

An Extensive Study of Static Regression Test Selection in Modern Software Evolution

Owolabi Legunsen, Farah Hariri, August Shi,
Yafeng Lu, Lingming Zhang, and Darko Marinov

FSE 2016

Seattle, Washington

November 16, 2016



CCF-1409423, CCF-1421503,
CCF-1438982, CCF-1439957,
CCF-1566589

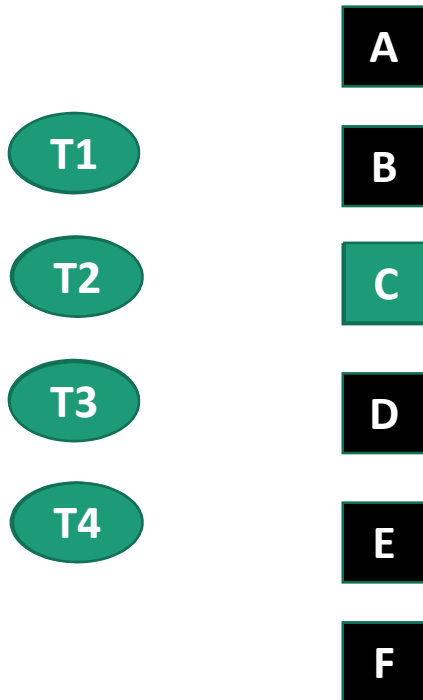


ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN



Regression Testing

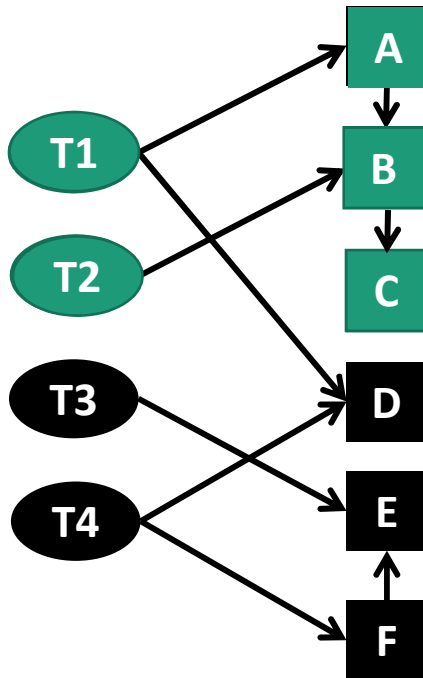
- Rerun tests to ensure that code changes did not break existing functionality



- **Problem:** Regression testing can be very slow! (many tests)

Regression Test Selection (RTS)

- Speed up regression testing by rerunning **only** tests that are **affected** by code changes



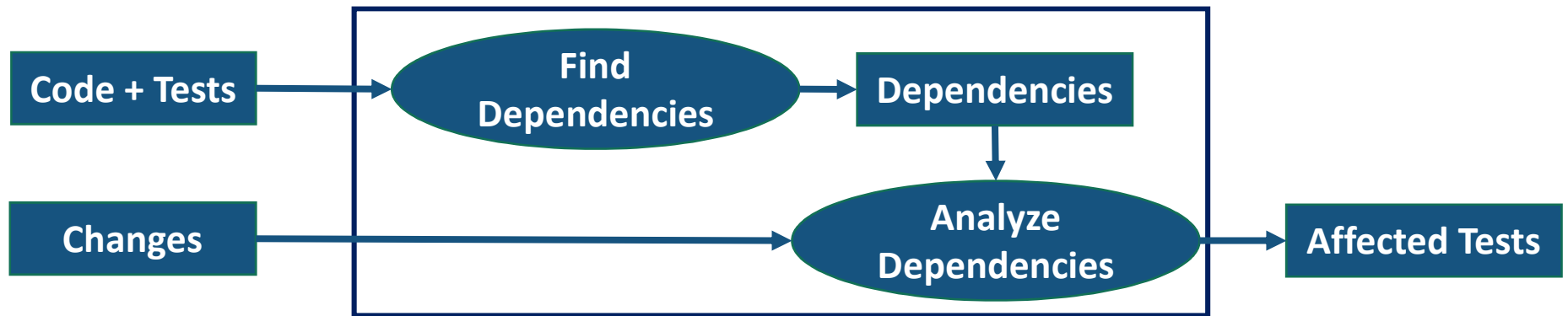
Finding dependencies can be done statically or dynamically

- This paper: we studied **static RTS** approaches and compared with state-of-the-art **dynamic RTS**

Motivation for our Study

- Dynamic RTS has been getting adopted recently
- Dynamic RTS may not always be applicable
 - Instrumentation costs can be high
 - Dependencies may be incomplete, e.g., due to non-determinism
- Static RTS was proposed previously but not evaluated at scale on modern software

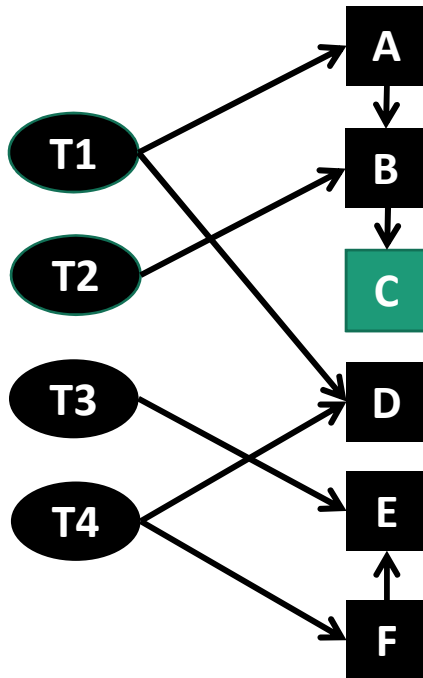
How RTS works



- An **affected test** can behave differently due to code changes
- A test is affected if any of its dependencies changed

Finding and Analyzing Dependencies

- Dependencies: entities that can affect test behavior



1. Finding Dependencies:

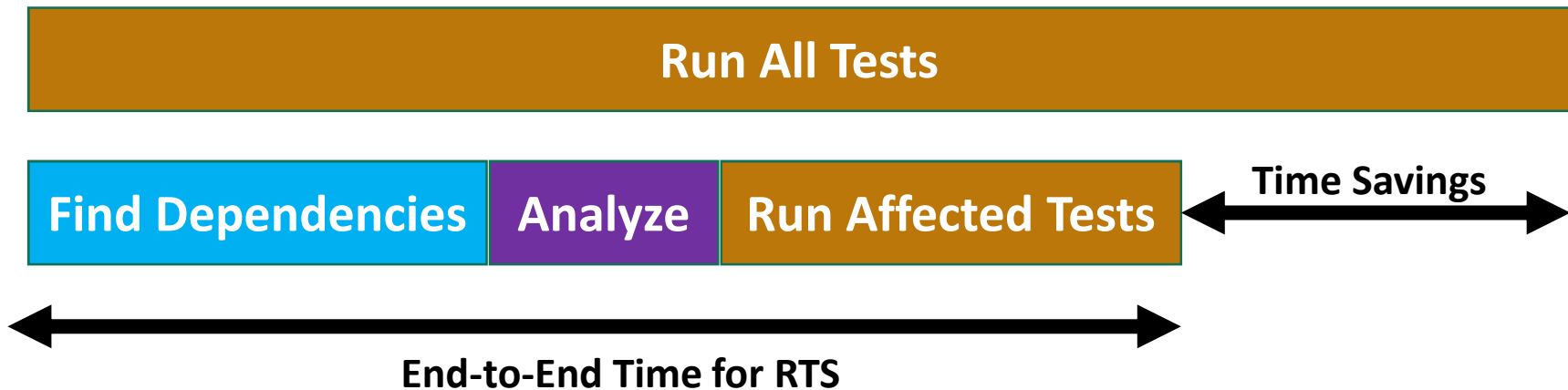
- T1 depends on A, B, C, D, T1
- T2 depends on B, C, T2
- T3 depends on E, T3
- T4 depends on D, E, F, T4

2. Analyzing Dependencies:

- T1 & T2 are affected

Important RTS Considerations




- **End-to-end time** of RTS must be less than time to run all tests



- RTS is **safe** if it selects to rerun *all* affected tests
- RTS is **precise** if it selects to rerun *only* affected tests

RTS Techniques Evaluated

- Finding dependencies can be done dynamically or statically
- Dependencies can be at different levels of granularity, e.g., methods, classes, jar files, etc.
- In this paper, we compare these approaches:

	Class-Level Dynamic	Class-Level Static	Method-Level Static
End-to-End Time	?	?	
Safety	?	?	
Precision	?	?	

See details on method-level RTS in paper



Class-Level Dynamic RTS (Ekstazi^[1])

Ekstazi

- **Find Dependencies:** dynamically track classes used while running each test class
- **Changes:** classes whose .class (bytecode) files differ
- **Analyze Dependencies:** select test classes for which any of its dependencies changed

Class-Level STAtic RTS (STARTS)

- First, statically build a class dependency graph
 - Each class has an edge to direct parents and referenced classes
- **Find Dependencies:** classes reachable from test class in the graph
- **Changes:** computed in same way as Ekstazi
- **Analyze Dependencies:** select test classes that reach a changed class in the graph

Variants of RTS Techniques

- We studied 12 RTS techniques in total
- 2 variants of the static/dynamic class-level RTS
 - **Offline:** pre-compute dependencies *before* changes are known
 - **Online:** compute dependencies *after* changes are known
- 8 variants of static method-level RTS technique

See details on method-level RTS in paper

Research Questions

- RQ1: How do RTS techniques compare w.r.t. number of tests selected?
- RQ2: How do RTS techniques compare w.r.t. end-to-end time?
- RQ3: How do static RTS techniques compare with class-level dynamic RTS in terms of precision and safety?
- RQ4: How do variants of method-level static RTS influence the cost/safety trade-offs?

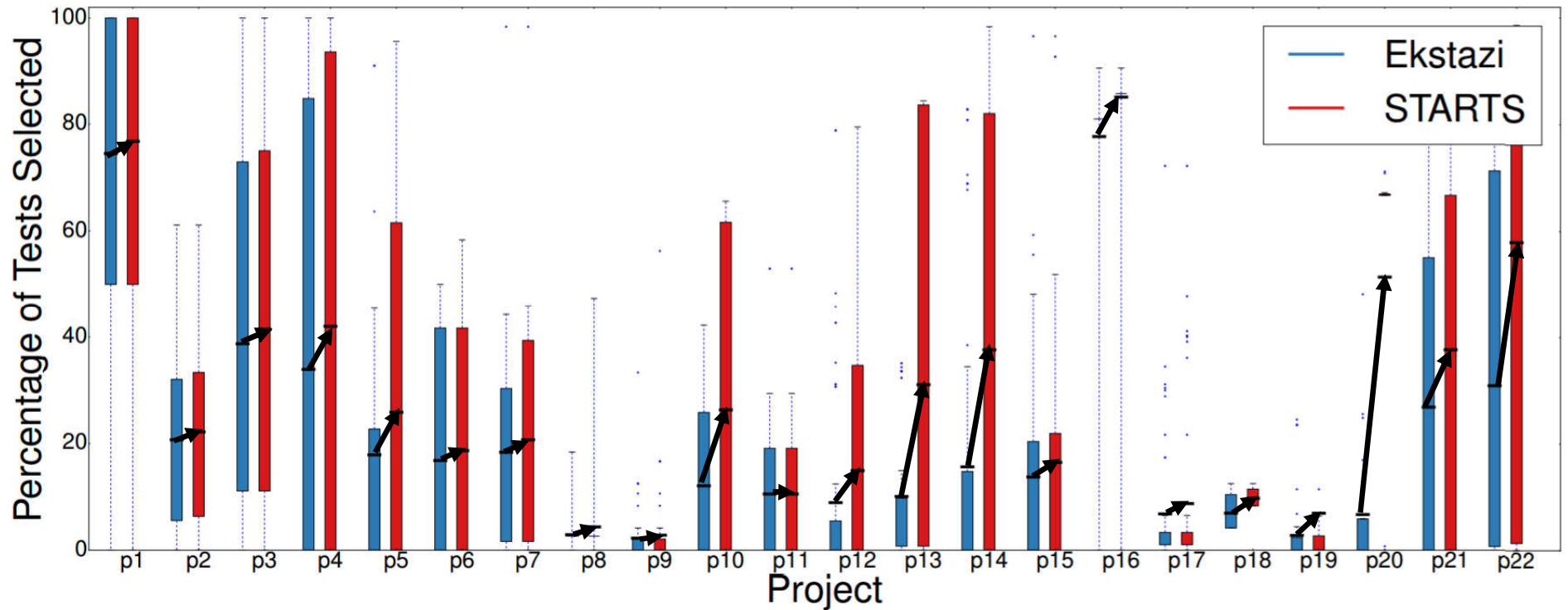
See answer to RQ4 in paper

Experimental Setup



- 22 open-source projects from ASF and GitHub
 - Single-module Maven projects with JUnit4 tests
 - Project sizes: from 2 kLOC to 185 kLOC
- 985 revisions of these 22 projects
 - Selection criteria: subset of latest 100 commits
 - Compile successfully
 - All tests pass
 - Ekstazi runs successfully

RQ1: Tests Selected



Ekstazi selects **fewer** tests
than STARTS
20.6% vs. 29.4%

RQ2: End-to-End Time

Project	EKSTAZI (OFFLINE)			EKSTAZI (ONLINE)			STARTS (OFFLINE)			STARTS (ONLINE)		
	min	max	avg	min	max	avg	min	max	avg	min	max	avg
p1	92.4	132.7	115.5	88.1	147.3	126.2	72.4	113.8	102.9	75.7	130.9	114.8
p2	82.9	122.3	105.4	85.4	146.2	121.6	64.4	114.0	90.9	65.8	127.3	99.5
p3	74.9	122.6	103.0	76.4	139.0	113.9	65.0	109.6	91.8	64.3	124.5	97.8
p4	92.4	119.4	106.7	97.0	131.2	114.3	83.0	117.7	99.3	83.0	125.0	102.2
p5	81.2	112.8	101.4	87.2	129.1	106.2	73.7	116.7	94.5	80.7	116.9	97.7
p6	73.6	124.0	93.4	70.2	148.3	98.3	61.3	114.6	85.7	62.0	125.3	87.6
p7	71.5	107.0	88.9	75.3	114.4	95.0	62.2	97.9	77.5	61.2	106.2	82.9
p8	31.6	109.6	44.4	31.7	117.2	49.0	25.5	104.5	42.9	24.8	330.2	54.1
p9	41.7	93.2	56.4	45.4	96.0	58.2	35.6	80.7	49.6	36.0	91.9	51.3
p10	43.6	91.3	54.1	46.1	108.9	57.9	34.7	105.0	64.2	35.6	110.2	65.3
p11	29.5	99.4	49.9	28.3	103.2	50.1	24.4	105.9	44.8	23.5	105.6	47.1
p12	28.4	101.8	44.0	26.6	109.8	45.5	25.7	102.4	51.3	25.5	252.4	55.8
p13	29.4	74.7	44.7	30.6	83.0	47.6	25.2	99.2	55.6	25.9	103.5	57.1
p14	20.4	104.2	46.1	23.1	111.9	48.9	19.0	107.2	54.5	19.3	107.4	55.7
p15	6.5	104.7	26.4	6.7	107.5	27.0	5.6	104.9	26.0	5.6	103.4	26.1
p16	35.1	99.8	94.5	10.9	105.3	97.0	2.8	98.1	68.4	3.0	98.8	66.9
p17	4.1	96.6	19.9	4.1	109.6	21.3	3.4	94.9	22.0	3.4	96.4	22.3
p18	31.6	45.1	35.3	31.9	46.1	36.1	29.9	43.6	34.7	30.4	44.5	34.6
p19	10.9	69.4	17.8	11.3	96.7	20.6	9.6	97.5	19.4	9.7	98.0	19.9
p20	47.8	90.9	67.1	47.9	100.7	71.0	47.0	99.3	85.9	47.1	100.5	86.2
p21	1.1	101.5	52.4	1.1	101.8	52.5	0.9	100.7	59.8	0.8	100.8	59.7
p22	0.9	102.4	41.1	0.9	104.3	42.0	0.8	87.5	52.7	0.7	89.3	53.2
Average	42.3	101.2	64.0	42.1	111.7	68.2	35.1	100.7	62.5	35.6	122.2	65.3

RQ3: Safety and Precision

- Safety and precision were measured against Ekstazi
- **Safety violation:** STARTS misses Ekstazi-selected tests:

$$\text{SafetyViolation} = \frac{|E \setminus S|}{|E \cup S|}$$

E = tests selected by Ekstazi

S = tests selected by STARTS

- **Precision violation:** STARTS selects tests that Ekstazi does not:

$$\text{PrecisionViolation} = \frac{|S \setminus E|}{|E \cup S|}$$

RQ3: Safety and Precision

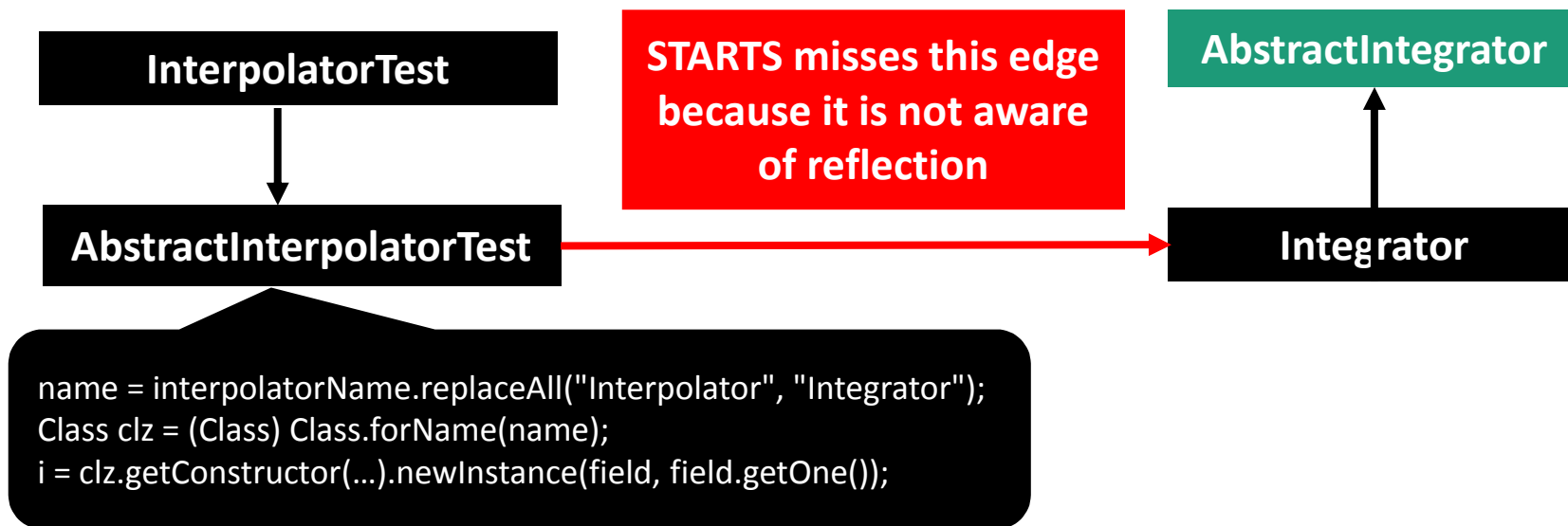
$$\text{SafetyViolation} = \frac{|E \setminus S|}{|E \cup S|}$$

$$\text{PrecisionViolation} = \frac{|S \setminus E|}{|E \cup S|}$$

Project	Safety Violation %				Precision Violation %			
	STARTS				STARTS			
	revs	min	max	avg	revs	min	max	avg
p1	0.0	n/a	n/a	n/a	4.5	33.3	66.7	50.0
p2	0.0	n/a	n/a	n/a	9.3	6.7	75.0	42.3
p3	0.0	n/a	n/a	n/a	20.0	20.0	33.3	27.5
p4	0.0	n/a	n/a	n/a	16.0	5.9	90.9	61.7
p5	0.0	n/a	n/a	n/a	30.3	4.8	100.0	51.1
p6	0.0	n/a	n/a	n/a	22.2	14.3	16.7	16.3
p7	0.0	n/a	n/a	n/a	36.8	3.6	25.0	16.0
p8	0.0	n/a	n/a	n/a	16.9	42.9	100.0	72.9
p9	0.0	n/a	n/a	n/a	6.3	25.0	40.7	28.9
p10	0.0	n/a	n/a	n/a	41.7	28.3	64.8	53.9
p11	0.0	n/a	n/a	n/a	0.0	n/a	n/a	n/a
p12	0.0	n/a	n/a	n/a	39.4	0.8	98.0	50.0
p13	0.0	n/a	n/a	n/a	49.2	6.7	96.3	54.9
p14	0.0	n/a	n/a	n/a	39.1	14.0	98.0	62.4
p15	0.0	n/a	n/a	n/a	16.1	7.1	40.0	26.2
p16	2.0	100.0	100.0	100.0	77.5	5.6	100.0	13.1
p17	0.0	n/a	n/a	n/a	14.3	20.0	56.8	33.3
p18	0.0	n/a	n/a	n/a	80.0	50.0	50.0	50.0
p19	1.8	50.0	50.0	50.0	17.5	35.5	100.0	61.2
p20	0.0	n/a	n/a	n/a	75.0	63.0	100.0	91.0
p21	0.0	n/a	n/a	n/a	51.0	15.4	92.3	31.9
p22	0.0	n/a	n/a	n/a	63.2	7.9	100.0	50.1
Average	0.2	6.8	6.8	6.8	33.0	18.7	70.2	42.9

Reflection caused all Safety Violations

Example simplified from Apache commons-math



Since this paper was accepted ...

- We are making STARTS safer with respect to reflection
- We are evaluating STARTS on larger software systems
- We have improved the STARTS tool to
 - handle multi-module Maven projects
 - find dependencies from bytecode much faster

Conclusions

- We performed the first, large-scale empirical study of static RTS and its comparison with dynamic RTS
- At the class level, we found static RTS (STARTS) comparable with state-of-the-art dynamic RTS (Ekstazi)
 - Similar end-to-end times
 - STARTS had very few safety violations
- Method-level static RTS requires more work to be usable