Automated Diagnosis of Software Configuration Errors

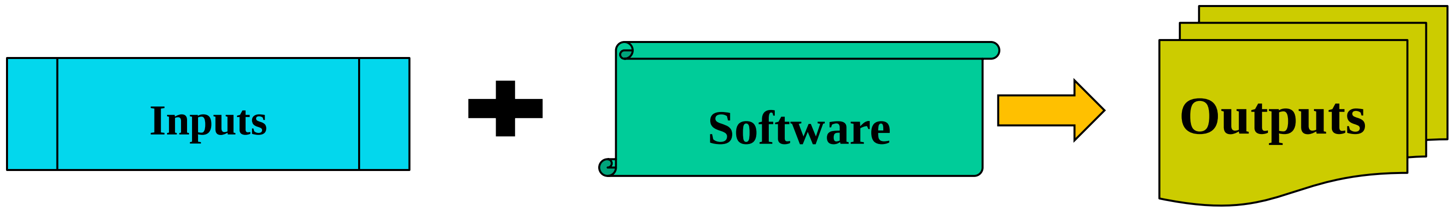
Sai Zhang, Michael D. Ernst

University of Washington

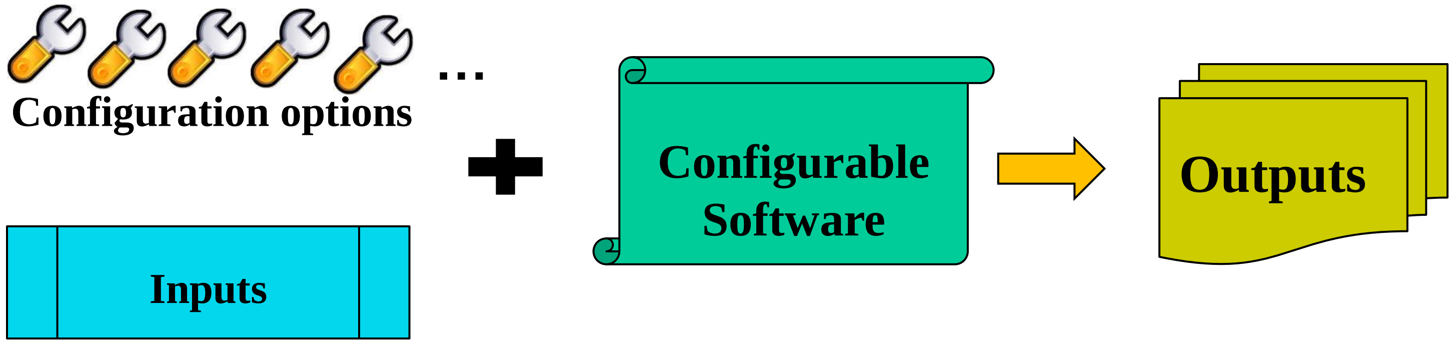
Computer Science & Engineering

UNIVERSITY *of* WASHINGTON

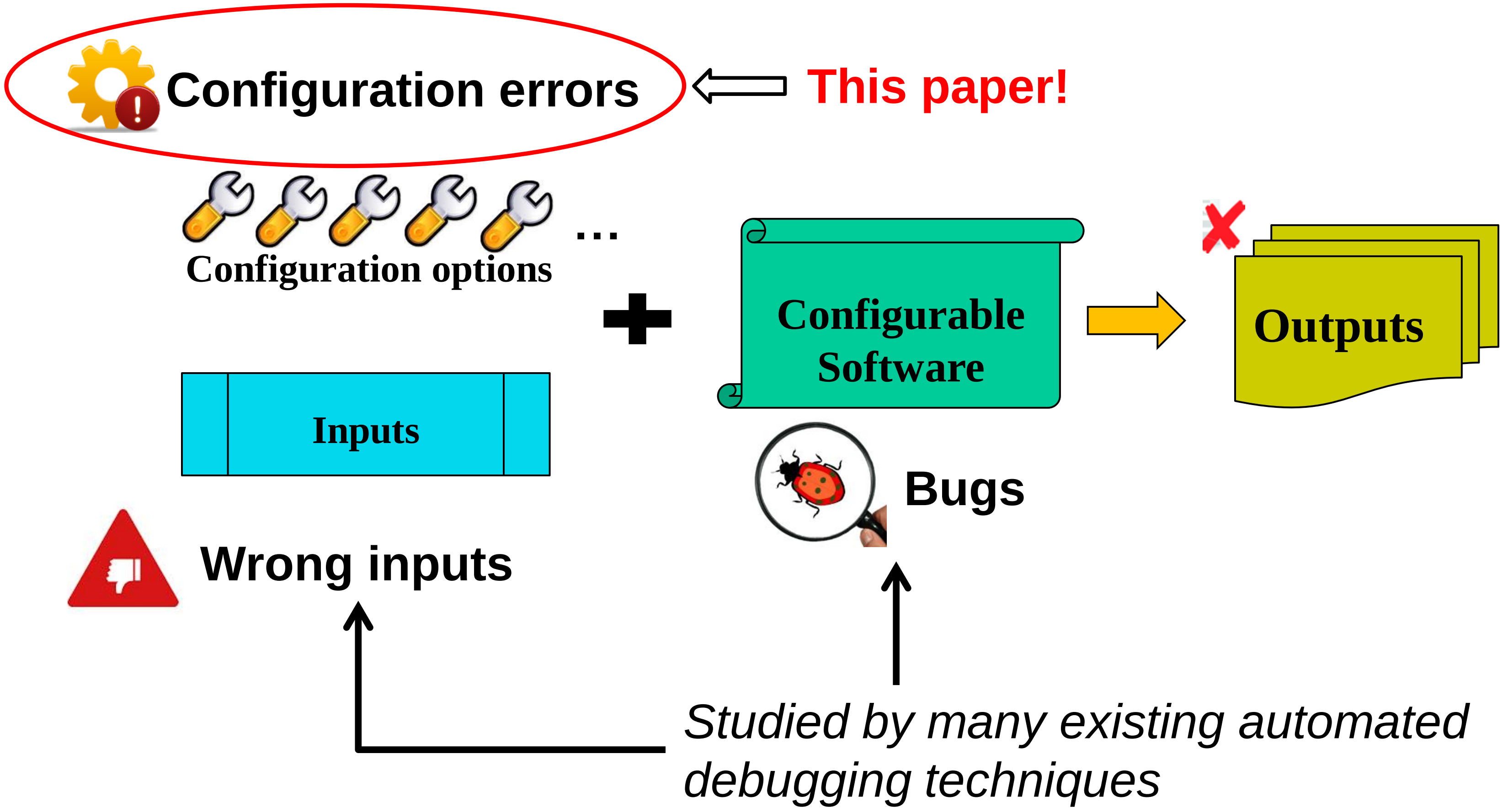
A typical software workflow



Modern software is often configurable



Possible root causes of wrong output



Why configuration errors?

- Fixable by *changing configuration options*
- Actionable by system administrators or end-users

- 17% of the total technical support cost [Kapoor '03, Yin '11]

- Configuration options ***vs.*** Inputs
  - Options: customize program behaviors by altering the control flow
  - Input values: produce output for a specific task

Outline

- ➡ Example
- The ConfDiagnoser Technique
- Evaluation
- Related Work
- Contributions

An example configuration error

- A "bug report" against the Randoop test generation tool

  … *Randoop fails to generate tests for NanoXML using the following command:* **`java randoop.main.Main NanoXML ...`**

  . . . ,but Randoop works perfectly well on its own examples, such as BinaryTree, TreeMap, etc.

Difficulty in diagnosing the Randoop error

- A silent failure
  - No crashing points
  - No stacktrace
  - No error message
- Inputs are already minimized

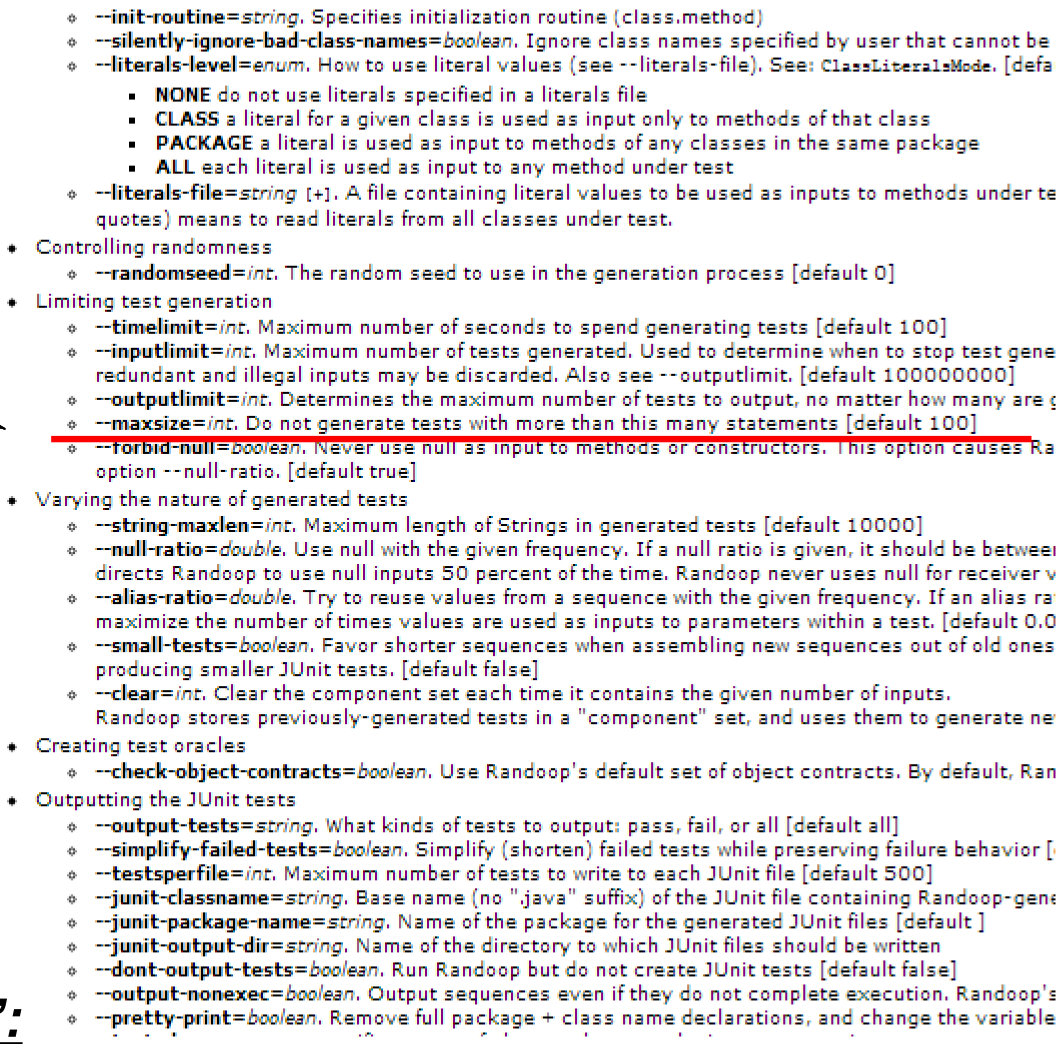
Delta debugging [Zeller'02], dynamic slicing [Zhang'06], capture/replay [Whitaker'01], stack trace analysis [Rakbin'11], tainting [Attariyan'12] …

Inapplicable

Root cause of the Randoop configuration error

57 Randoop options in total

...
maxsize = 100
...



- --**init-routine**=*string*. Specifies initialization routine (class.method)
- --**silently-ignore-bad-class-names**=*boolean*. Ignore class names specified by user that cannot be
- --**literals-level**=*enum*. How to use literal values (see --literals-file). See: ClassLiteralsMode. [defa
  - **NONE** do not use literals specified in a literals file
  - **CLASS** a literal for a given class is used as input only to methods of that class
  - **PACKAGE** a literal is used as input to methods of any classes in the same package
  - **ALL** each literal is used as input to any method under test
- --**literals-file**=*string* [+]. A file containing literal values to be used as inputs to methods under te quotes) means to read literals from all classes under test.

- Controlling randomness
  - --**randomseed**=*int*. The random seed to use in the generation process [default 0]
- Limiting test generation
  - --**timelimit**=*int*. Maximum number of seconds to spend generating tests [default 100]
  - --**inputlimit**=*int*. Maximum number of tests generated. Used to determine when to stop test gene redundant and illegal inputs may be discarded. Also see --outputlimit. [default 100000000]
  - --**outputlimit**=*int*. Determines the maximum number of tests to output, no matter how many are
  - --**maxsize**=*int*. Do not generate tests with more than this many statements [default 100]
  - --**forbid-null**=*boolean*. Never use null as input to methods or constructors. This option causes Ra option --null-ratio. [default true]
- Varying the nature of generated tests
  - --**string-maxlen**=*int*. Maximum length of Strings in generated tests [default 10000]
  - --**null-ratio**=*double*. Use null with the given frequency. If a null ratio is given, it should be betwee directs Randoop to use null inputs 50 percent of the time. Randoop never uses null for receiver v
  - --**alias-ratio**=*double*. Try to reuse values from a sequence with the given frequency. If an alias ra maximize the number of times values are used as inputs to parameters within a test. [default 0.0
  - --**small-tests**=*boolean*. Favor shorter sequences when assembling new sequences out of old ones producing smaller JUnit tests. [default false]
  - --**clear**=*int*. Clear the component set each time it contains the given number of inputs. Randoop stores previously-generated tests in a "component" set, and uses them to generate ne
- Creating test oracles
  - --**check-object-contracts**=*boolean*. Use Randoop's default set of object contracts. By default, Ran
- Outputting the JUnit tests
  - --**output-tests**=*string*. What kinds of tests to output: pass, fail, or all [default all]
  - --**simplify-failed-tests**=*boolean*. Simplify (shorten) failed tests while preserving failure behavior [
  - --**testsperfile**=*int*. Maximum number of tests to write to each JUnit file [default 500]
  - --**junit-classname**=*string*. Base name (no ".java" suffix) of the JUnit file containing Randoop-gene
  - --**junit-package-name**=*string*. Name of the package for the generated JUnit files [default]
  - --**junit-output-dir**=*string*. Name of the directory to which JUnit files should be written
  - --**dont-output-tests**=*boolean*. Run Randoop but do not create JUnit tests [default false]
  - --**output-nonexec**=*boolean*. Output sequences even if they do not complete execution. Randoop's
  - --**pretty-print**=*boolean*. Remove full package + class name declarations, and change the variable

Randoop code:

```
...
Sequence seq = createNewSeq();
if (seq.size() >  maxsize) {
     return null;
}
...
```

Resolve the reported ``bug'':

```
java randoop.main.Main --maxsize=1000 NanoXML
```

ConfDiagnoser's diagnosis report

- A ranked list of suspicious configuration options
- The top-ranked option for the Randoop error:

```
Suspicious configuration option: maxsize
```

Option name

```
It affects the behavior of predicate:
"newSequence.size() > GenInputsAbstract.maxsize"
(line 312, class: randoop.ForwardGenerator)
```

```
This predicate evaluates to true:
  3.3% of the time in normal runs
  32.5% of the time in the undesired run
```

Explanation

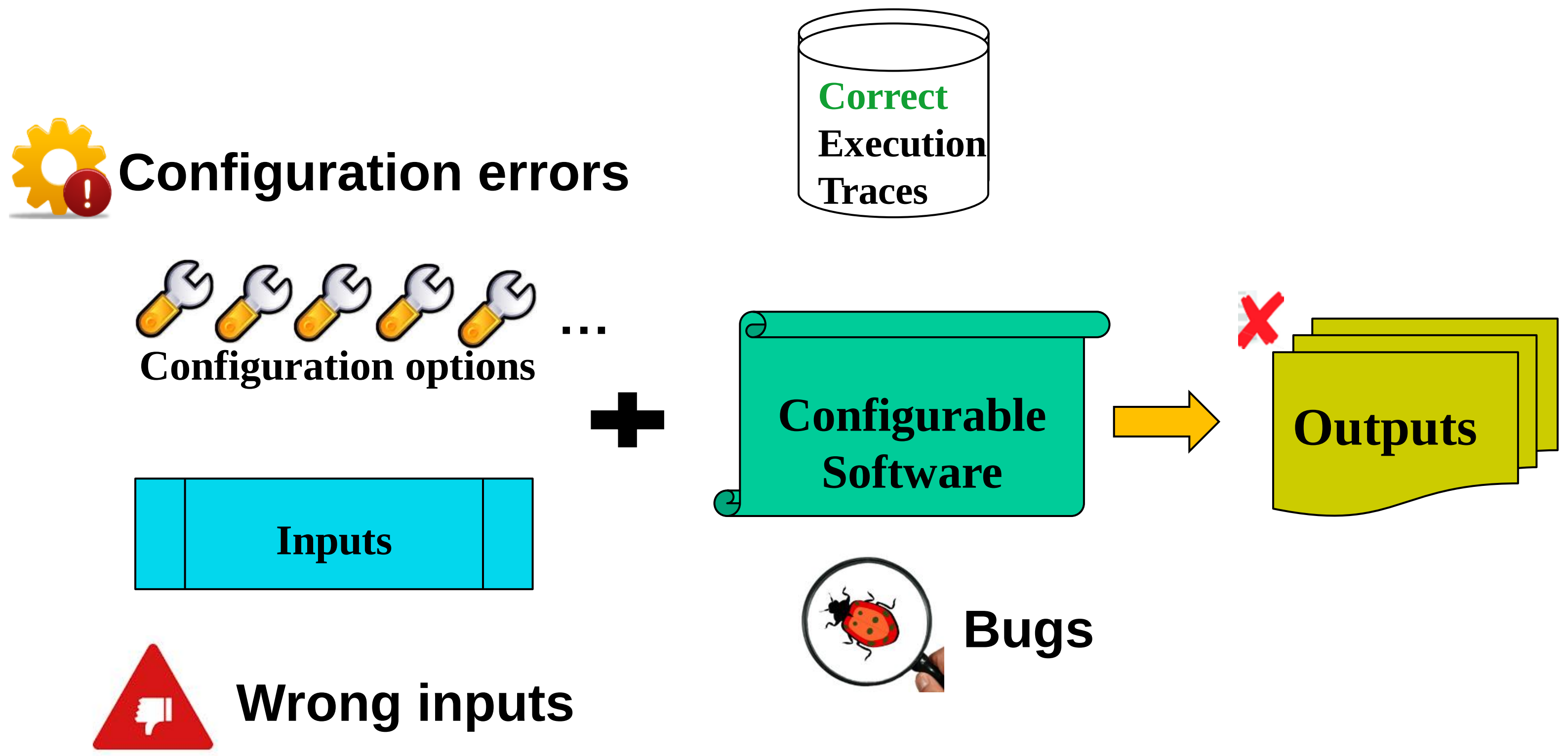
Outline

- Example
- The ConfDiagnoser Technique
- Evaluation
- Related Work
- Contributions

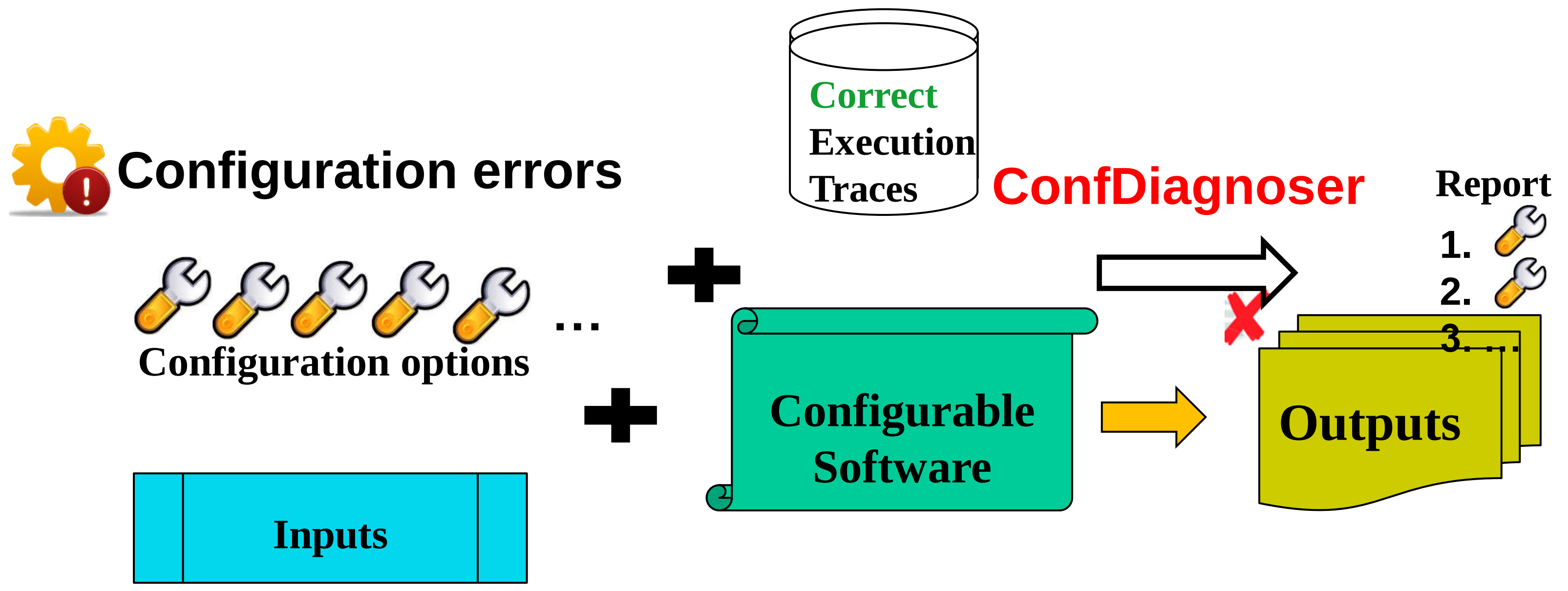
Outline

- Example
- ➡ The ConfDiagnoser Technique
- Evaluation
- Related Work
- Contributions

ConfDiagnoser's assumptions



ConfDiagnoser's assumptions



ConfDiagnoser's advantages

- Fully-automatically diagnoses configuration errors
- Diagnoses both crashing and non-crashing errors
- Requires no OS-level support

ConfDiagnoser's insight

- Control flow propagates most configuration options' effects
- Correct execution traces serve as approximate oracles
  - The control flow difference provides debugging clues

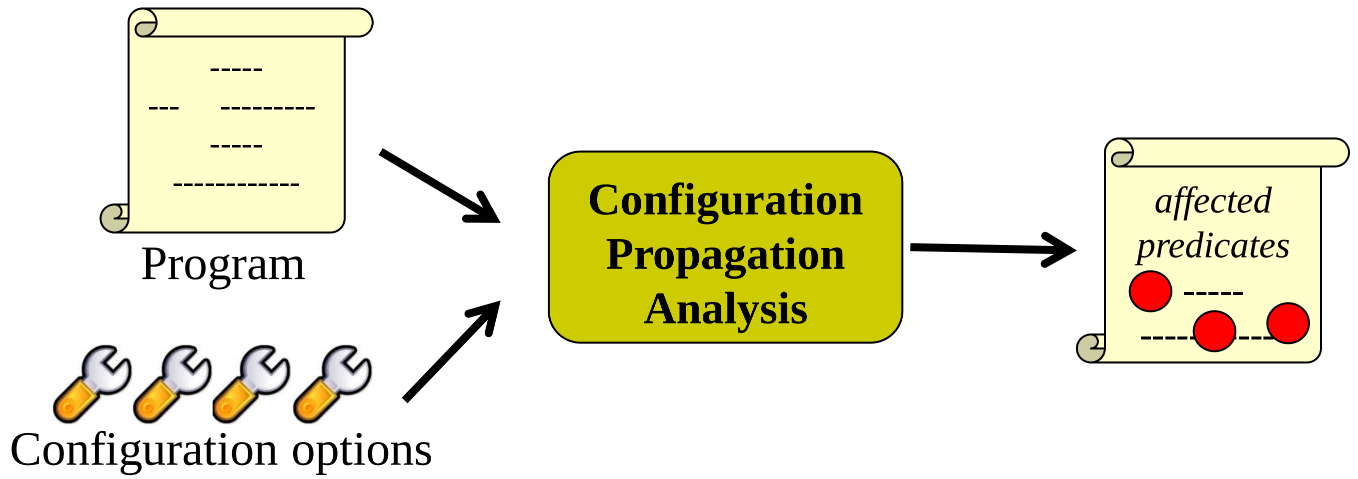
```
//a configuration option
int maxsize = readFromCommandLine();
...
Sequence seq = createNewSeq();
if (seq.size() >  maxsize) {
    return null;
}
```

This predicate evaluates to true:

3.3% of the time in **correct** runs

32.5% of the time in the **bad** runs

The ConfDiagnoser technique

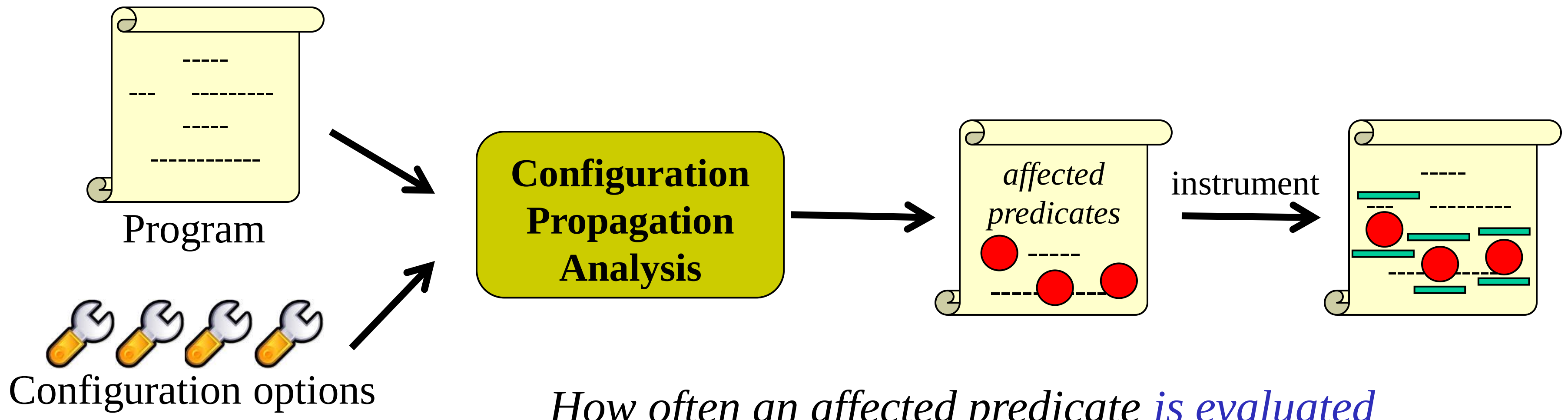


Compute a forward thin slice [Sridharan'07]

```
//a configuration option
int maxsize = readFromCommandLine();
Sequence seq = createNewSequence();
...
if (seq.size() >  maxsize) {
    return null;
}
...
```

affected predicate

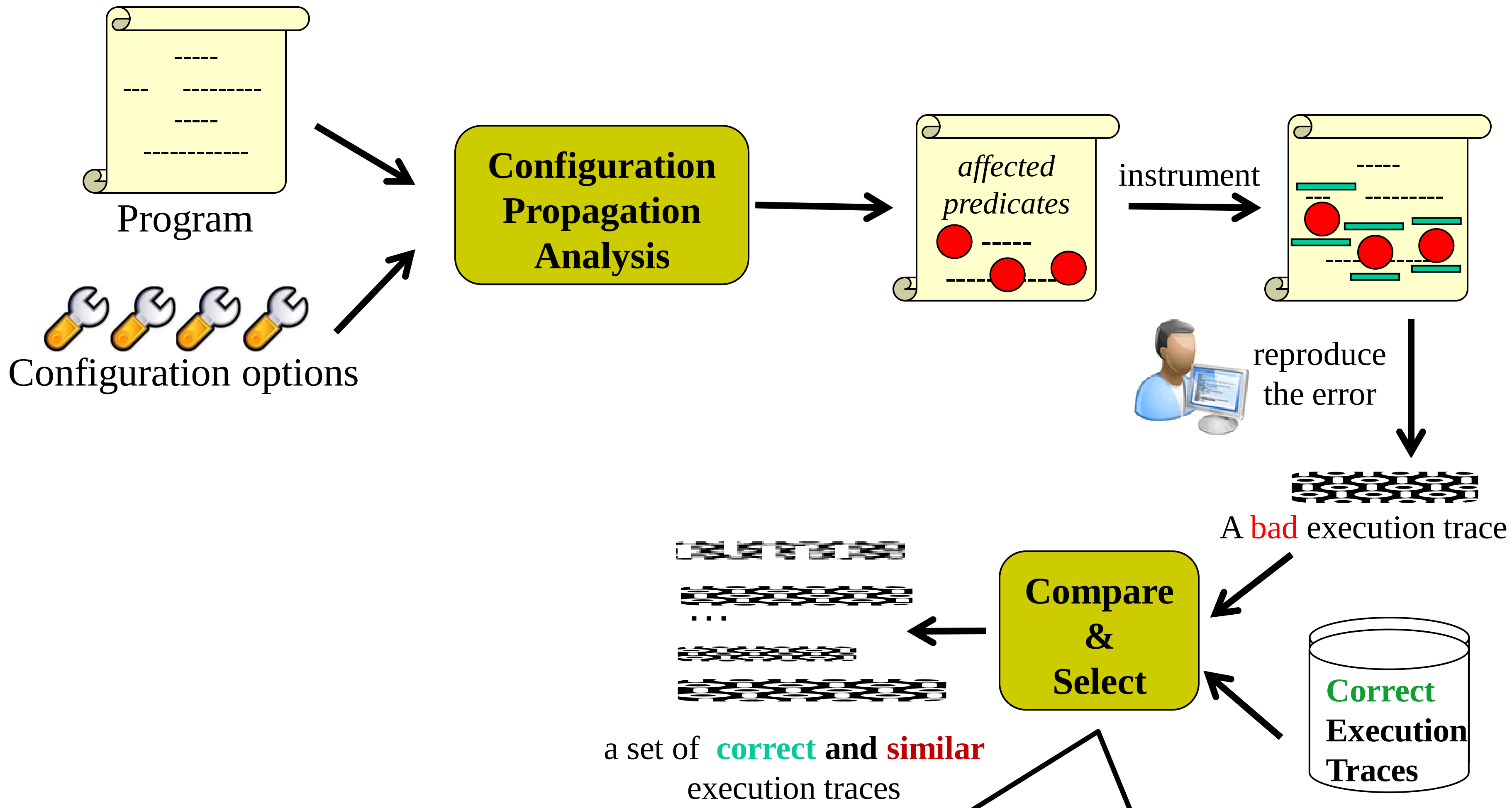
The ConfDiagnoser technique



How often an affected predicate is evaluated

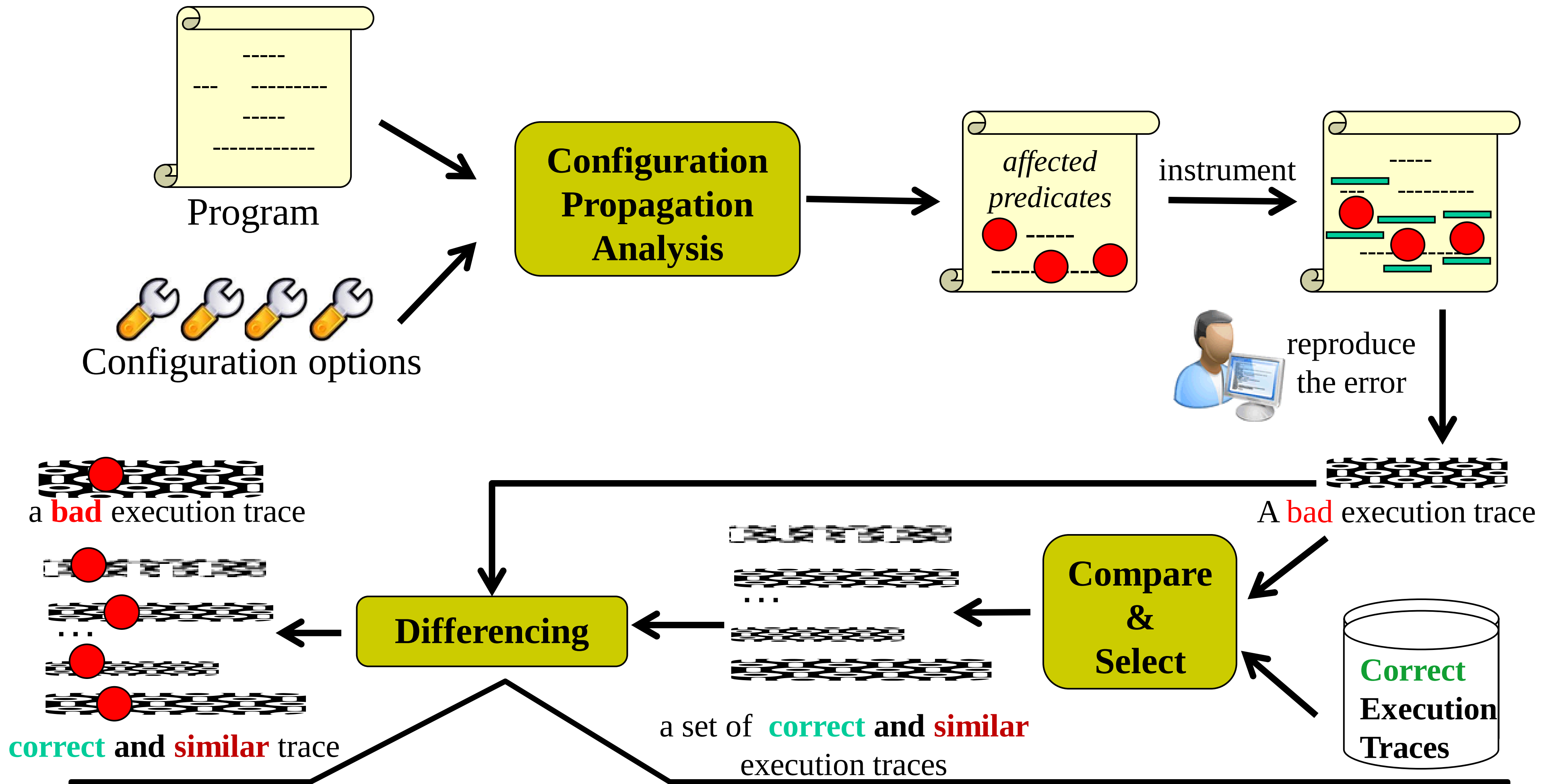
How often an affected predicate evaluates to true

The ConfDiagnoser technique



1. Convert a trace into a vector
2. Compute the cosine similarity between 2 vectors

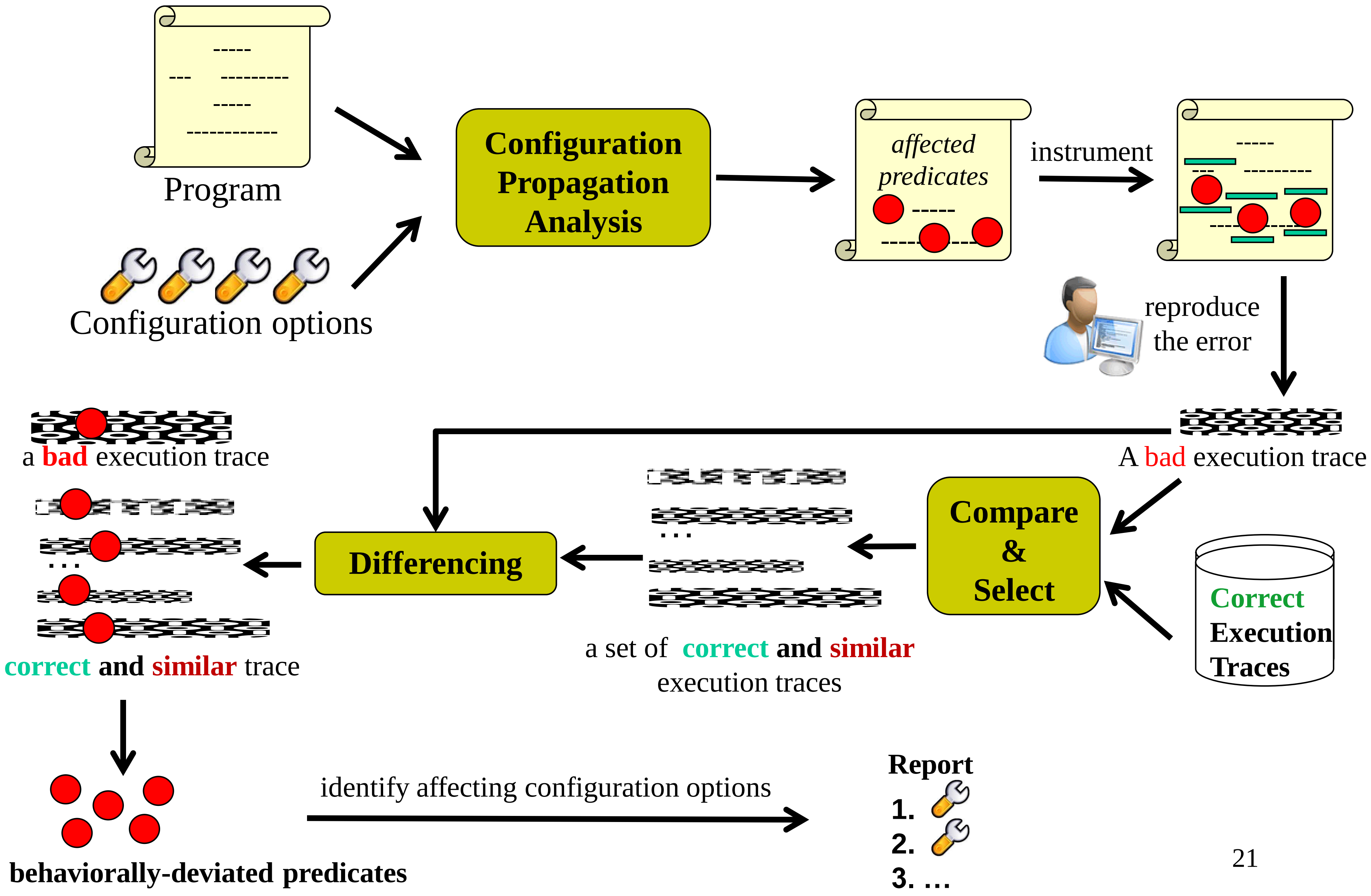
The ConfDiagnoser technique



1. **Compare each predicate's behavior between the bad and correct traces .**
2. **A metric for predicate's behavior :**

$$\frac{1}{\frac{1}{exec\ frequency} + \frac{1}{true\ ratio}}$$

The ConfDiagnoser technique



Outline

- Example
- The ConfDiagnoser Technique
- ➡ Evaluation
- Related Work
- Contributions

Research questions

- How effective is ConfDiagnoser in diagnosing errors?
  - Diagnosis accuracy
  - Time cost
  - Comparison with three existing techniques
    - One configuration error diagnosis technique
    - Two general automated debugging techniques

14 configuration errors from 5 subjects

Subject	LOC	#Options	#Non-crashing Errors	#Crashing Errors
Randoop	18587	57	1	
Weka	3810	14	1	
Synoptic	19153	37	1	
Soot	159271	49	1	
JChord	23391	79	1	9

Collected from FAQ, forum posts, mailing list questions …

Collected from [Rabkin ASE'11]

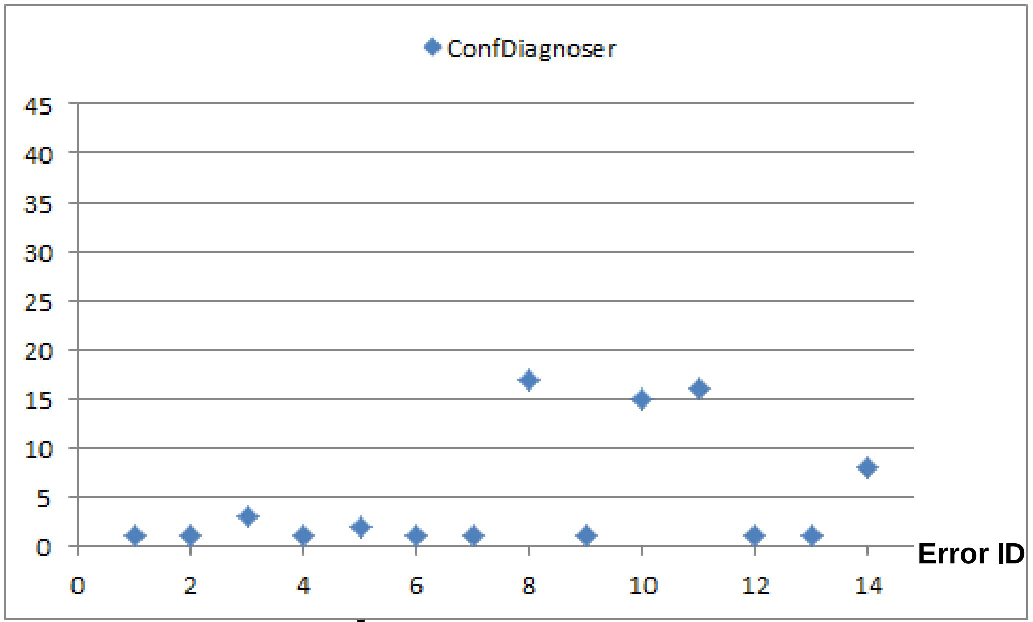
- Correct executions for each program
  - 6 – 16 examples from its user manual

ConfDiagnoser's accuracy and efficiency

- Measure accuracy by the absolute root cause ranking

1. 🔧
2. 🔧
3. …

Root Cause Rank



non-crashing errors | crashing errors

Average rank: 5th

8 errors ranks first

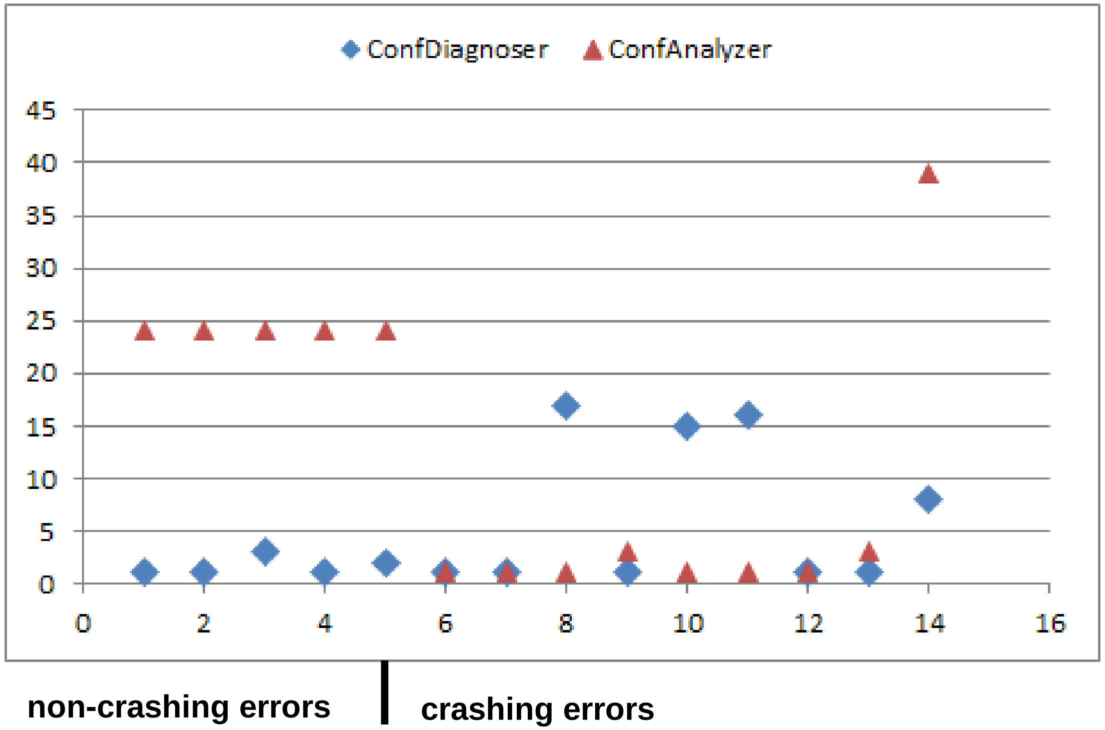
10 errors ranks in the top 3

Better for non-crashing errors

- Time cost: 4 mins / error (on average)

Comparison with ConfAnalyzer [Rabkin '11]

- The most recent configuration error diagnosis technique
  - Use dynamic tainting
  - Only supports crashing errors

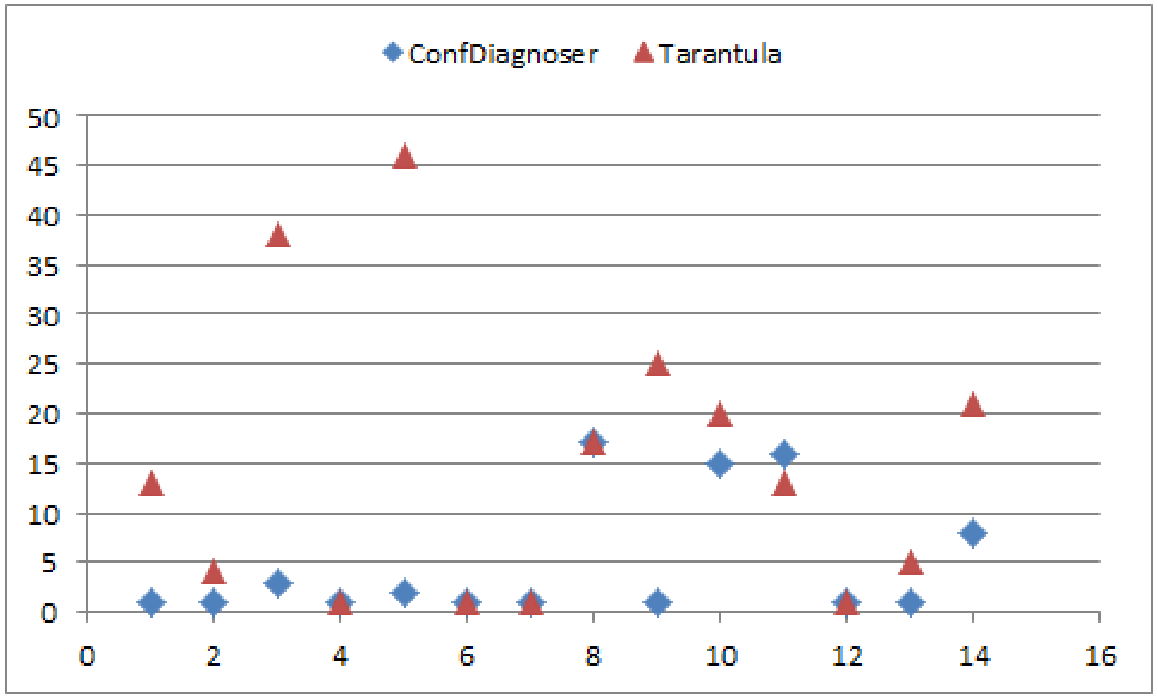


Average rank
- **ConfDiagnoser: 5th**
- **ConfAnalyzer: 12th**

ConfDiagnoser produces:
- **Better results on 8 errors**
- **Same results on 3 errors**
- **Worse results on 3 errors**

Comparison with Tarantula [Jones '03]

- Tarantula-based configuration debugging
  - Use statement coverage to localize suspicious statements
  - Use thin slicing to identify the affecting configuration options



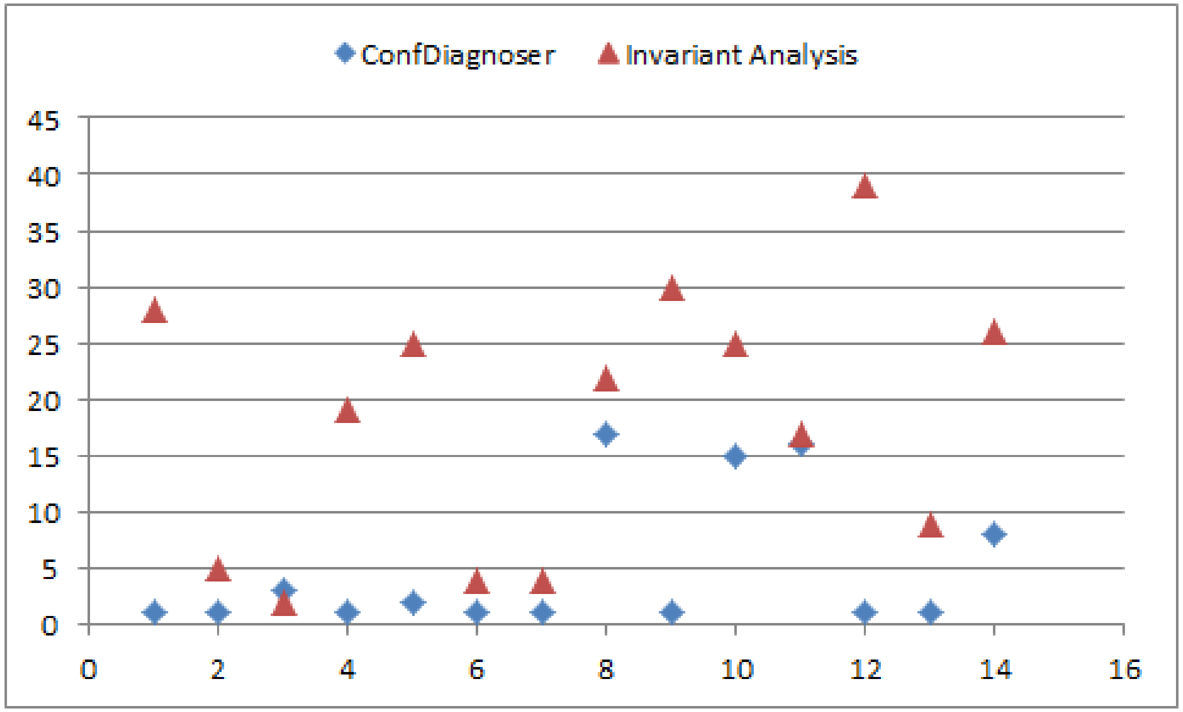
Average rank
- **ConfDiagnoser: 5th**
- **Tarantula: 15th**

Tarantula's statement-level granularity is too fine-grained

- Many statements get the same suspiciousness value
- Statement coverage does not indicate predicate evaluation results

Comparison with Invariant Analysis [McCamant '04]

- Invariant Analysis-based configuration debugging
  - Use method invariant difference to localize suspicious methods
  - Use thin slicing to identify the affecting configuration options



Average rank

- ConfDiagnoser: 5th

- Invariant Analysis: 18th

Invariant analysis' method-level granularity is too coarse-grained

- Some control flow changes inside a method are not be reflected by invariants

Experimental conclusion

- ConfDiagnoser is accurate and efficient
- ConfDiagnoser outperforms existing techniques
  - One configuration error diagnosis technique
  - Two general automated debugging techniques

Outline

- Assumption, Goal, and Insight
- The ConfDiagnoser Technique
- Evaluation
- ➡ Related Work
- Contributions

Related work on configuration error diagnosis

- Tainting-based techniques
  - Dynamic tainting [**Attariyan'08**]
  - Static tainting [**Rabkin'11**]

  Focuses exclusively on crashing errors

- Search-based techniques
  - Delta debugging [**Zeller'02**], Chronus [**Whitaker'04**]

  Requires a correct state for comparison, or OS-level support

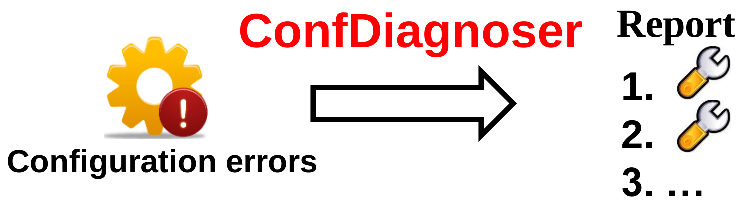
- Domain-specific techniques
  - PeerPressure [**Wang'04**]
  - RangeFixer [**Xiong'12**]

  Targets a specific kind of configuration errors, and does not support a general language like Java

Outline

- Assumption, Goal, and Insight
- The ConfDiagnoser Technique
- Evaluation
- Related Work
- ➡ Contributions

Contributions



- A technique to diagnose configuration errors

  *Compare **relevant predicate** behaviors between executions*
  - Fully automated
  - Can diagnose both crashing and non-crashing errors
  - Requires no OS-level support

- Experiments that demonstrate its usefulness
  - Accurate and fast
  - Outperforms three existing techniques

- The ConfDiagnoser tool implementation

  http://config-errors.googlecode.com

[Backup Slides]

Representation of configuration options inside ConfDiagnoser

- A configuration option is represented as a class field

- An example configuration option in Randoop:
  - $\underbrace{\texttt{randoop.main.GenInputsAbsract}}_{\text{Class name}}.\underbrace{\texttt{maxsize}}_{\text{Field name}}$

- Made a 24-LOC syntactic change to 5 subject programs
  - Transform configuration option into class field