Localising Faults in Test Execution Traces

Gulsher Laghari, Alessandro Murgia and Serge Demeyer

August 30, 2015

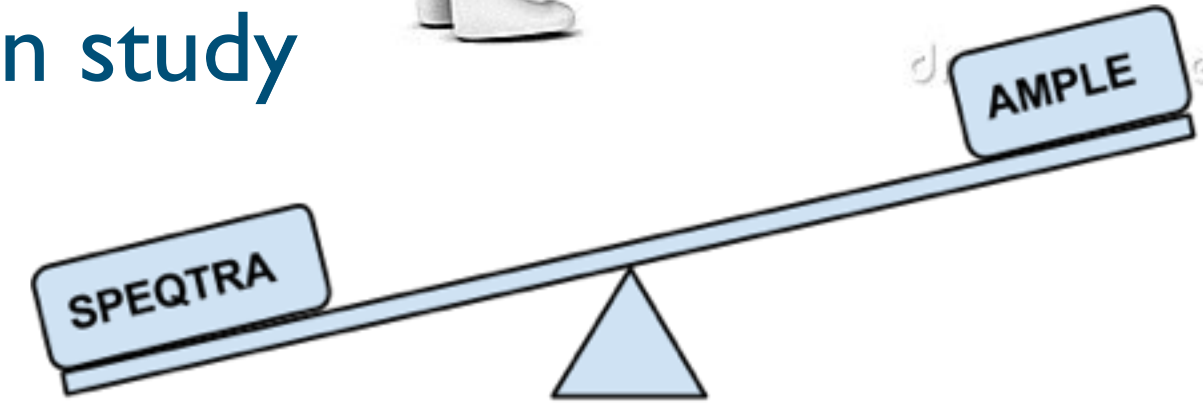
Ansymo
Antwerp Systems & Software Modelling
University of Antwerp

Universiteit Antwerpen

Overview

- Fault localisation

- Replication study



Continuous integration

Jenkins

Strider CD

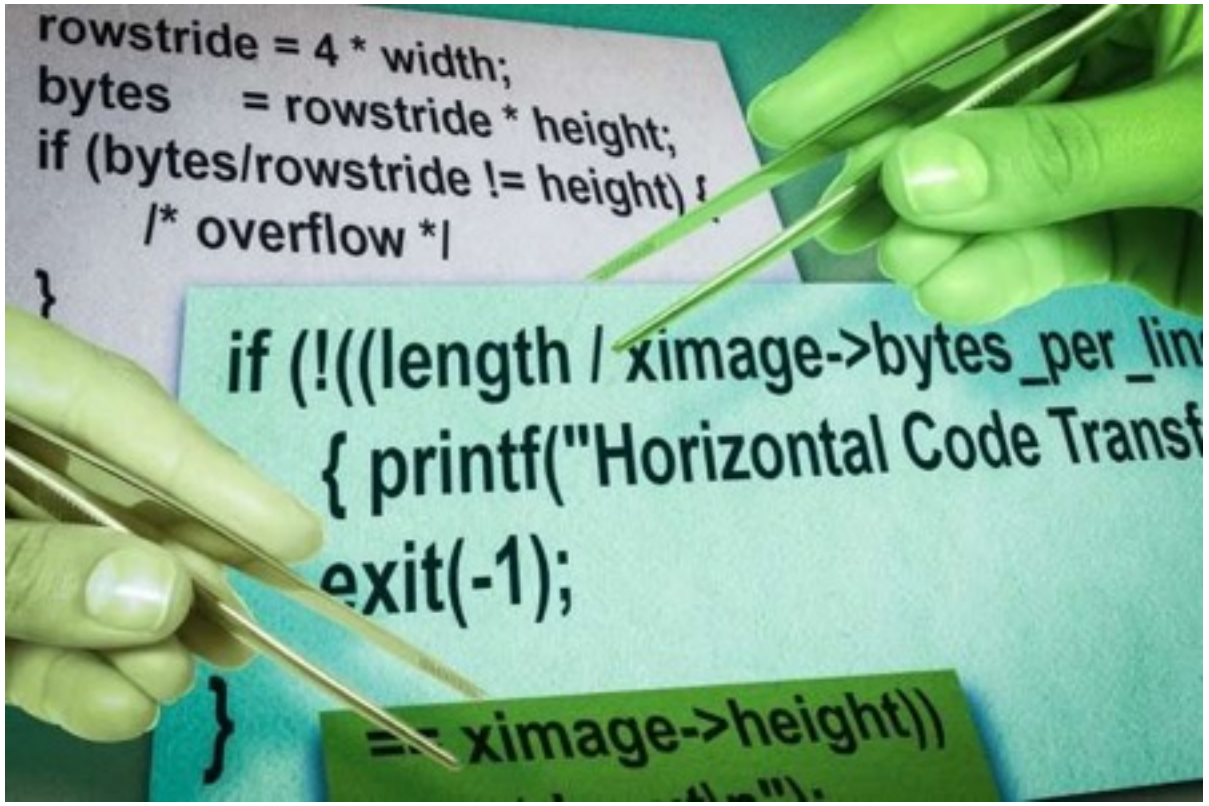
Open Source Continuous Integration & Deployment Server

TeamCity

Bamboo

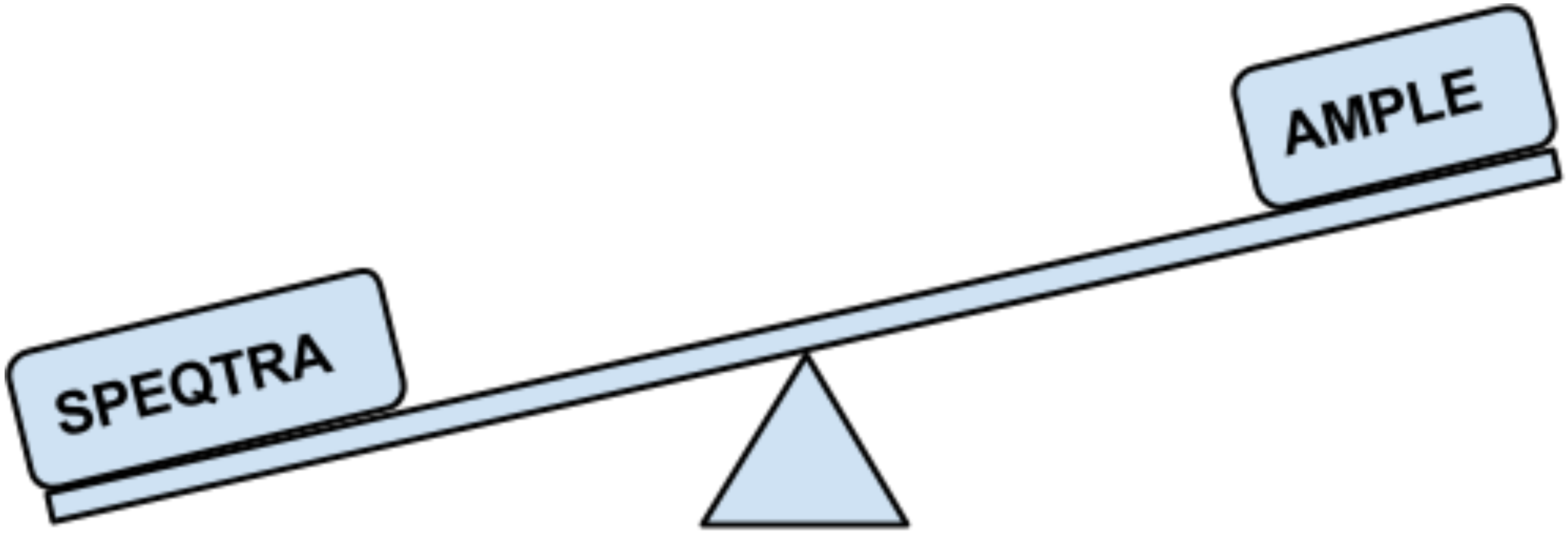
Introduction

Fault localisation

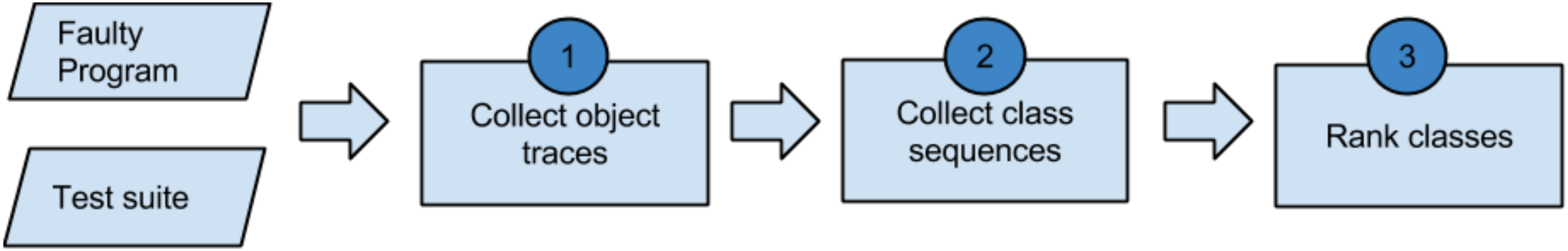


```
rowstride = 4 * width;
bytes     = rowstride * height;
if (bytes/rowstride != height) {
    /* overflow */
}
if (!((length / ximage->bytes_per_lin
    { printf("Horizontal Code Transf
    exit(-1);
}
      == ximage->height))
```

Introduction



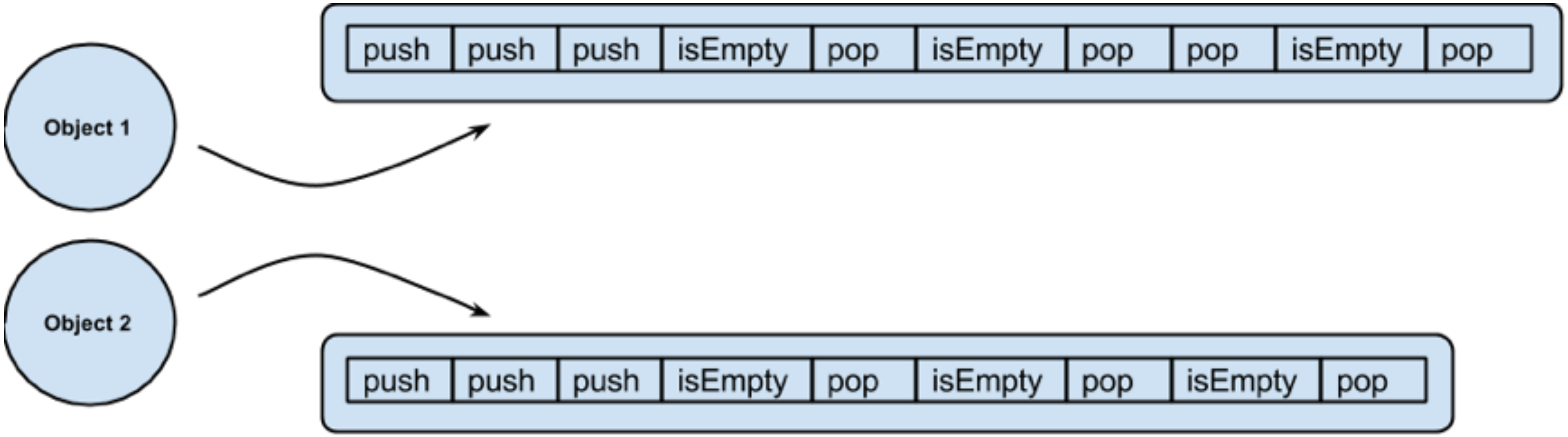
Heuristics under investigation -1/5



Heuristics under investigation -2/5

1. Collecting traces

- Traces for every created object



Object 1

push | push | push | isEmpty | pop | isEmpty | pop | pop | isEmpty | pop

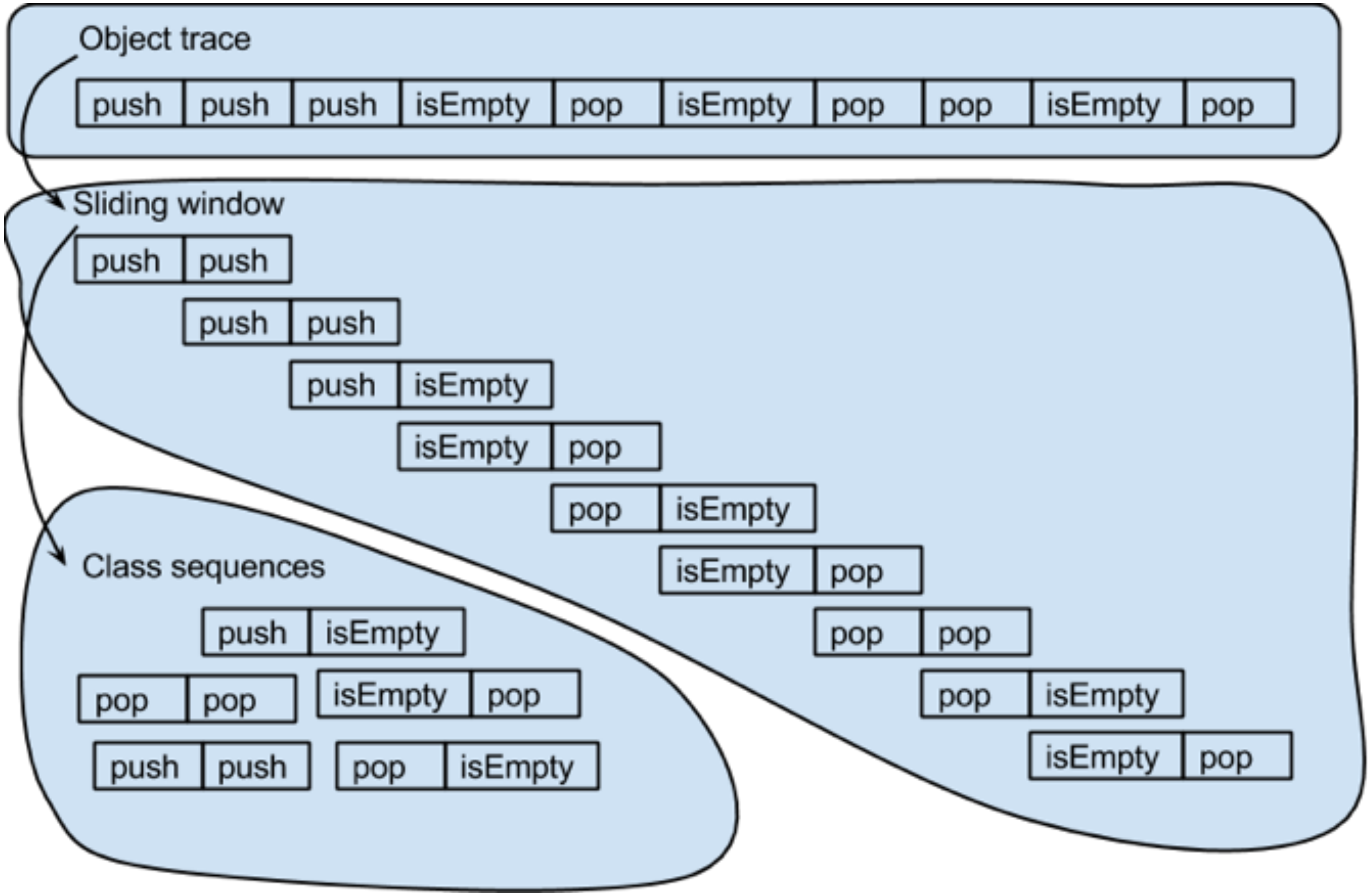
Object 2

push | push | push | isEmpty | pop | isEmpty | pop | isEmpty | pop

Heuristics under investigation -3/5

2. Collecting Class sequences — AMPLE

Sliding Window



Object trace

push push push isEmpty pop isEmpty pop pop isEmpty pop

Sliding window

push push

push push

push isEmpty

isEmpty pop

pop isEmpty

isEmpty pop

pop pop

pop isEmpty

isEmpty pop

Class sequences

push isEmpty

pop pop

isEmpty pop

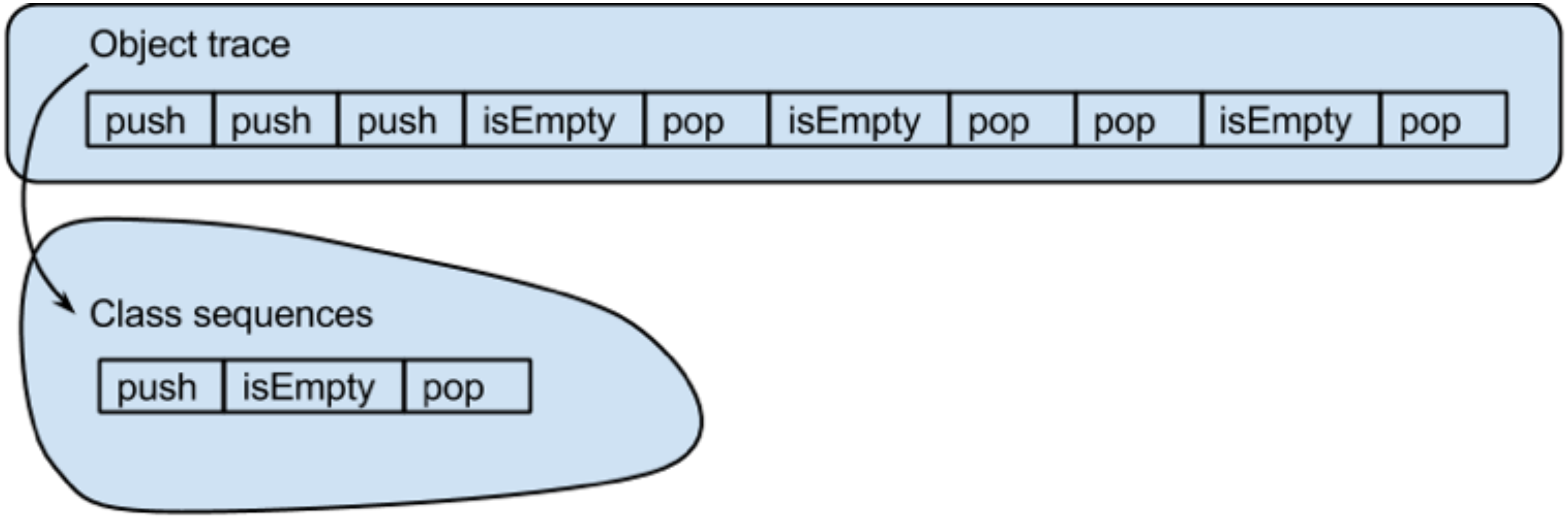
push push

pop isEmpty

Heuristics under investigation -4/5

2. Collecting Class sequences — SPEQTRA

Frequent sequences



Heuristics under investigation -5/5

3. Ranking classes

Weight per class sequence

AMPLE weighting scheme	SPEQTRA weighting scheme
$W(X) = \begin{cases} \frac{k(X)}{n} & \text{if } X \text{ not in failing test} \\ 1 - \frac{k(X)}{n} & \text{if } X \text{ in failing test} \end{cases}$	$W(X) = \frac{a_{11}(X)}{a_{11}(X) + a_{01}(X) + a_{10}(X)}$
Where: • n = number of passing tests • k(X) = number of passing tests that contain sequence X	Where : • a_{11}(X) = number of failing tests in which sequence is found • a_{10}(X) = number of passing tests in which sequence is found • a_{01}(X) = number of failing tests in which sequence is not found

Weight per class

$$W(C) = \frac{1}{n} \sum_{i=1}^{n} W(X_i)$$

Where n = number of sequences in class C and W(Xi) is weight of sequence

Experimental Setup

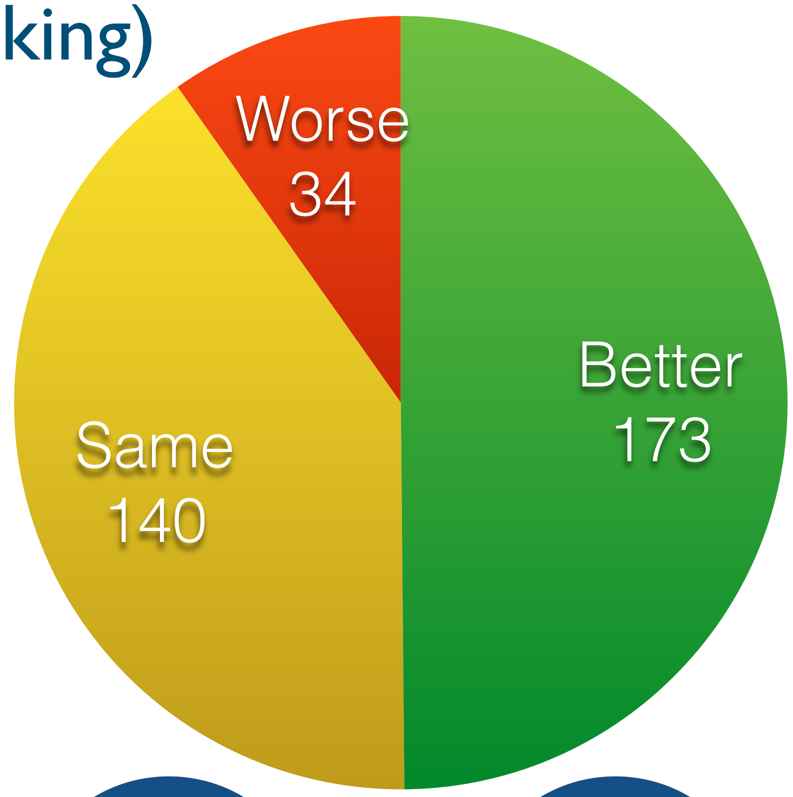
Replication case: NanoXML

Version*	# of classes	LOC	# of faults	# of tests
1	16	4334	7	214
2	19	5806	7	214
3	21	7185	10	216
5	23	7646	8	216

* Version 4 has no documented faults

Results and Discussion - 1/5

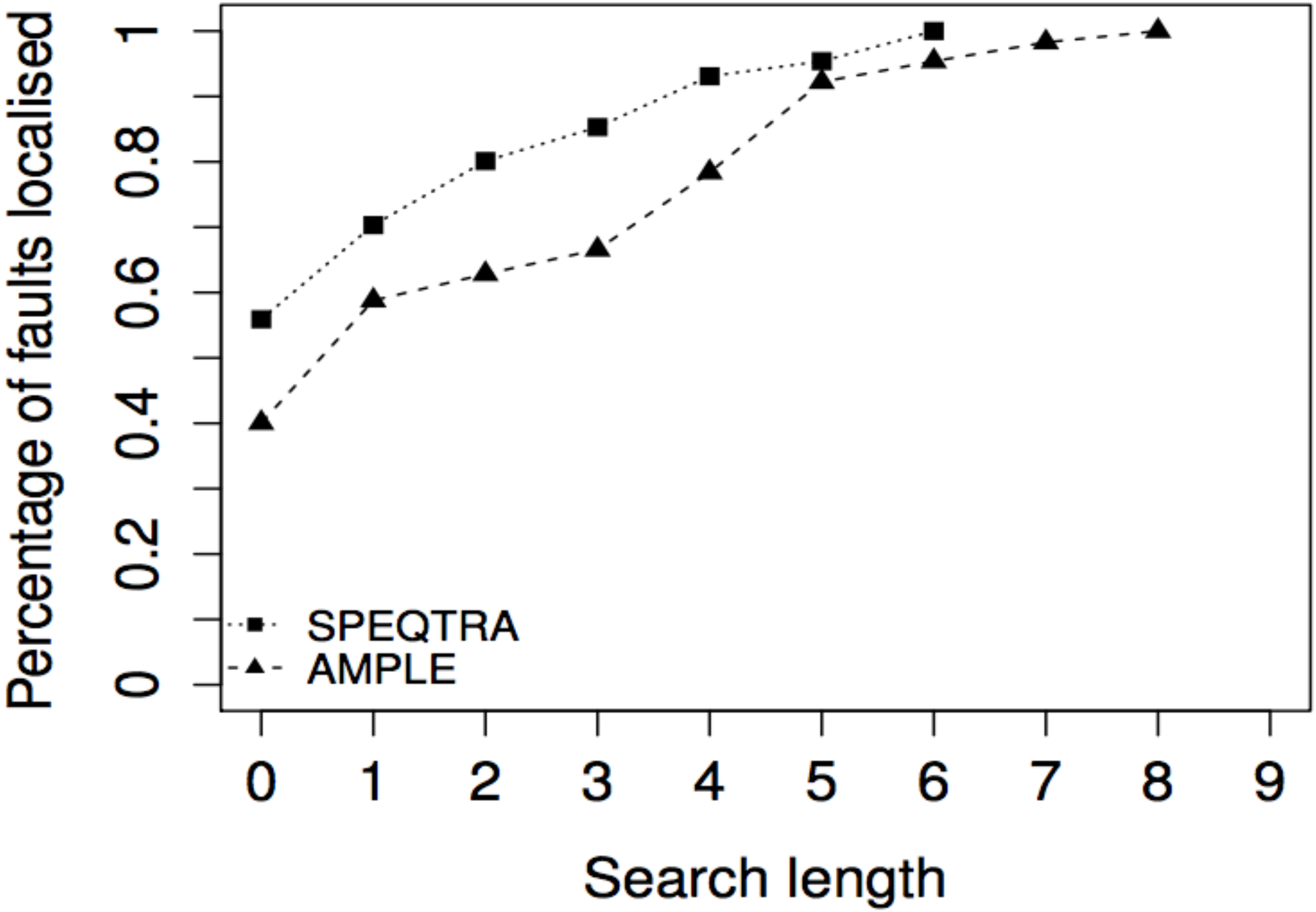
SPEQTRA vs AMPLE (347 Ranking)



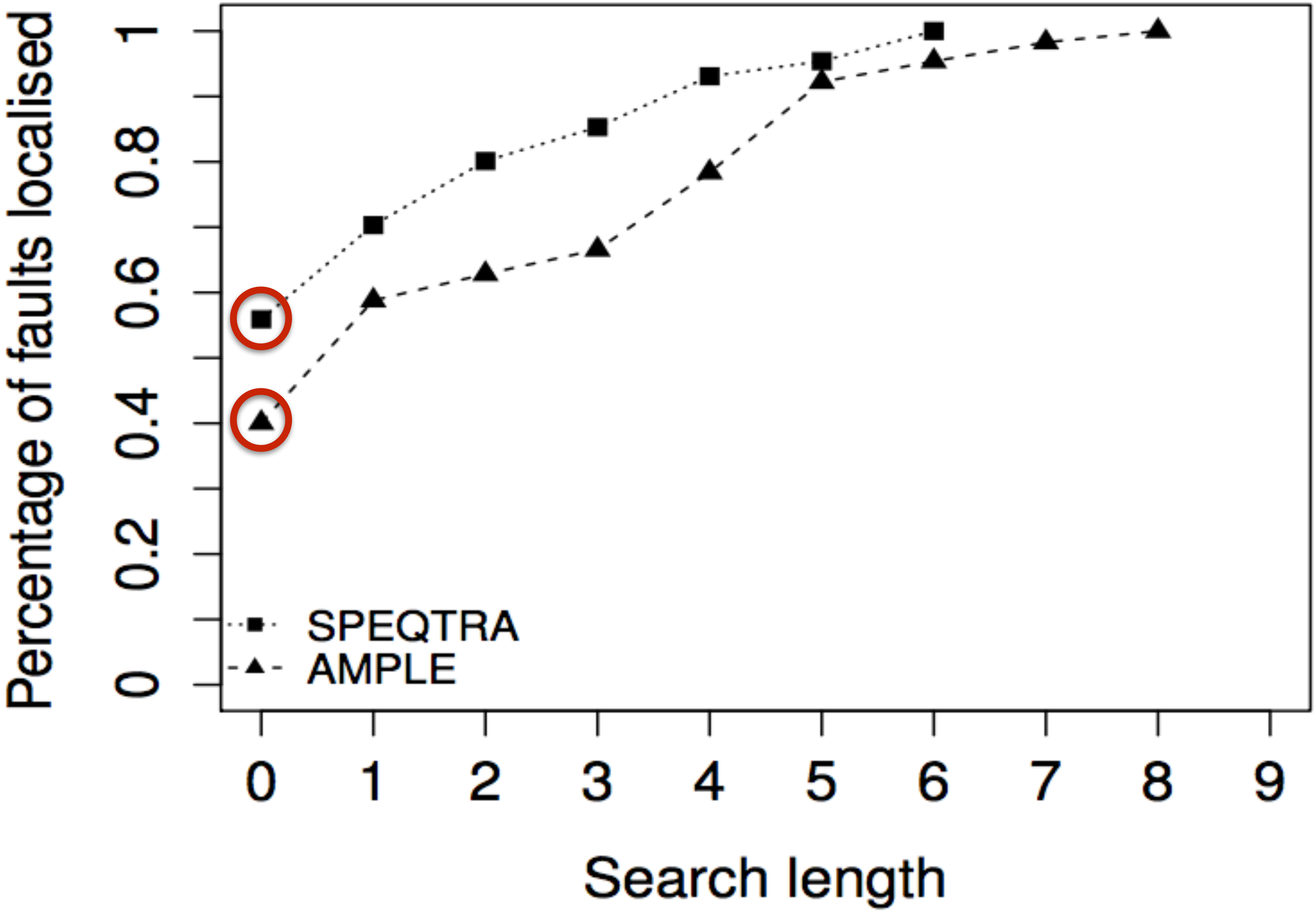
Average search length

SPEQTRA	AMPLE
1.20	2.07

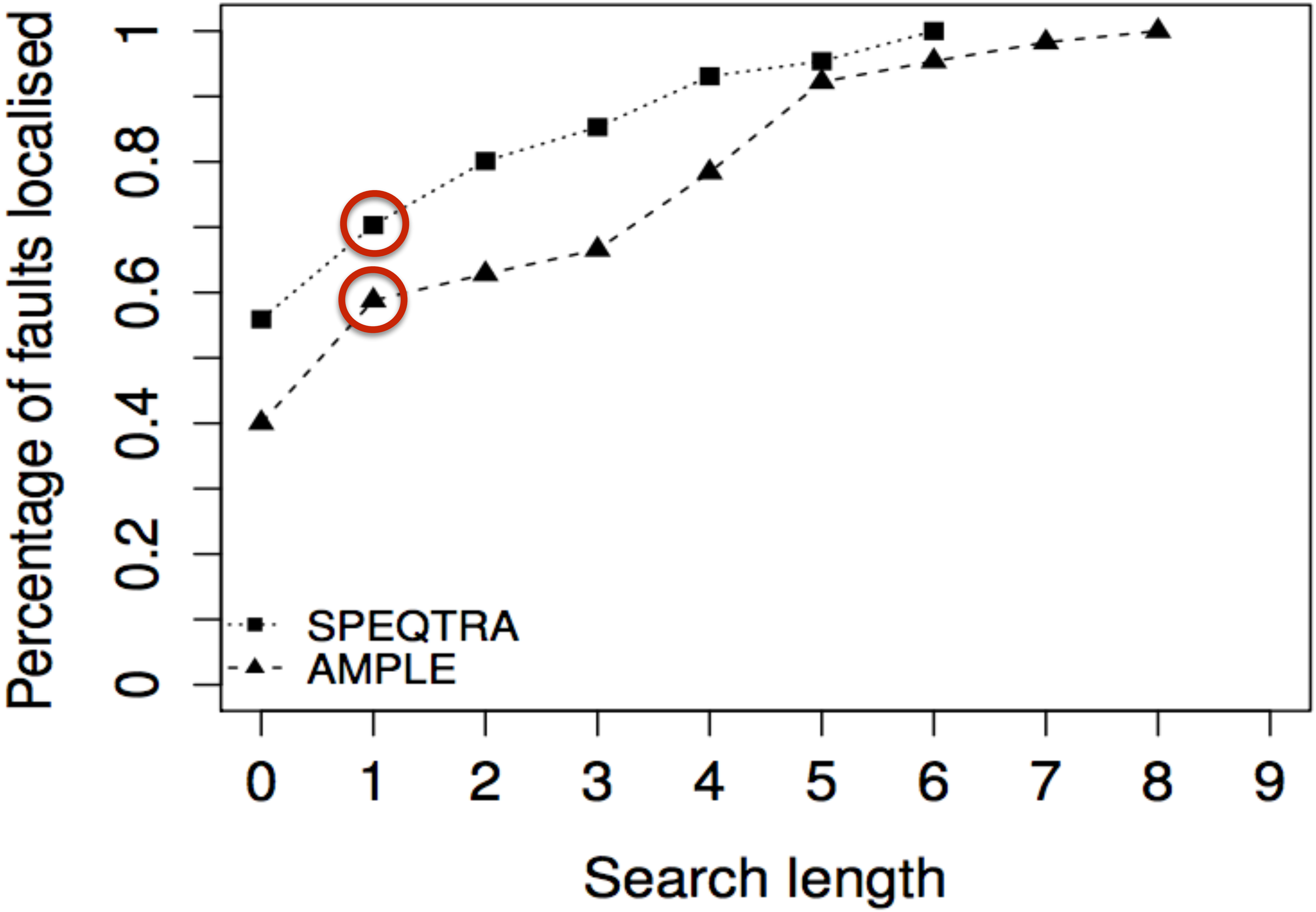
Results and Discussion - 2/5



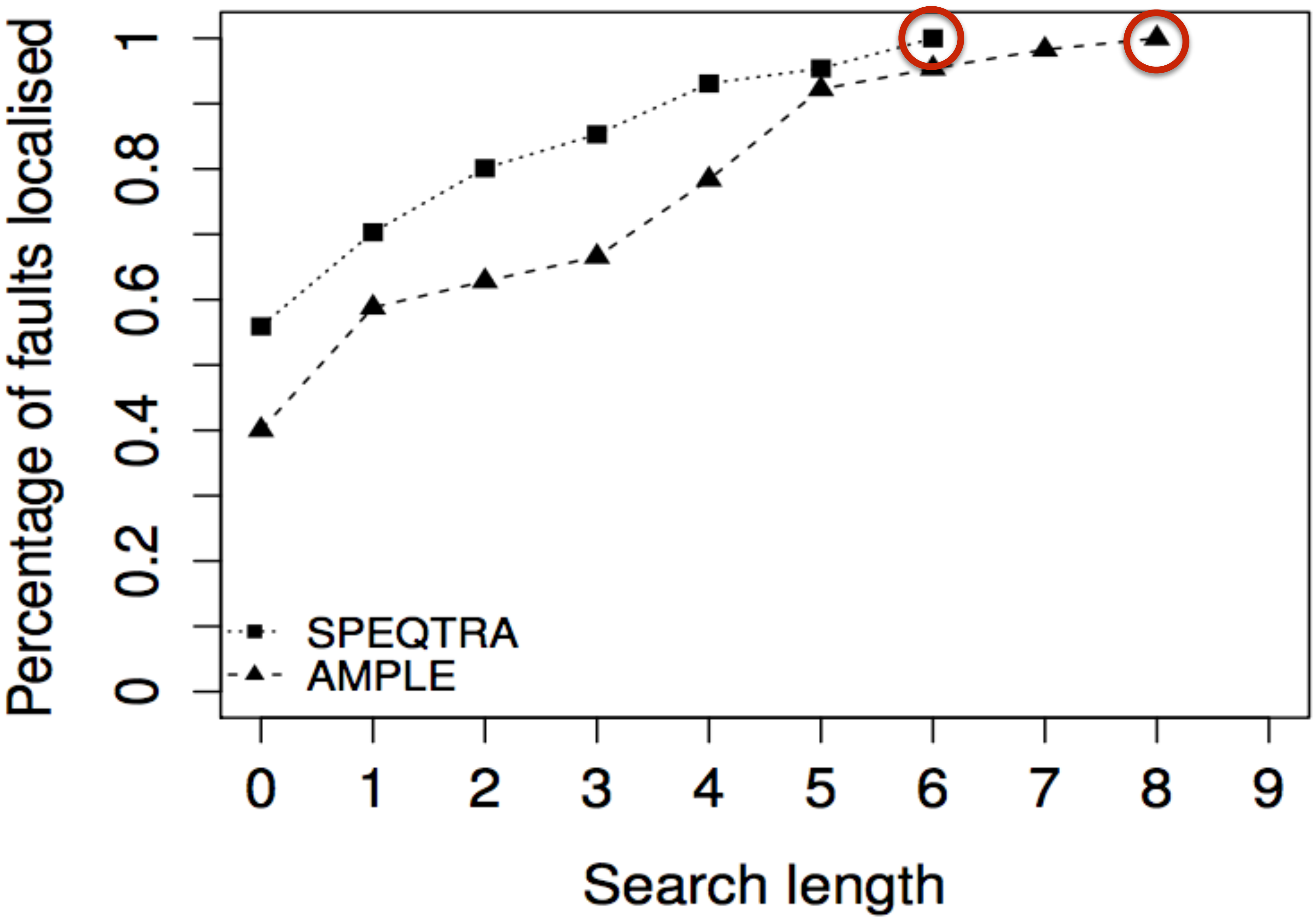
Results and Discussion - 3/5



Results and Discussion - 4/5



Results and Discussion - 5/5



Conclusion

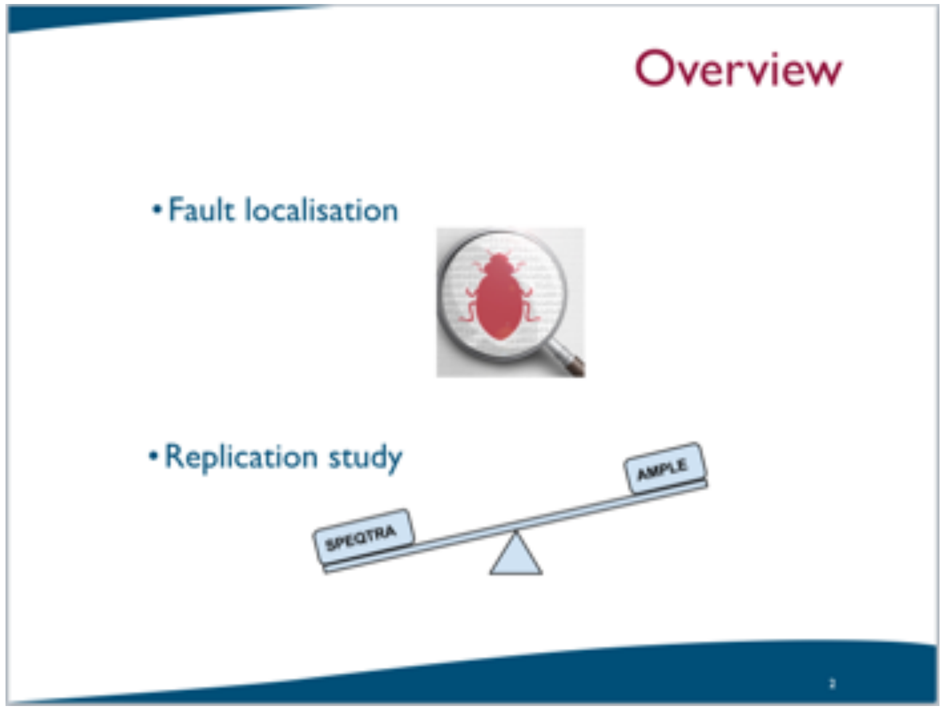
SPEQTRA

- Save computation time
- Handle faulty call sequence of any length
- Locate the faults at class level

SPEQTRA performed better than AMPLE

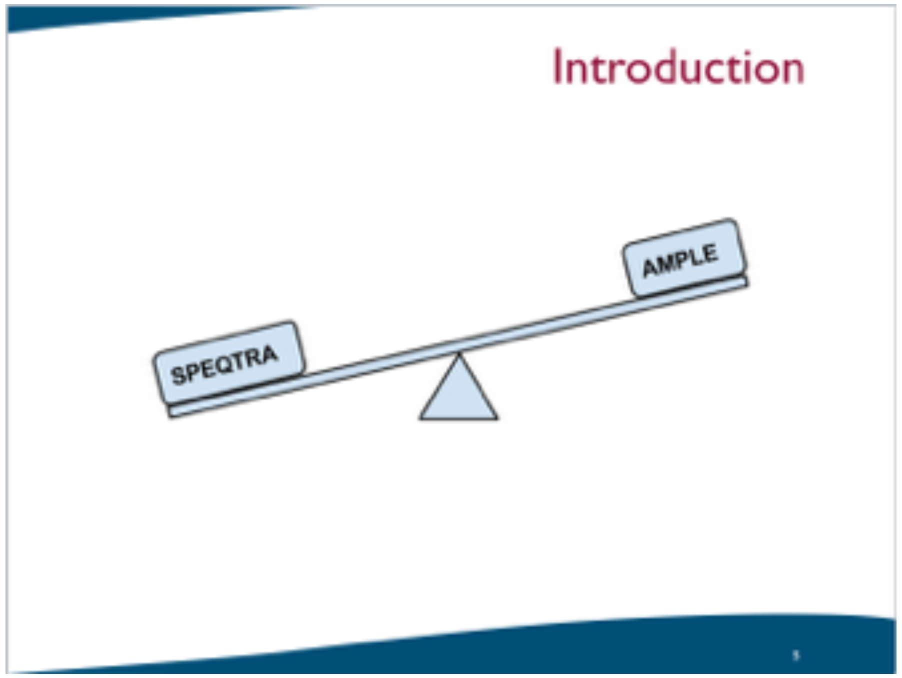
- Search Length 0 (56% / 40%)
- Worst search length (6 / 8)

Summary

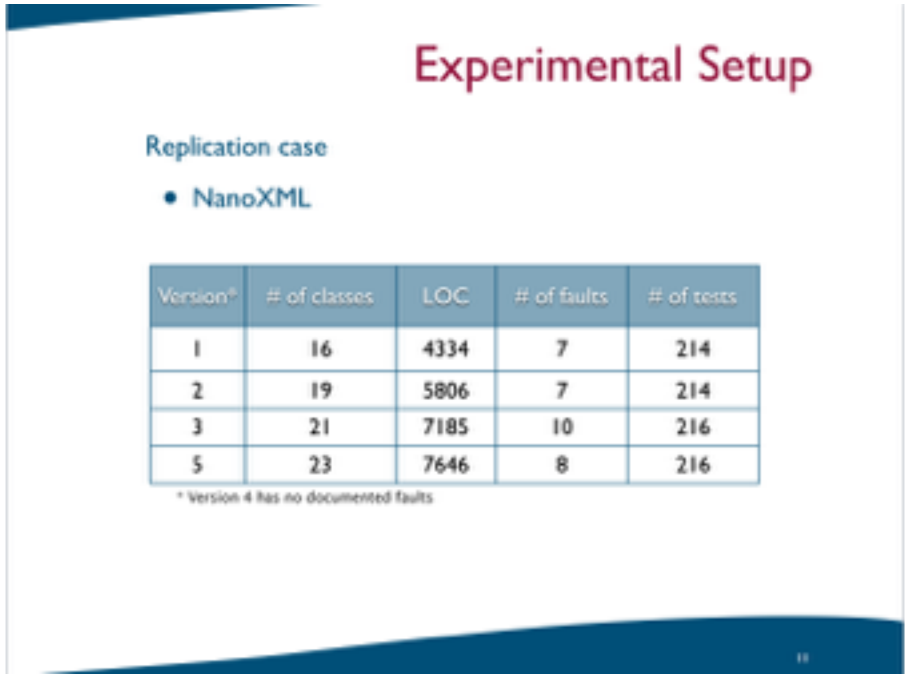


Overview

- Fault localisation
- Replication study



Introduction



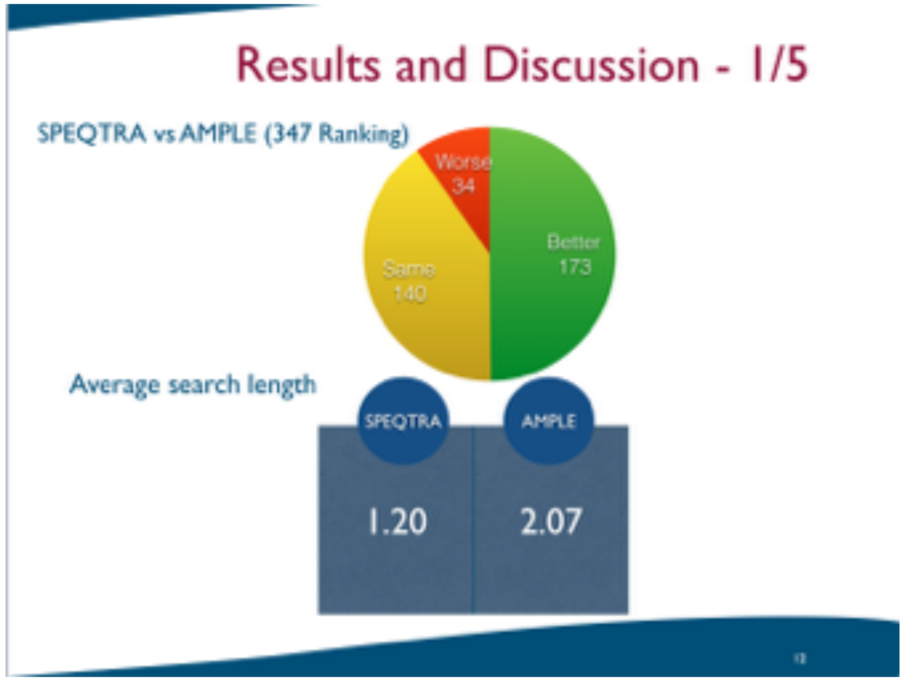
Experimental Setup

Replication case

- NanoXML

Version*	# of classes	LOC	# of faults	# of tests
1	16	4334	7	214
2	19	5806	7	214
3	21	7185	10	216
5	23	7646	8	216

* Version 4 has no documented faults



Results and Discussion - 1/5

SPEQTRA vs AMPLE (347 Ranking)

Average search length

SPEQTRA	AMPLE
1.20	2.07