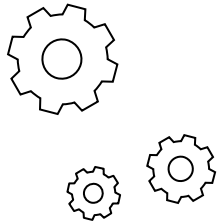
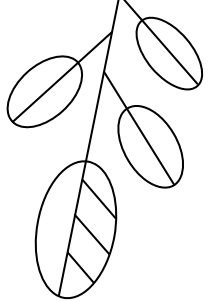



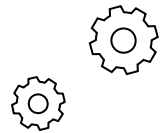
350,000+

This is how many functional tests are there in GCC. LLVM and Clang have mostly same huge test packages. Industrial compilers, like ICC, can have **1 mln+** tests

So we need some elastic CI platform that can easily make distributed testing in parallel and will be optimized for it

A BUNCH OF METRICS

Definition (classification context) [edit]

For classification tasks, the terms *true positives*, *true negatives*, *false positives*, and *false negatives* (see Type I and type II errors for definitions) compare the results of the classifier under test with trusted external judgments. The terms *positive* and *negative* refer to the classifier's prediction (sometimes known as the *expectation*), and the terms *true* and *false* refer to whether that prediction corresponds to the external judgment (sometimes known as the *observation*).

Let us define an experiment from P positive instances and N negative instances for some condition. The four outcomes can be formulated in a 2×2 contingency table or confusion matrix, as follows:

Sources: [5][6][7][8][9][10][11][12][13] view · talk · edit

		Predicted condition			
	Total population = P + N	**Positive (PP)**	**Negative (PN)**	Informedness, bookmaker informedness (BM) = TPR + TNR − 1	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
Actual condition	**Positive (P)**	**True positive (TP),** **hit**	**False negative (FN),** **type II error, miss, underestimation**	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate $= \frac{FN}{P} = 1 - TPR$
	Negative (N)	**False positive (FP),** **type I error, false alarm, overestimation**	**True negative (TN),** **correct rejection**	False positive rate (FPR), probability of false alarm, fall-out $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$
	Prevalence $= \frac{P}{P+N}$	Positive predictive value (PPV), precision $= \frac{TP}{PP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{PN} = 1 - NPV$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Negative likelihood ratio (LR−) $= \frac{FNR}{TNR}$
	Accuracy (ACC) = $\frac{TP + TN}{P + N}$	False discovery rate (FDR) $= \frac{FP}{PP} = 1 - PPV$	Negative predictive value (NPV) = $\frac{TN}{PN}$ $= 1 - FOR$	Markedness (MK), deltaP (Δp) = PPV + NPV − 1	Diagnostic odds ratio (DOR) = $\frac{LR+}{LR-}$
	Balanced accuracy (BA) $= \frac{TPR + TNR}{2}$	F_1 score $= \frac{2PPV \times TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$	Fowlkes–Mallows index (FM) $= \sqrt{PPV \times TPR}$	Matthews correlation coefficient (MCC) $= \sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$	Threat score (TS), critical success index (CSI), Jaccard index $= \frac{TP}{TP + FN + FP}$

Precision and recall are then defined as:[23]

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Terminology and derivations from a confusion matrix

condition positive (P)
the number of real positive cases in the data

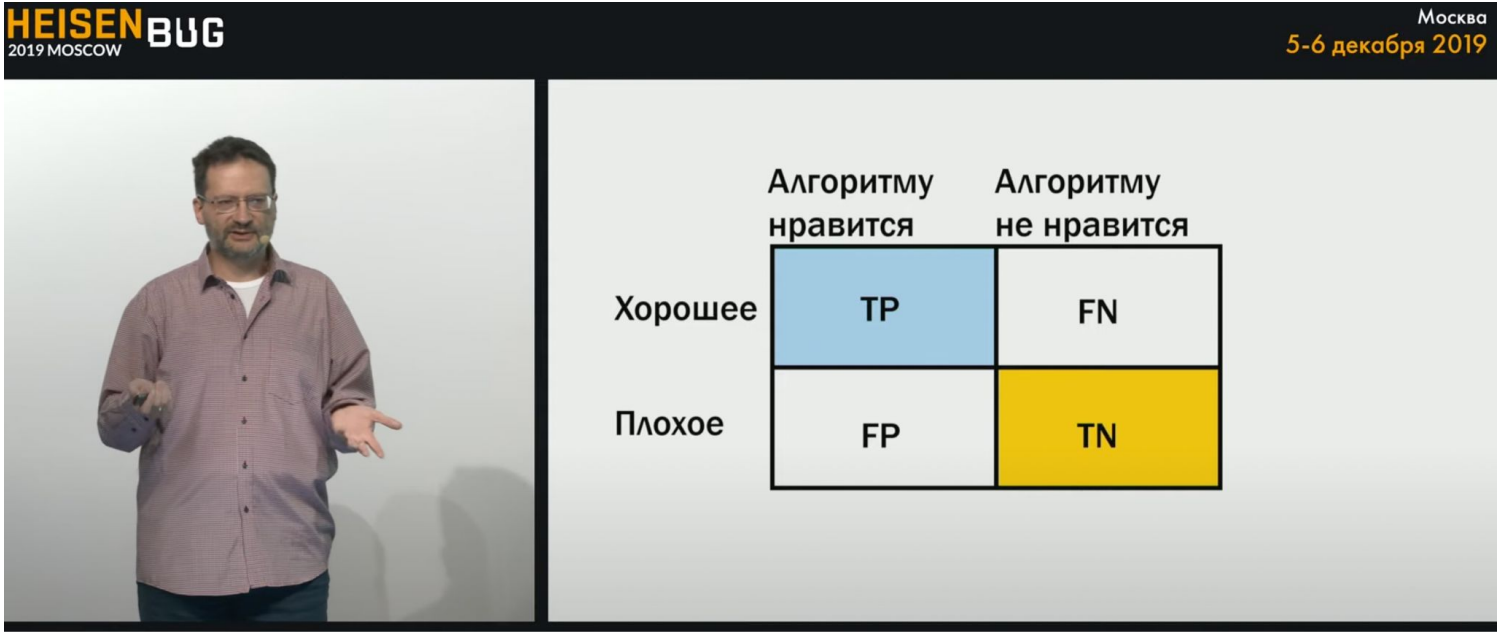
condition negative (N)
the number of real negative cases in the data

PRECISION AND RECALL

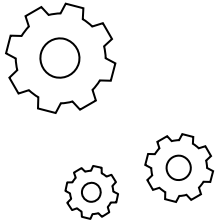
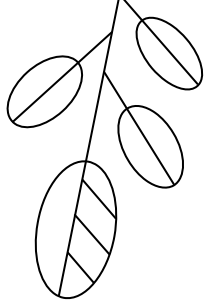
$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

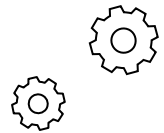
PRECISION AND RECALL



Roman Poborchiy, Heisenbug 2019

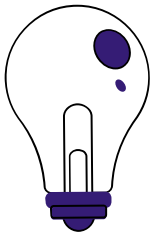



So we need proper dashboards for aggregation of results

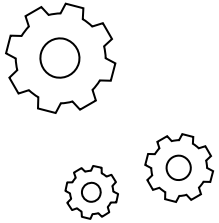
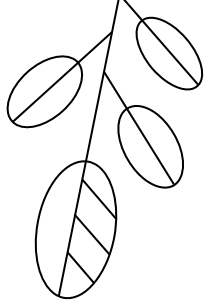
02 EXISTING SOLUTIONS

WHAT DO WE ALREADY HAVE

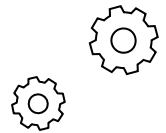


EXISTING SOLUTIONS

- Commercial tools and commercial test suites
- Custom scripts and test frameworks
- Millions of not structured tests
- Recreation of a same wheel again and again

BUT


Apes together strong.

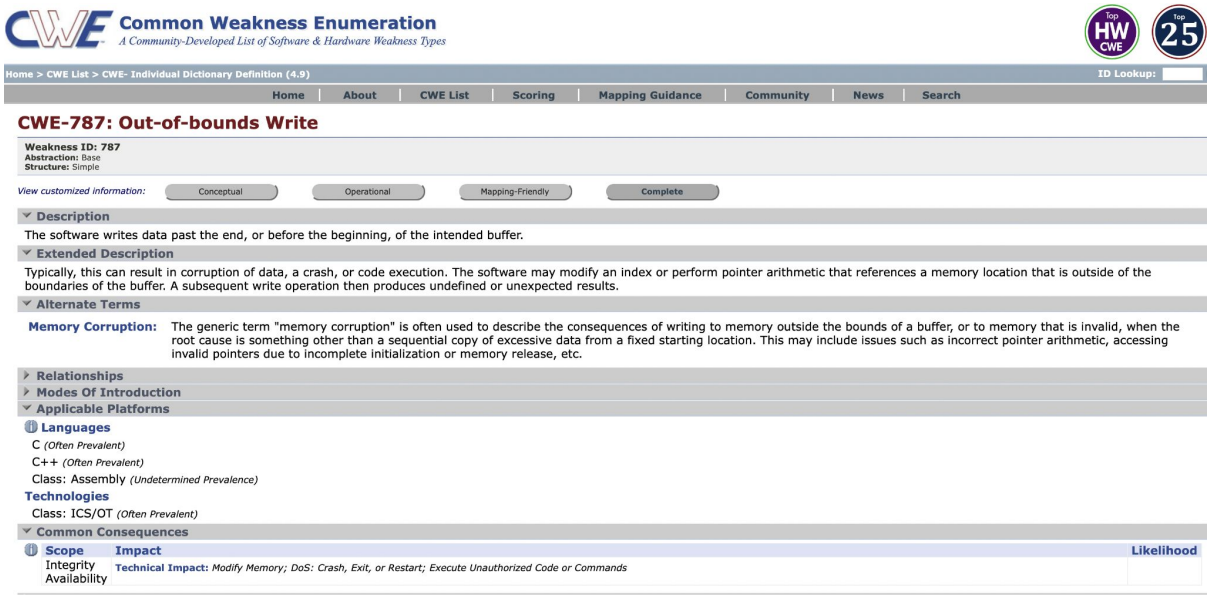
SO


NIST
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE
CWE
AUTOSAR
Enabling continuous innovations
spec
CERT
MISRA

CWE : MAIN LIST OF WEAKNESSES

- Common Weakness Enumeration (CWE™) is a community-developed list of common software and hardware weakness types

- **Weaknesses** that could result in systems, networks, or hardware being **vulnerable to attack**



Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types

Home > CWE List > CWE- Individual Dictionary Definition (4.9) ID Lookup:

Home | About | CWE List | Scoring | Mapping Guidance | Community | News | Search

CWE-787: Out-of-bounds Write

Weakness ID: 787
Abstraction: Base
Structure: Simple

View customized information: Conceptual | Operational | Mapping-Friendly | Complete

Description

The software writes data past the end, or before the beginning, of the intended buffer.

Extended Description

Typically, this can result in corruption of data, a crash, or code execution. The software may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent write operation then produces undefined or unexpected results.

Alternate Terms

Memory Corruption: The generic term "memory corruption" is often used to describe the consequences of writing to memory outside the bounds of a buffer, or to memory that is invalid, when the root cause is something other than a sequential copy of excessive data from a fixed starting location. This may include issues such as incorrect pointer arithmetic, accessing invalid pointers due to incomplete initialization or memory release, etc.

Relationships

Modes Of Introduction

Applicable Platforms

Languages

C *(Often Prevalent)*
C++ *(Often Prevalent)*
Class: Assembly *(Undetermined Prevalence)*

Technologies

Class: ICS/OT *(Often Prevalent)*

Common Consequences

Scope	Impact	Likelihood
Integrity Availability	**Technical Impact:** *Modify Memory; DoS: Crash, Exit, or Restart; Execute Unauthorized Code or Commands*	

Demonstrative Examples

Example 1

The following code attempts to save four different identification numbers into an array.

Example Language: **C**

```
int id_sequence[3];

/* Populate the id array. */

id_sequence[0] = 123;
id_sequence[1] = 234;
id_sequence[2] = 345;
id_sequence[3] = 456;
```

Since the array is only allocated to hold three elements, the valid indices are 0 to 2; so, the assignment to id_sequence[3] is out of bounds.

Example 2

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

Example Language: **C**

```
int returnChunkSize(void *) {

  /* if chunk info is valid, return the size of usable memory,

  * else, return -1 to indicate an error

  */
  ...
}
int main() {
  ...
  memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
  ...
}
```

AUTOMOTOR INDUSTRY AND MORE

MISRA + AUTOSAR™ Enabling continuous innovations

C/C++ benchmarks

Mandatory, *Required*, *Advisory* guidelines

More code-style inspections than anything else

Dir 4.4	Sections of code should not be "commented out"
Category	Advisory
Applies to	C90, C99

Amplification

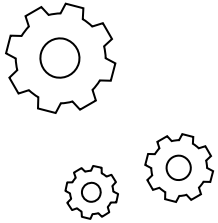
This rule applies to both `//` and `/* ... */` styles of comment.

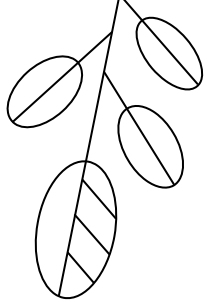
Rationale

Where it is required for sections of source code not to be compiled then this should be achieved by use of conditional compilation (e.g. *#if* or *#ifdef* constructs with a comment). Using start and end comment markers for this purpose is dangerous because C does not support nested comments, and any comments already existing in the section of code would change the effect.

See also

Rule 3.1, Rule 3.2





CERT SEI

Content Overview

Secure Coding

This site supports the development of coding standards for commonly used programming languages. The standards are developed through a community effort by members of the software development and software security communities.

CERT C Coding Standard

This standard provides rules for secure coding in the C programming language.

CERT C++ Coding Standard

This standard provides rules for secure coding in the C++ programming language.

CERT Java Coding Standard

The CERT Oracle Secure Coding Standard for Java provides rules for secure coding in the Java programming language. This coding standard affects the wide range of software systems developed in the Java programming language.

CERT Android Coding Standard

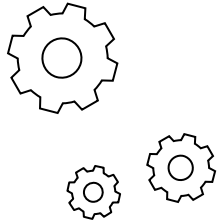
This standard provides rules for secure coding of applications (apps) for the Android platform.

CERT Perl Coding Standard

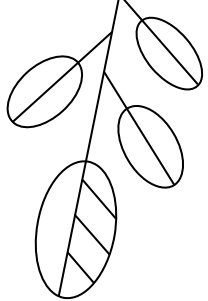
The CERT Perl Secure Coding Standard provides rules and recommendations for secure coding in the Perl programming language.

Agile Collaboration Group

The Agile Collaboration Group provides a forum for sharing experience and knowledge about applying Agile in larger programs. It can provide a continuing stream of unbiased guidance on using Agile methods and has become a resource for "lessons learned" about applying Agile to larger-scale projects.



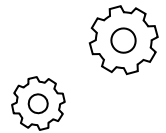
CERT SEI



SER00-J. Enable serialization compatibility during class evolution

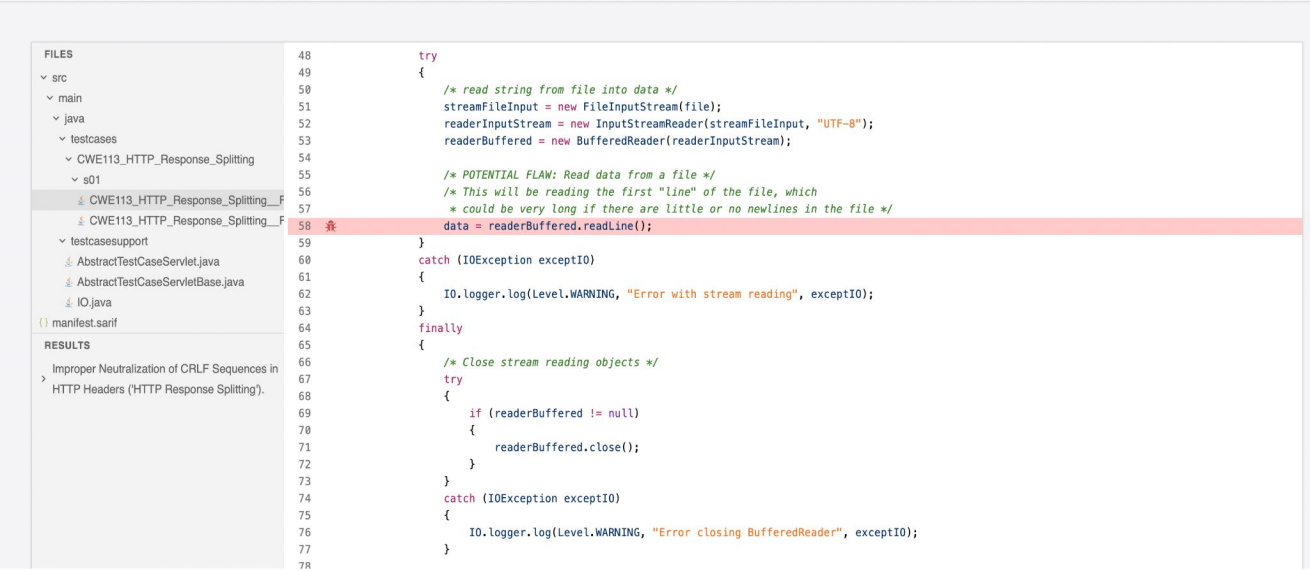
```
class GameWeapon implements Serializable {
   int numOfWeapons = 10;

   public String toString() {
       return
String.valueOf(numOfWeapons);
   }
}
```

```
class GameWeapon implements Serializable {
   private static final long serialVersionUID = 24L;

   int numOfWeapons = 10;

   public String toString() {
       return String.valueOf(numOfWeapons);
   }
}
```



NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

"To promote U.S. innovation and industrial competitiveness by advancing measurement science, standards, and technology in ways that enhance economic security and improve our quality of life."




```c
/* TEMPLATE GENERATED TESTCASE FILE
Filename: CWE690_NULL_Deref_From_Return__int64_t_malloc_21.c
Label Definition File: CWE690_NULL_Deref_From_Return.free.label.xml
Template File: source-sinks-21.tmpl.c
*/
/*
* @description
* CWE: 690 Unchecked Return Value To NULL Pointer
* BadSource: malloc Allocate data using malloc()
* Sinks:
*    GoodSink: Check to see if the data allocation failed and if not, use data
*    BadSink : Don't check for NULL and use data
* Flow Variant: 21 Control flow: Flow controlled by value of a static global variable. All functions contained
in one file.
*
*
*/

#include <wchar.h>

#ifndef OMITBAD

/* The static variable below is used to drive control flow in the sink function */
static int badStatic = 0;

static void badSink(int64_t * data)
{
   if(badStatic)
   {
      /* FLAW: Initialize memory buffer without checking to see if the memory allocation function failed */
      data[0] = 5LL;
      printLongLongLine(data[0]);
      free(data);
   }
}

. . .
```

STANDARD PERFORMANCE EVALUATION CORPORATION

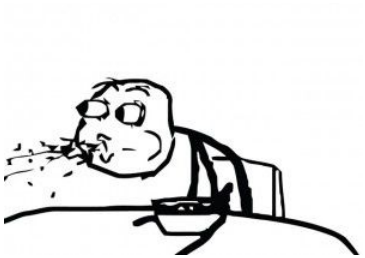
ДОКЛАД Performance 19.11 / 10:30 – 11:15 (МСК)

Tussle — новый формат бенчмаркинга реальных приложений

Зал 2 RU

Команда Azul Systems заопенсорсила новый фреймворк для измерения производительности реальных приложений — Tussle. В своем докладе Алексей опишет старые способы замеров и пояснит, почему throughput и latency больше не могут рассказать вам о производительности приложения. Он также объяснит, почему SPECjbb2015, на которой меряются основные производители железа, — большой обман и расскажет про суперсовременную методологию на основе Tussle.

#enterprise #benchmark



Alexey Ignatenko, Joker 2022

STANDARD PERFORMANCE EVALUATION CORPORATION

All SPEC JBB2015 Results Published by SPEC

These results have been submitted to SPEC; see the disclaimer before studying any results.

Search published SPECjbb2015results

Last update: *2022-09-20t12:36*

I Search I SPECjbb2015-Composite I SPECjbb2015-Distributed I SPECjbb2015-MultiJVM I

SPECjbb2015-Composite (185):

[Search in SPECjbb2015-Composite results]

Tester Name	System Name	JVM Name	JVM Version	max-jOPS	critical-jOPS
ASUSTeK Computer Inc.	RS700-E10-RS12U HTML	Oracle Java SE 16.0.1	Java HotSpot 64-bit Server VM, version 16.0.1	252973	215468
ASUSTeK Computer Inc.	RS700A-E11-RS12U HTML	Oracle Java SE 16.0.1	Java HotSpot 64-bit Server VM, version 16.0.1	338796	298117
ASUSTeK Computer Inc.	RS720-E10-RS12 HTML	Oracle Java SE 17	Java HotSpot 64-bit Server VM, version 17	255915	226936
ASUSTeK Computer Inc.	RS700A-E11-RS12U HTML	Oracle Java SE 17.0.1	Java HotSpot 64-bit Server VM, version 17.0.1	372676	329839
Cisco Systems	Cisco UCS C240 M5 HTML	Oracle Java SE 8u131	Java HotSpot 64-bit Server VM, version 1.8.0_131	155296	75071
Cisco Systems	Cisco UCS C480 M5 HTML	Oracle Java SE 8u131	Java HotSpot 64-bit Server VM, version 1.8.0_131	262190	97569
Cisco Systems	Cisco UCS C240 M5 HTML	Oracle Java SE 11.0.2	Java HotSpot 64-bit Server VM, version 11.0.2	162355	95906
Cisco Systems	Cisco UCS C245 M6 HTML	Oracle Java SE 16.0.1	Java HotSpot 64-Bit Server VM,version 16.0.1	193086	176283

SPEC

NOT ONLY ABOUT PERFORMANCE

401.bzip2
SPEC CPU2006 Benchmark Description

Benchmark Name

401.bzip2

Benchmark Author

Julian Seward <jseward [at] acm.org>

Benchmark Program General Category

Compression

Benchmark Description

401.bzip2 is based on Julian Seward's bzip2 version 1.0.3. The only difference between bzip2 1.0.3 and 401.bzip2 is that SPEC's version of bzip2 performs no file I/O other than reading the input. All compression and decompression happens entirely in memory. This is to help isolate the work done to only the CPU and memory subsystem.

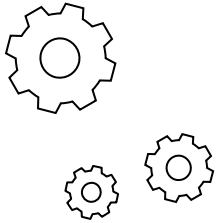
Input Description

401.bzip2's reference workload has six components: two small JPEG images, a program binary, some program source code in a tar file, an HTML file, and a "combined" file, which is representative of an archive that contains both highly compressible and not very compressible files.

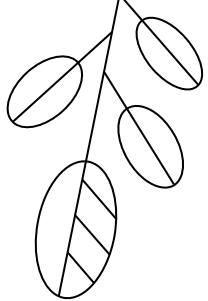
Each input set is compressed and decompressed at three different blocking factors ("compression levels"), with the end result of the process being compared to the original data after each decompression step.

WHAT ELSE?

lit – **L**LVM **I**ntegrated Tester

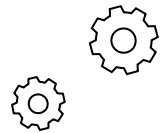


LIT



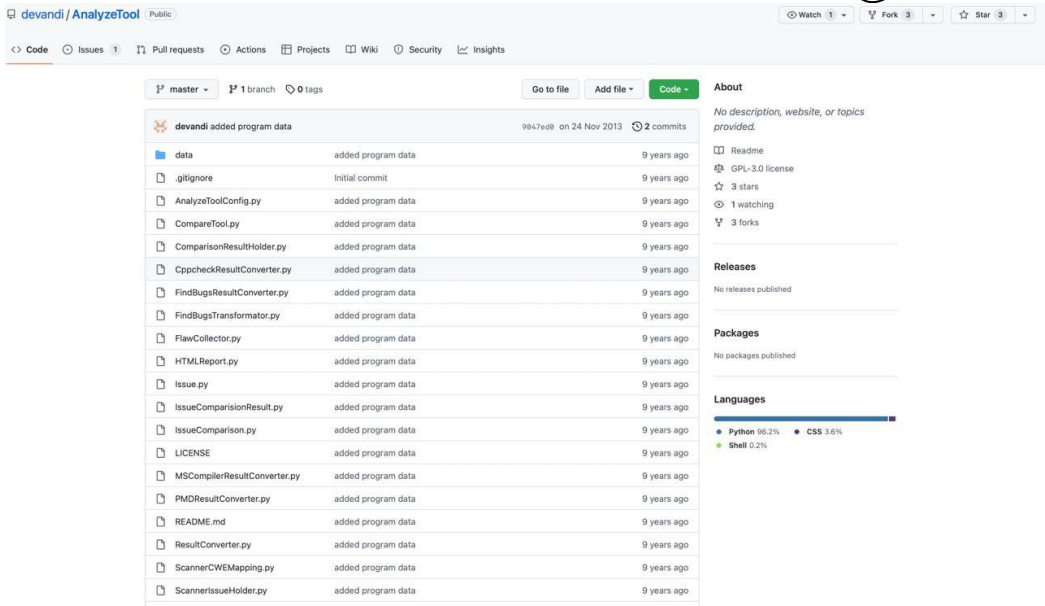
- Python scripts
- Inline syntax in test sources
- Only cli application

```
// RUN: %clang_cc1 -fsyntax-only -verify -std=c++11 %s
// RUN: %clang_cc1 -fsyntax-only -verify -std=c++1y %s
-DCXX1Y

struct NonLiteral { NonLiteral(); };

// A type is a literal type if it is:

// [C++1y] - void
constexpr void f() {}
#ifndef CXX1Y
// expected-error@-2 {{'void' is not a literal type}}
#endif
```



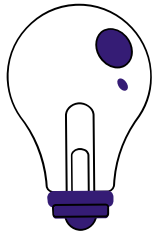
PROBLEMS OF EXISTING SOLUTIONS

- No universal frameworks
- No aggregation/structuralization of benchmarks
- No modern solutions
- No optimization and automatization
- We see a good scientific, but low class engineering work

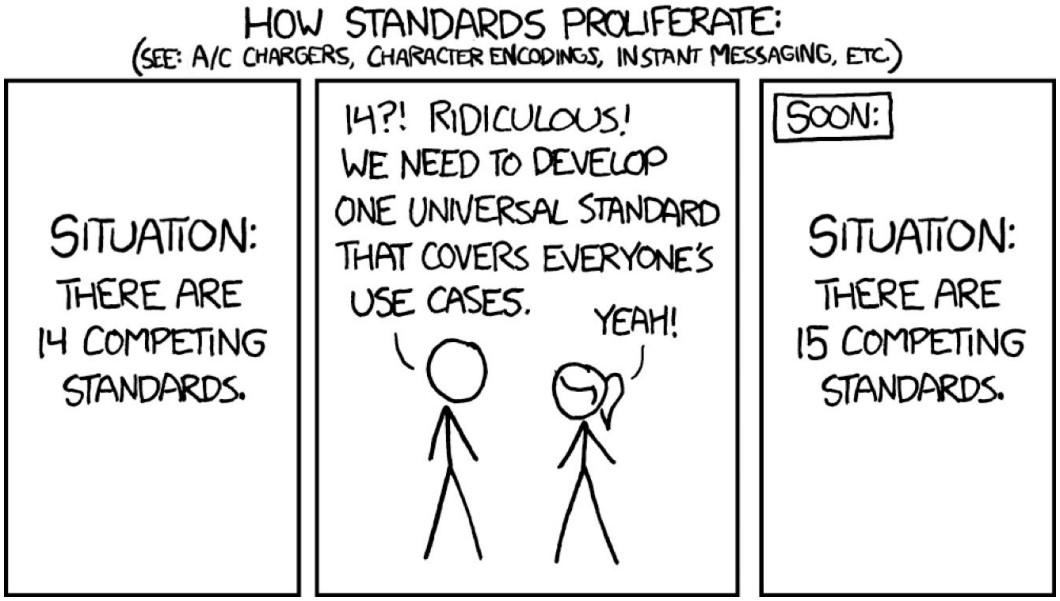


03 SAVE

WHAT DO WE SUGGEST



HOW DO WE SOLVE IT



HOW TO CLOSE THESE GAPS

Native CLI application that can process readable tests with different plugins

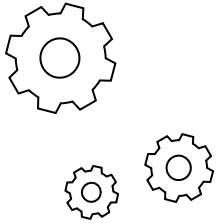
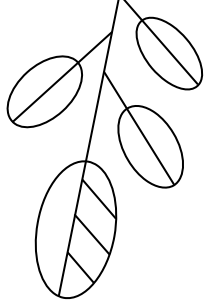
01

https://github.com/saveourtool/save

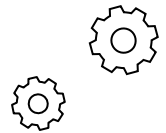
https://github.com/saveourtool/save-cloud

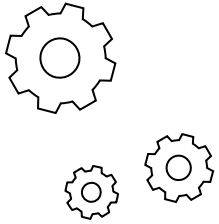
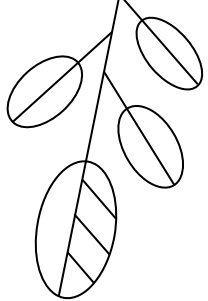
02

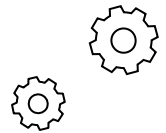
Some specific **CI/CD service**, that has standard benchmarks and executes tests on flexible cluster

Software **A**nalysis **V**erification & **E**valuation



SAVE-CLI

CONCEPT POINTS

What did we want (and still want) to achieve?

01

Native application

Kotlin Native Multiplatform
Win/Linux/MacOS

02

Plugins and reporters for test logic (processing)

Plugins and reporters should have a common interface SARIF

03

Configuration mechanism

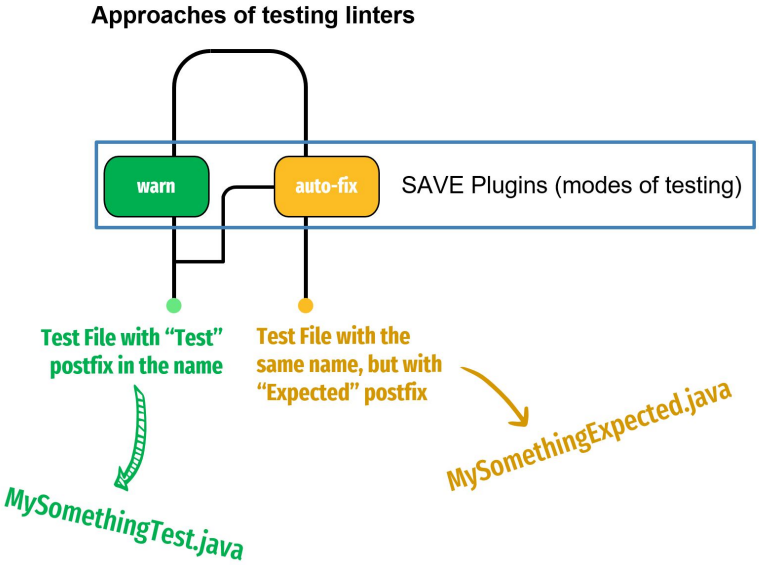
Hierarchical inheritance
Logic via config DSL (~~TOML~~)

DEFAULT PLUGINS

[FIX] plugin

1. Execute tested tool that should be tested on the test resource with Test suffix in the name

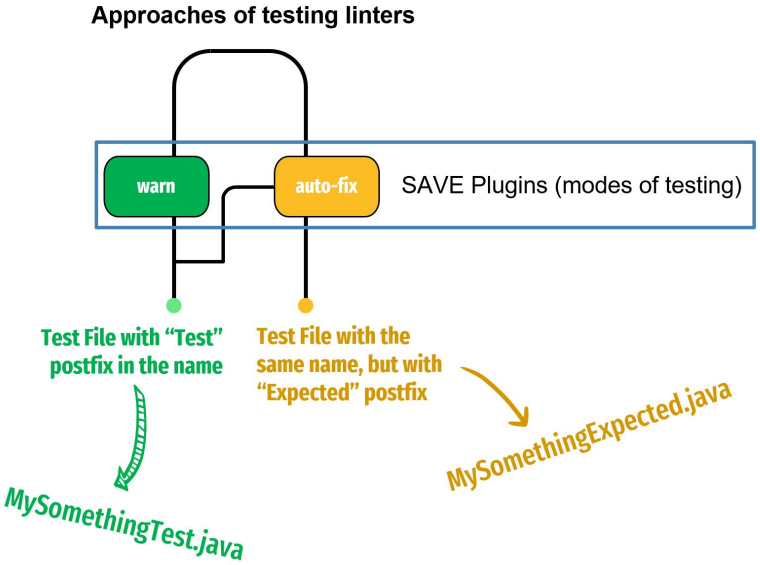
2. Compare the result with the resource with Expected suffix in the name



DEFAULT PLUGINS

[WARN] plugin

1. Execute tested tool that should be tested on the test resource with Test suffix in the name
2. Map and Compare the output with special metadata



CONFIGURATION

```
[general]
    tags = ["documentation", "custom tag", “other tags"]
    description = "Test for diktat - linter and formatter for Kotlin"
    suiteName = "warnings"
    execCmd = "java -jar ktlint -R diktat.jar"
    expectedWarningsPattern = "// ;warn:(.+):(\\d+): (.+)"

[warn]
    testNameSuffix = "Test.kt"
    actualWarningsPattern="(\\w+\\..+):(\\d+):(\\d+): (\\[.*\\].*)$"
    exactWarningsMatchHasColumn = true
    warningTextHasLine = true

[fix]
    execFlags = "-F"
```

Common section with the main configuration

Plugins (test execution logic). Optional

IN-FILE DSL

```
package com.saveourtool.diktat.test.resources.test.paragraph1.naming.enum_

// ;warn:3:1: [MISSING_KDOC_TOP_LEVEL] all public and internal top-level classes and functions should have
Kdoc{{.*}}
// ;warn:30: [WRONG_DECLARATIONS_ORDER] declarations of constants and enum members should be sorted{{.*}}
enum class EnumTestDetection {
   // ;warn:$line+1:5: [ENUM_VALUE] enum values should be in selected UPPER_CASE format: paSC_SAl_l{{.*}}
   paSC_SAl_l,

   PascAsl_f
   // ;warn:$line-2:5: [ENUMS_SEPARATED] {.*}} last enum entry must end with a comma

   // ;warn:1:9: {{.*}}[PACKAGE_NAME_INCORRECT_PREFIX] package name should start from company's domain:
com.saveourtool.save{{.*}}
}
```

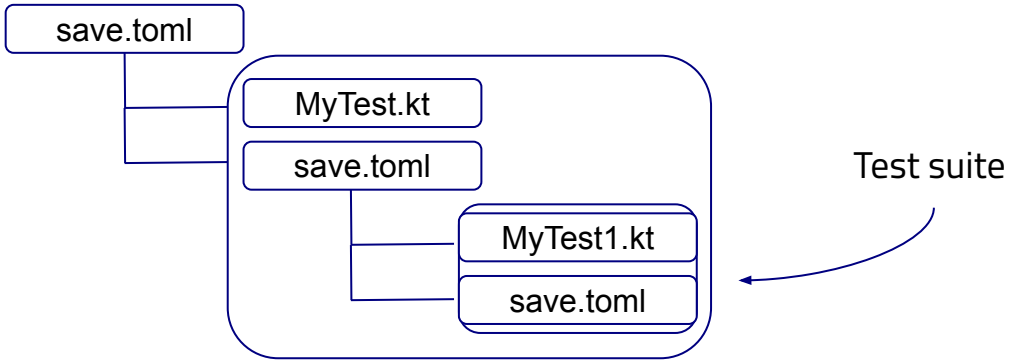
IN-FILE DSL

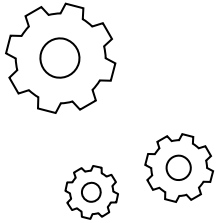
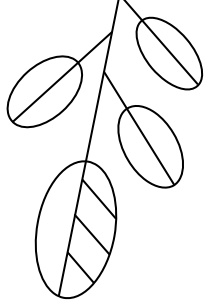
```
package com.saveourtool.diktat.test.resources.test.paragraph1.naming.enum_

// ;warn:3:1: [MISSING_KDOC_TOP_LEVEL] all public and internal top-level classes and functions should have
Kdoc{{.*}}
// ;warn:30: [WRONG_DECLARATIONS_ORDER] declarations of constants and enum members should be sorted{{.*}}
enum class EnumTestDetection {
   // ;warn:$line+1:5: [ENUM_VALUE] enum values should be in selected UPPER_CASE format: paSC_SAl_l{{.*}}
   paSC_SAl_l,

   PascAsl_f
   // ;warn:$line-2:5: [ENUMS_SEPARATED] {.*}} last enum entry must end with a comma

   // ;warn:1:9: {{.*}}[PACKAGE_NAME_INCORRECT_PREFIX] package name should start from company's domain:
com.saveourtool.save{{.*}}
}
```

IN-FILE DSL

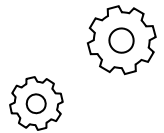
```
package com.saveourtool.diktat.test.resources.test.paragraph1.naming.enum_

// ;warn:3:1: [MISSING_KDOC_TOP_LEVEL] all public and internal top-level classes and functions should have
Kdoc{{.*}}
// ;warn:30: [WRONG_DECLARATIONS_ORDER] declarations of constants and enum members should be sorted{{.*}}
enum class EnumTestDetection {
   // ;warn:$line+1:5: [ENUM_VALUE] enum values should be in selected UPPER_CASE format: paSC_SAl_l{{.*}}
   paSC_SAl_l,

   PascAsl_f
   // ;warn:$line-2:5: [ENUMS_SEPARATED] {.*}} last enum entry must end with a comma

   // ;warn:1:9: {{.*}}[PACKAGE_NAME_INCORRECT_PREFIX] package name should start from company's domain:
com.saveourtool.save{{.*}}
}
```

CONFIGURATION INHERITANCE

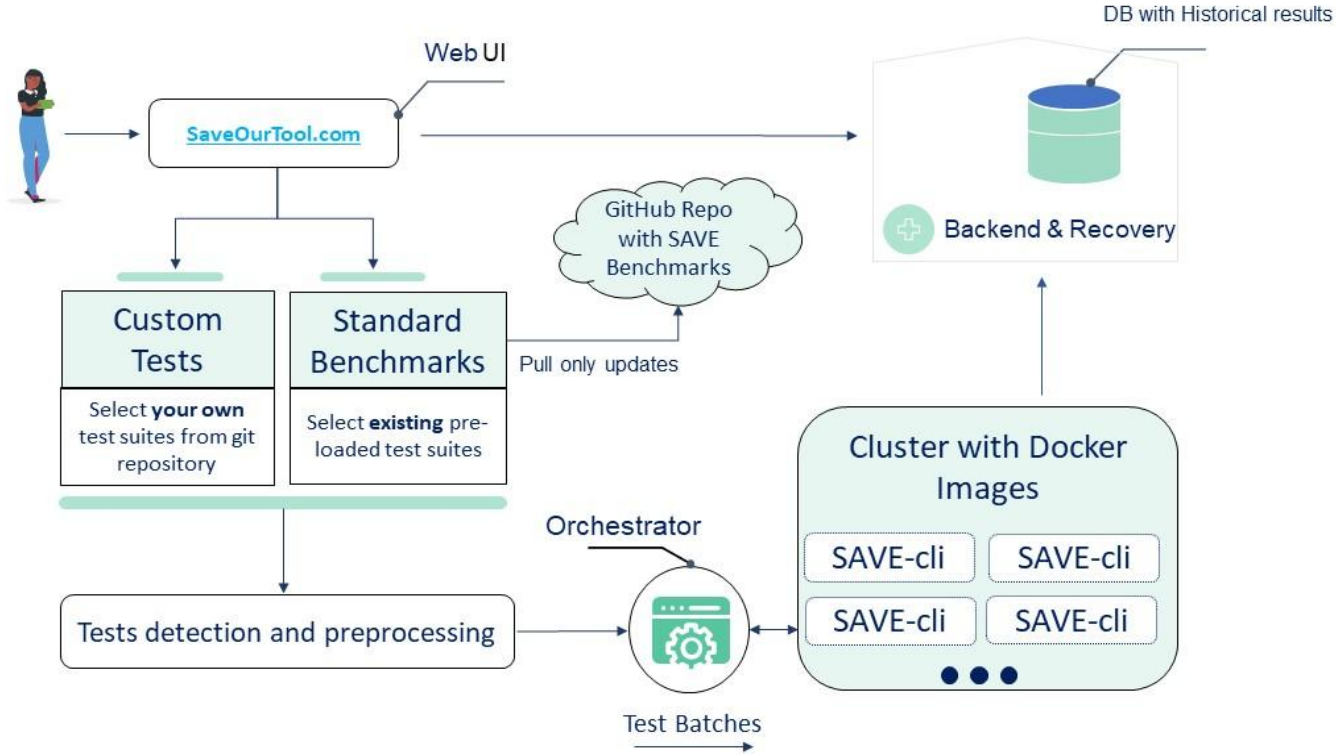


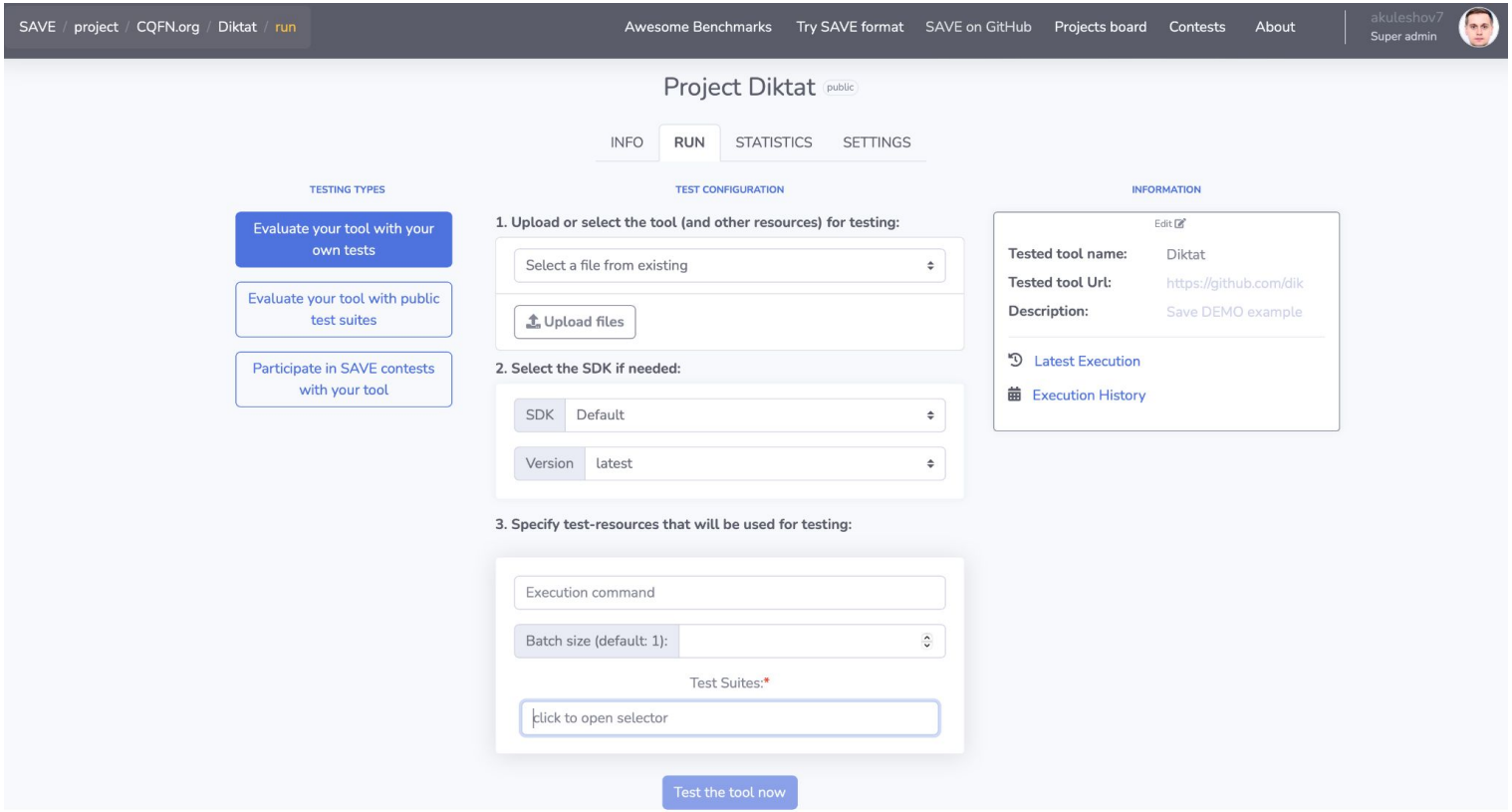



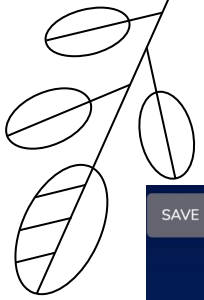
SAVE-CLOUD

Processing mechanism on the High-Level




SAVE / project / CQFN.org / Diktat / run
Awesome Benchmarks
Try SAVE format
SAVE on GitHub
Projects board
Contests
About
akuleshov7
Super admin
Project Diktat
public
INFO
RUN
STATISTICS
SETTINGS
TESTING TYPES
Evaluate your tool with your own tests
Evaluate your tool with public test suites
Participate in SAVE contests with your tool
TEST CONFIGURATION
1. Upload or select the tool (and other resources) for testing:
Select a file from existing
Upload files
2. Select the SDK if needed:
SDK
Default
Version
latest
3. Specify test-resources that will be used for testing:
Execution command
Batch size (default: 1):
Test Suites:*
click to open selector
Test the tool now
INFORMATION
Edit
Tested tool name:
Diktat
Tested tool Url:
https://github.com/dik
Description:
Save DEMO example
Latest Execution
Execution History
Copyright © SAVE 2021-2022
Version 0.4.0-alpha.0.81+7fdc3cb



SAVE / sandbox

Awesome Benchmarks | Try SAVE format | SAVE on GitHub | Projects board | Contests | About

akuleshov7
Super admin

Sandbox

try your SAVE configuration online

save.toml | setup.sh | test | toml | chrome

```
[general]
tags = ["demo"]
description = "saveourtool online demo"
suiteName = "Test"
execCmd="RUN_COMMAND"
language = "Kotlin"

[warn]
execFlags = "--build-upon-default-config -i"
actualWarningsPattern = "\\w+ - (\\d+)/(\\d+) - (.*)$" # (default value)
testNameRegex = ".*Test.*" # (default value)
patternForRegexInWarning = ["{{", "}}"]
# Extra flags will be extracted from a line that mathces this regex if it's present in a file
runConfigPattern = "# RUN: (.+)"
```

Upload files

← upload your tested tool and all other needed files

SDK | Default

Version | latest

SAVE / CQFN.org / Diktat / history / execution / 188

Awesome Benchmarks | SAVE format | SAVE on GitHub | Projects board | Contests | About

akuleshov7
Super admin

FINISHED
Project version:
264e5feb8f4c6410d70536d6fc4bdf090df62287

PASS RATE
83%

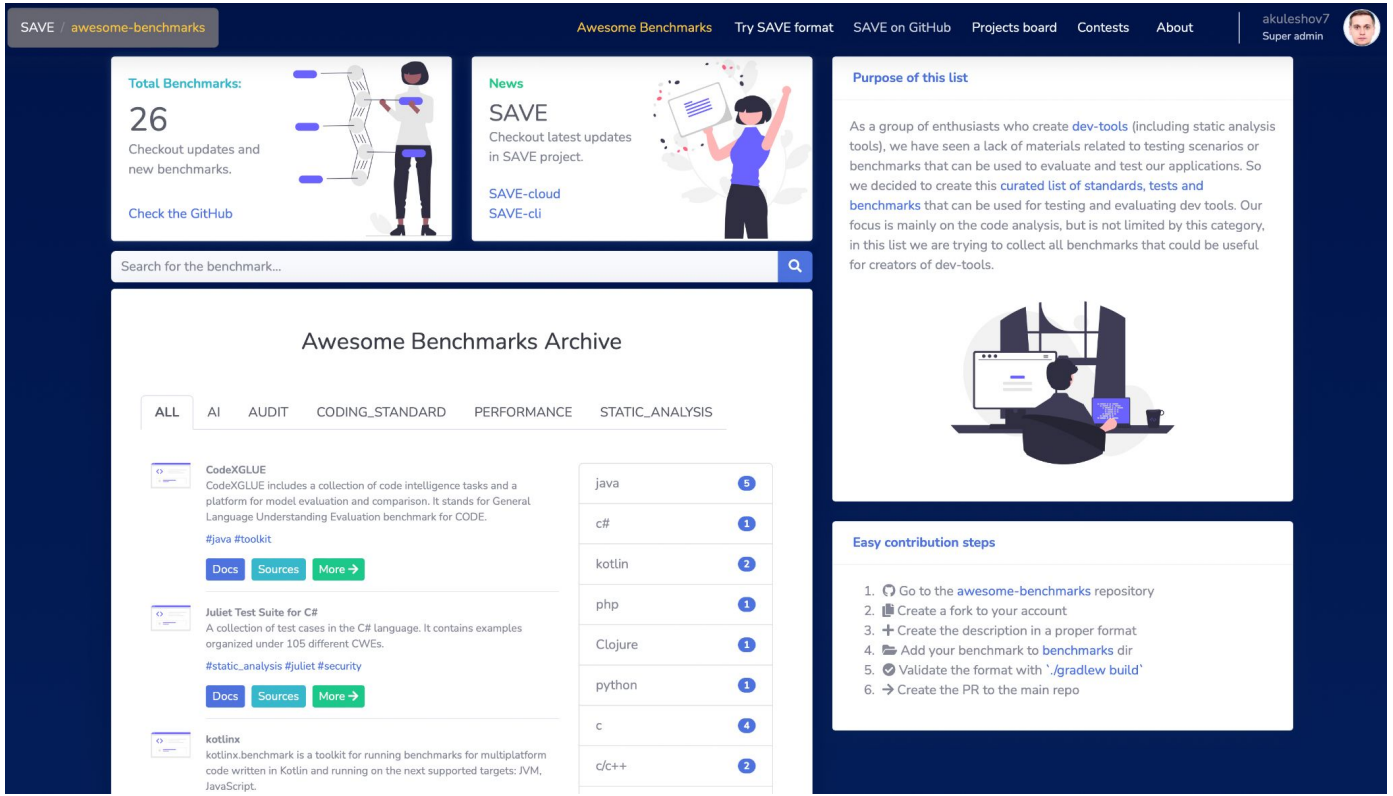
TESTS	RUNNING	FAILED	PASSED	PRECISION	RECALL
24	0	3	20	78	99

Rerun execution

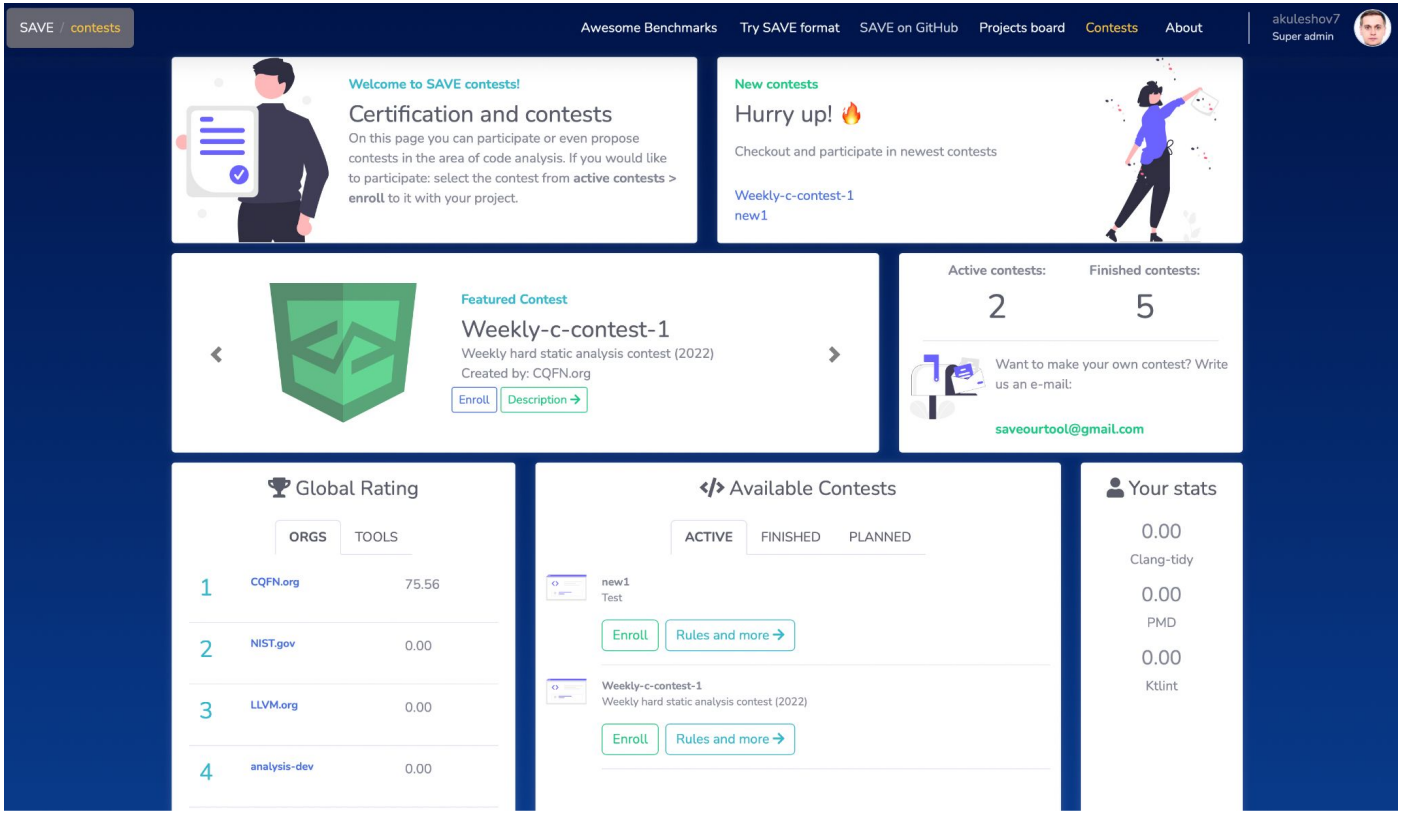
Status: ANY | File name: | Test suite: | Tags:

#	Start time	End time	Status	Missing	Matched	Test Name	Plugin type	Test suite	Tags	Agent ID
1	2022-08-08T15:31:14Z	2022-08-08T15:32:35Z	PASSED	0	0	fix/smoke/src/m ... Example1Test.kt	FixPlugin	Autofix: Smoke Tests	[fix, smoke]	d1d2e2f8bc18
Executed command	java -jar ktlint -R diktat.jar -F /tmp/FixPlugin--997674868/fix/smoke/src/main/kotlin/org/cqfn/save/Example1Test.kt									
Reason (additional info)	Completed successfully without additional information									
2	2022-08-08T15:31:14Z	2022-08-08T15:32:35Z	PASSED	0	0	fix/smoke/src/m ... nes/Bug1Test.kt	FixPlugin	Autofix: Smoke Tests	[fix, smoke]	d1d2e2f8bc18
3	2022-08-08T15:31:14Z	2022-08-08T15:32:35Z	PASSED	0	0	fix/smoke/src/m ... pty/Bug1Test.kt	FixPlugin	Autofix: Smoke Tests	[fix, smoke]	d1d2e2f8bc18
4	2022-08-08T15:31:14Z	2022-08-08T15:32:35Z	PASSED	0	0	fix/smoke/src/m ... nes/Bug1Test.kt	FixPlugin	Autofix: Smoke Tests	[fix, smoke]	d1d2e2f8bc18
5	2022-08-08T15:31:14Z	2022-08-08T15:32:35Z	PASSED	0	0	fix/smoke/src/m ... er1/Bug1Test.kt	FixPlugin	Autofix: Smoke Tests	[fix, smoke]	d1d2e2f8bc18
6	2022-08-08T15:31:14Z	2022-08-08T15:32:35Z	PASSED	0	7	fix_and_warn/sm ... er1/Bug1Test.kt	FixAndWarnPlugin	Autofix and Warn	[fix and warn]	d1d2e2f8bc18
7	2022-08-08T15:31:14Z	2022-08-08T15:32:35Z	PASSED	0	4	fix_and_warn/sm ... er2/Bug2Test.kt	FixAndWarnPlugin	Autofix and Warn	[fix and warn]	d1d2e2f8bc18
8	2022-08-08T15:31:14Z	2022-08-08T15:32:35Z	PASSED	0	4	fix_and_warn/sm ... Example1Test.kt	FixAndWarnPlugin	Autofix and Warn	[fix and warn]	d1d2e2f8bc18
9	2022-08-08T15:31:14Z	2022-08-08T15:32:35Z	PASSED	0	882	warn/chapter1/E ... nakeCaseTest.kt	WarnPlugin	Only Warnings: General	[warn, tag1]	d1d2e2f8bc18
10	2022-08-08T15:31:14Z	2022-08-08T15:32:35Z	IGNORED	0	0	warn/chapter1/GarbageTest.kt	WarnPlugin	Only Warnings: General	[warn, tag1]	d1d2e2f8bc18

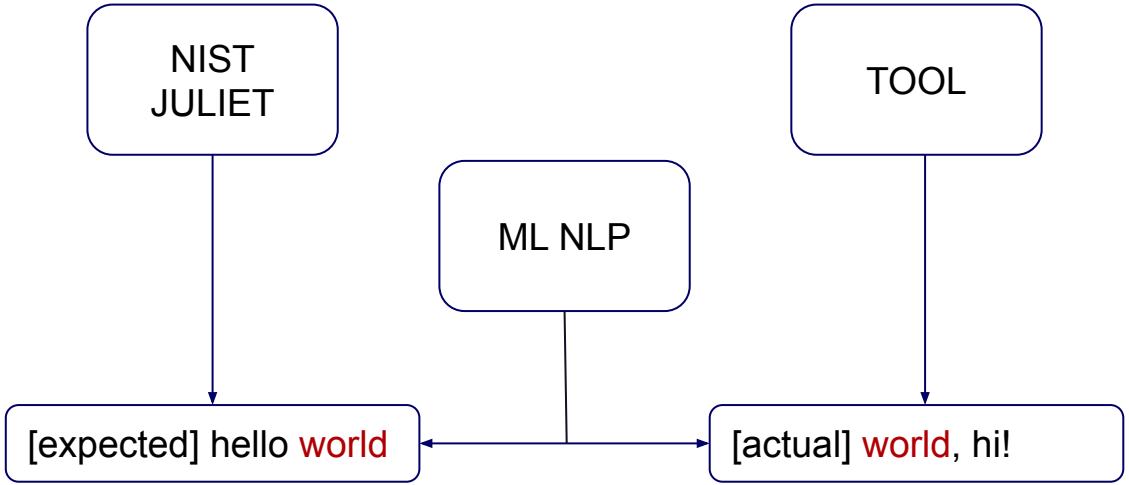
+ HOW WE WANT TO MAKE THE WORLD A BETTER PLACE

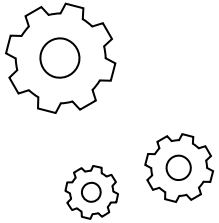


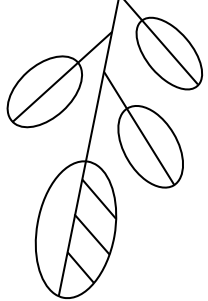
https://github.com/saveourtool/awesome-benchmarks


SAVE / contests
Awesome Benchmarks
Try SAVE format
SAVE on GitHub
Projects board
Contests
About
akuleshov7
Super admin
Welcome to SAVE contests!
Certification and contests
On this page you can participate or even propose contests in the area of code analysis. If you would like to participate: select the contest from active contests > enroll to it with your project.
New contests
Hurry up!
Checkout and participate in newest contests
Weekly-c-contest-1
new1
Featured Contest
Weekly-c-contest-1
Weekly hard static analysis contest (2022)
Created by: CQFN.org
Enroll
Description
Active contests:
2
Finished contests:
5
Want to make your own contest? Write us an e-mail:
saveourtool@gmail.com
Global Rating
ORGS
TOOLS
1 CQFN.org 75.56
2 NIST.gov 0.00
3 LLVM.org 0.00
4 analysis-dev 0.00
Available Contests
ACTIVE
FINISHED
PLANNED
new1
Test
Enroll
Rules and more
Weekly-c-contest-1
Weekly hard static analysis contest (2022)
Enroll
Rules and more
Your stats
0.00
Clang-tidy
0.00
PMD
0.00
Ktlint

AS A CONCLUSION:
SOMETHING ELSE INTERESTING?
PLANS?







AS A CONCLUSION:
WHEN OUR APPROACH CAN BE USED?

STATIC ANALYZERS AND AUTO-FIXERS

Checking warnings in the code

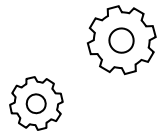
Checking auto-fixed code

COMPILERS AND THEIR PARTS

Testing generated IR and generated final code

Expected behaviour of compiled program

Warnings and errors in the front-end (parser)



THANK YOU FOR LISTENING!

https://github.com/**akuleshov7**
https://github.com/**saveourtool**