# More Precise Regression Test Selection via Reasoning about Semantics-Modifying Changes

**Yuki Liu**, Jiyang Zhang, Pengyu Nie, Milos Gligoric, Owolabi Legunsen



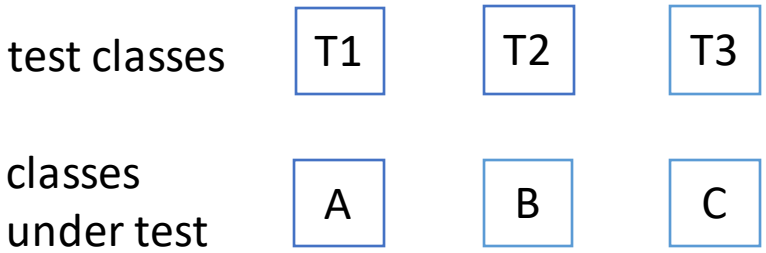
Cornell University



Google

# Regression testing

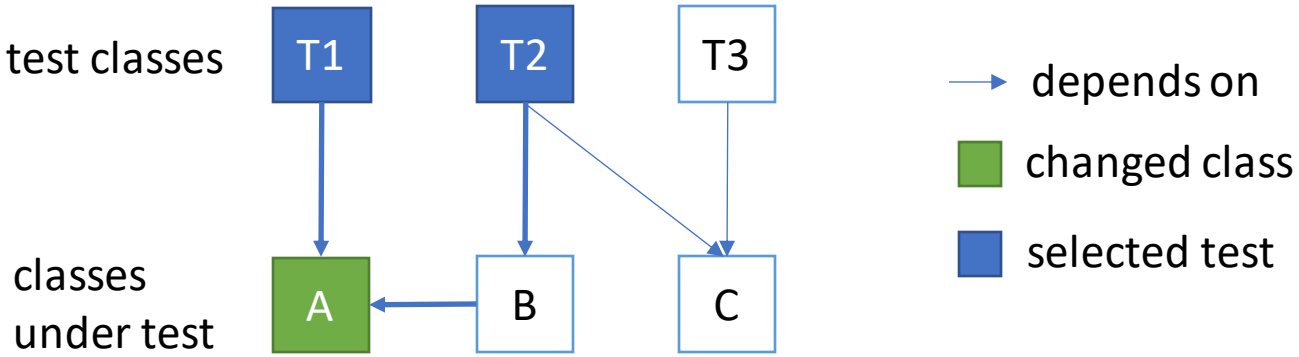
- Execute tests for each new code revision
- Check if the code changes break existing functionality




- Regression testing is costly
  - Google TAP handles 800k builds and runs 150 million tests per day
  - Microsoft's CloudBuild (used by >4k developers) handles 20k builds per day

# Regression test selection (RTS)

- Rerun only tests that are affected by changes
  - Safety: RTS selects all affected tests
  - Precision: RTS selects only affected tests
  - Goal: RTS runs fewer tests and runs tests faster than re-running all tests



# RTS tools that we improve: Ekstazi & STARTS

- Ekstazi[1]
  - dynamically tracks classes used while running tests
- STARTS[2]
  - statically builds a dependency graph
  - each class has an edge to direct parents and referenced classes

[1] Gligoric, Milos, Lamyaa Eloussi, and Darko Marinov. "Practical regression test selection with dynamic file dependencies." ISSTA 2015
[2] Legunsen, Owolabi, August Shi, and Darko Marinov. "STARTS: STAtic regression test selection." ASE 2017

# Motivation for this work

- Improve RTS precision without sacrificing safety

- Generalize related work like REKS[1], which improves RTS precision by skipping tests that are only affected by semantics-preserving changes

- Find semantics-modifying changes do not require re-running all tests that current RTS selects

[1] Wang, Kaiyuan, Chenguang Zhu, Ahmet Celik, Jongwook Kim, Don Batory, and Milos Gligoric. "Towards refactoring-aware regression test selection." ICSE 2018

# Leveraging semantics-modifying changes (removing throws clause example)

- The change only removes a throws clause from a method signature
- No other class uses reflection to invoke changed method
- Code still compiles
- Ekstazi and STARTS needlessly re-run 15 and 22 test classes

```
public class Percentile extends AbstractUnivariateStatistic {
-  public Percentile (final double quantile) throws MathException {
+  public Percentile (final double quantile) {
      ...
  }
}
```

Code is simplified from apache/commons-math

# Leveraging semantics-modifying changes (new method example)

- The change only adds a new method to a class
- No test class transitively depends on the newly added method
- The newly added method does not override another method
- Ekstazi and STARTS can skip 8 and 9 test classes

```
public abstract class Email {
+ public String getHeader(final String header) {
+   return headers.get(header);
+ }

  public void buildMimeMessage() { … }
}
```

Code is simplified from apache/commons-email

# How we found and leverage changes

5 projects


50 revisions each


Manually analyze changes (add, remove, modify) to constructors, fields, methods, classes, annotations…

13 findings where RTS may safely skip tests

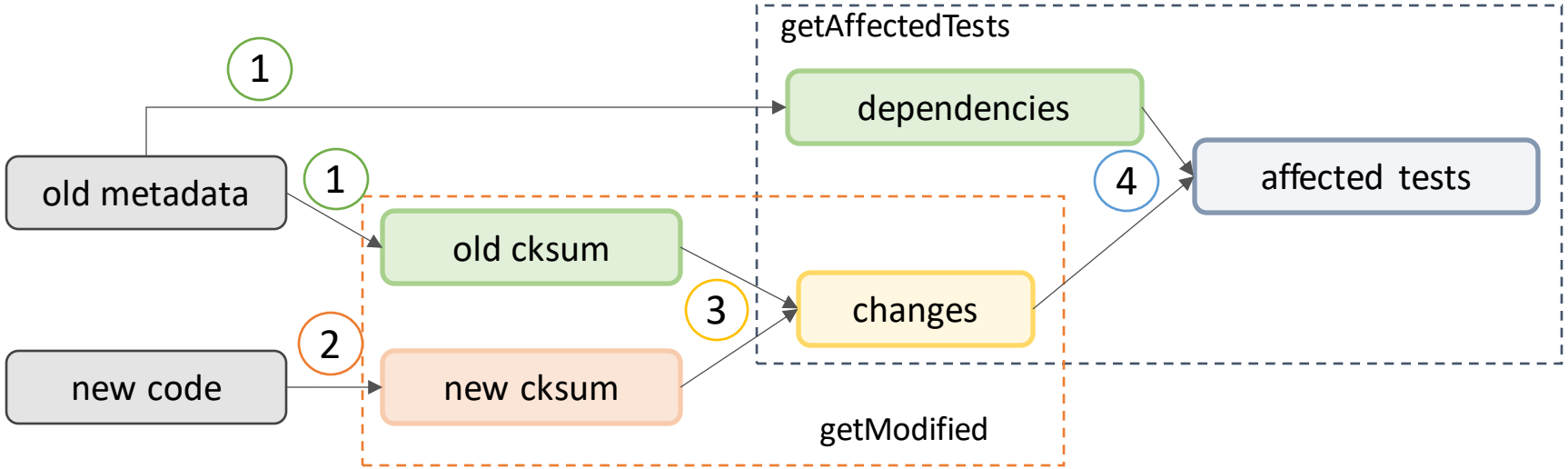
11 of them are semantics-modifying changes

2 are refactoring, sorting methods and renaming methods/classes

Apply findings to Ekstazi and STARTS
->
Implement FineEkstazi$^{F}$ and FineSTARTS$^{F}$

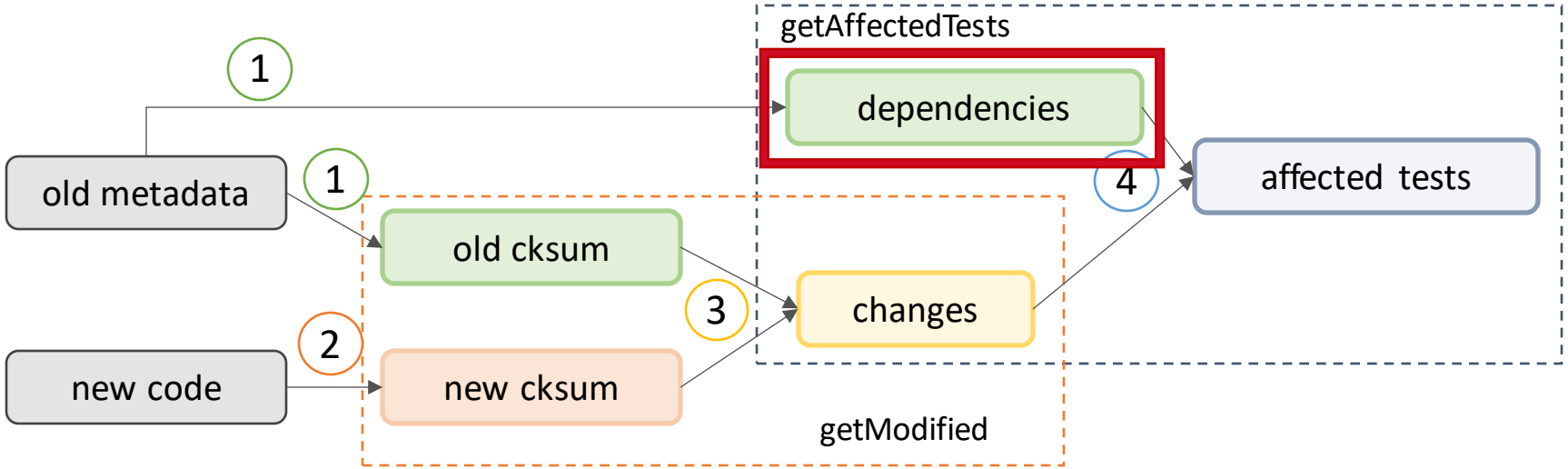
Hybrid dependency (class + method)
->
Implement FineEkstazi and FineSTARTS

# Overview of FineRTS$^{F}$




1. Load (field, constructor, method) metadata from running RTS on old revision
2. compute new checksum from current revision
3. compute changed classes using the old and new checksum
4. compute affected test classes where at least one dependency changed

# Overview of FineRTS

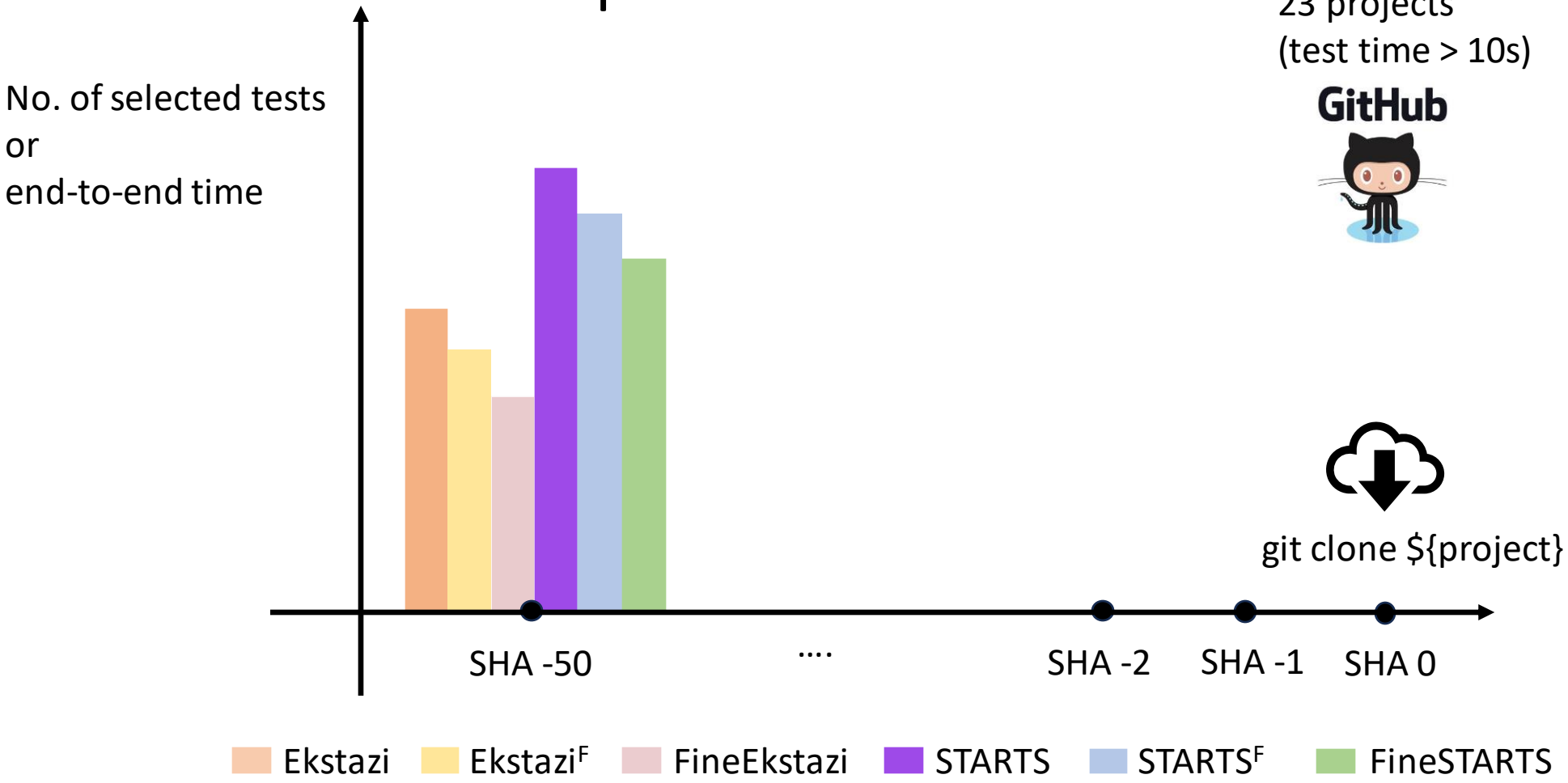



dependencies
class level -> hybrid of class, method, and field level

# Evaluation

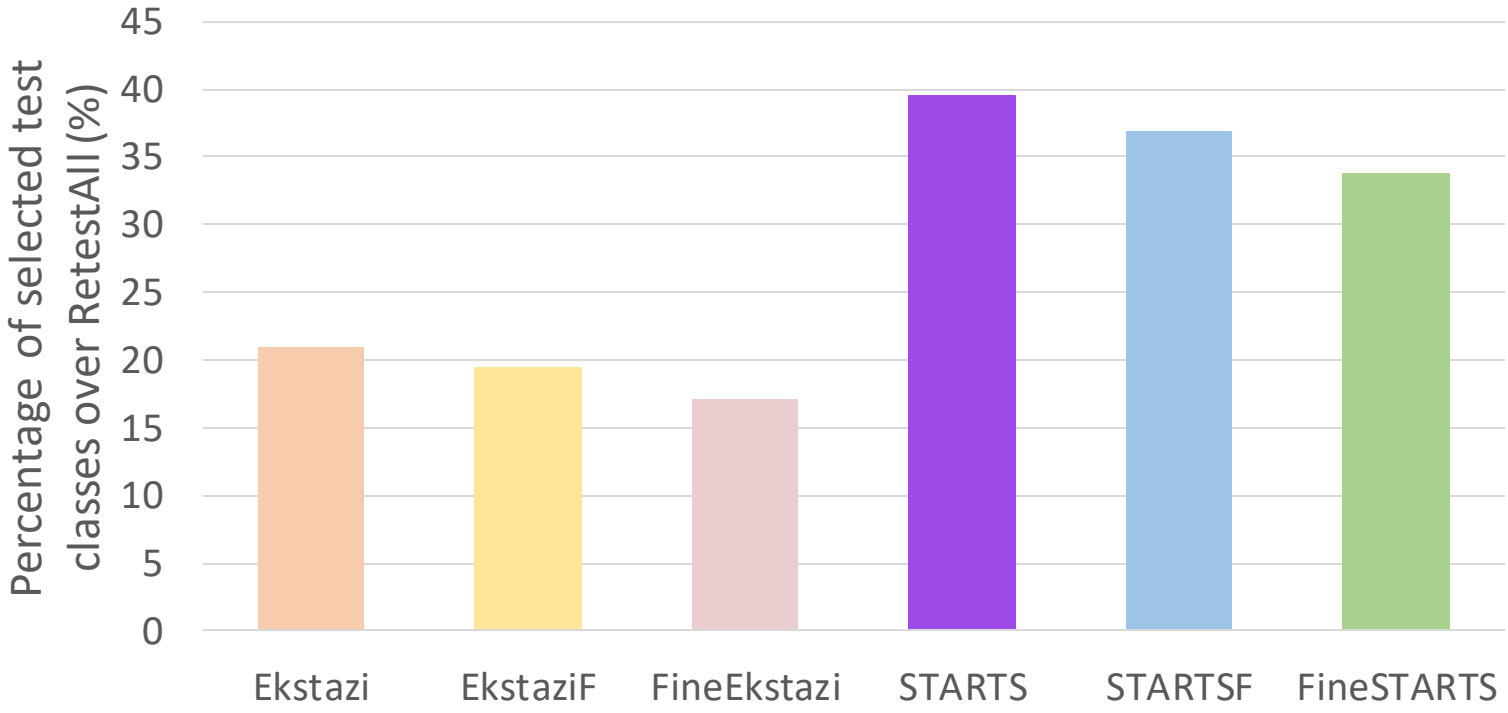
- RQ1: Impact on RTS selection rates
- RQ2: Impact on end-to-end time
- RQ3: Impact on safety
- RQ4: Spread of manual analysis findings

# Evaluation Setup



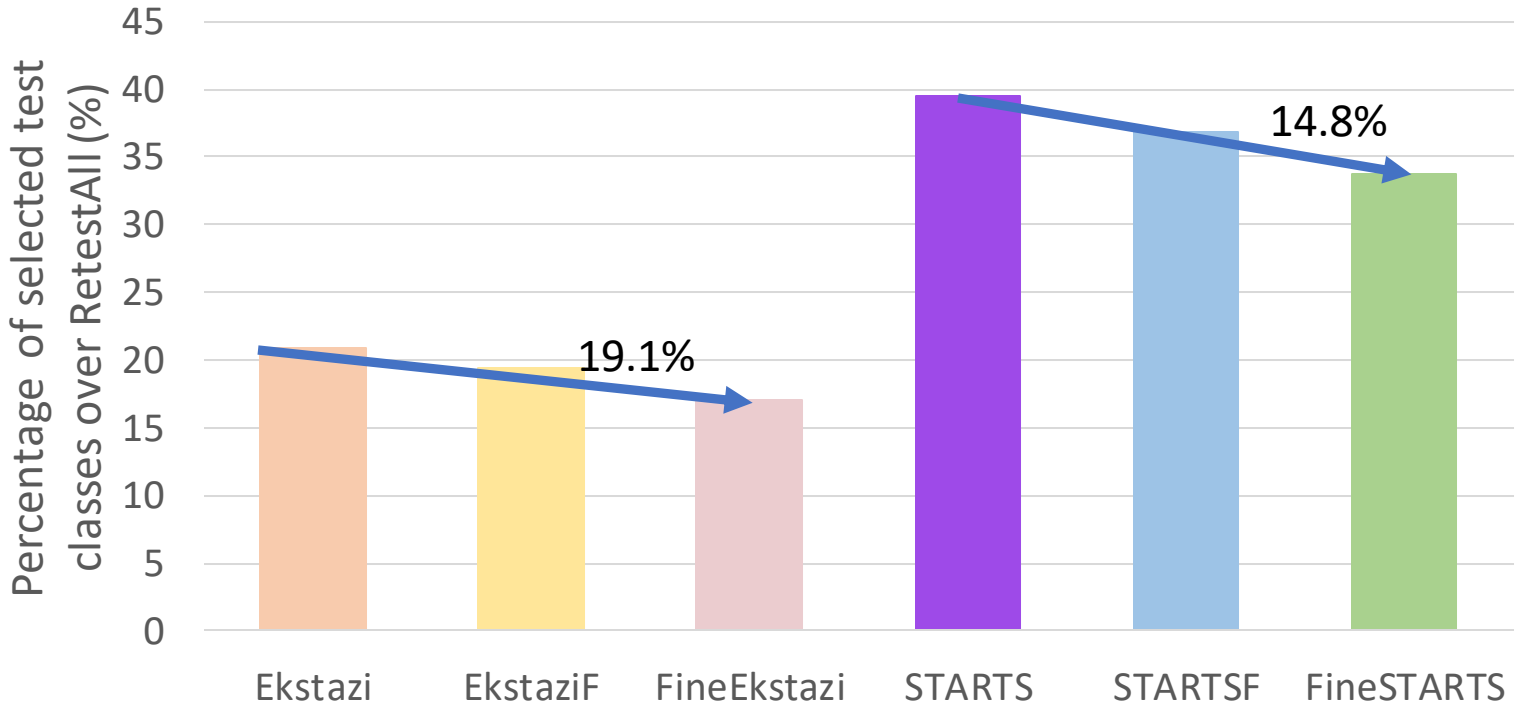

lower values on the y-axis are better

# RQ1: Reduction in number of selected tests

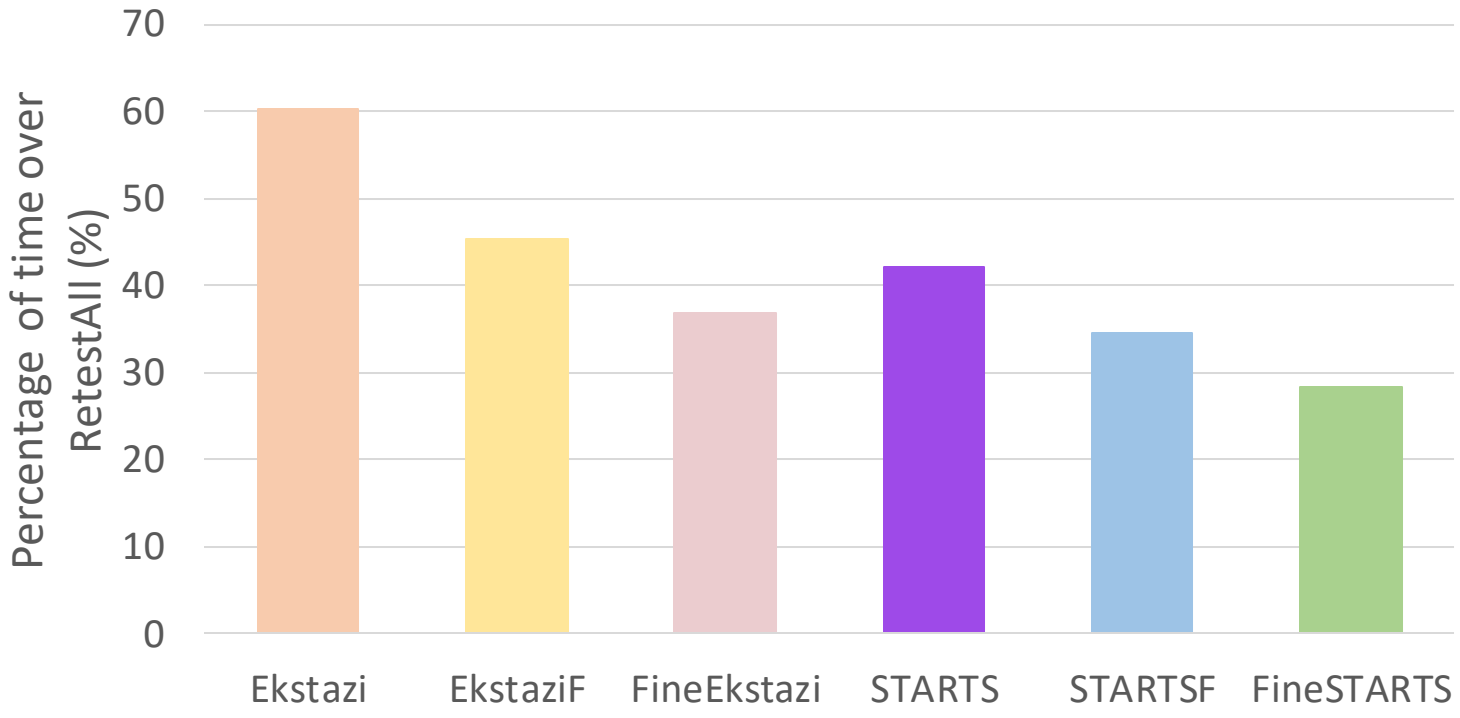


# RQ1: Reduction in number of selected tests

On average, FineEkstazi selects **19.1%** fewer tests than Ekstazi;
FineSTARTS selects **14.8%** fewer tests than STARTS.

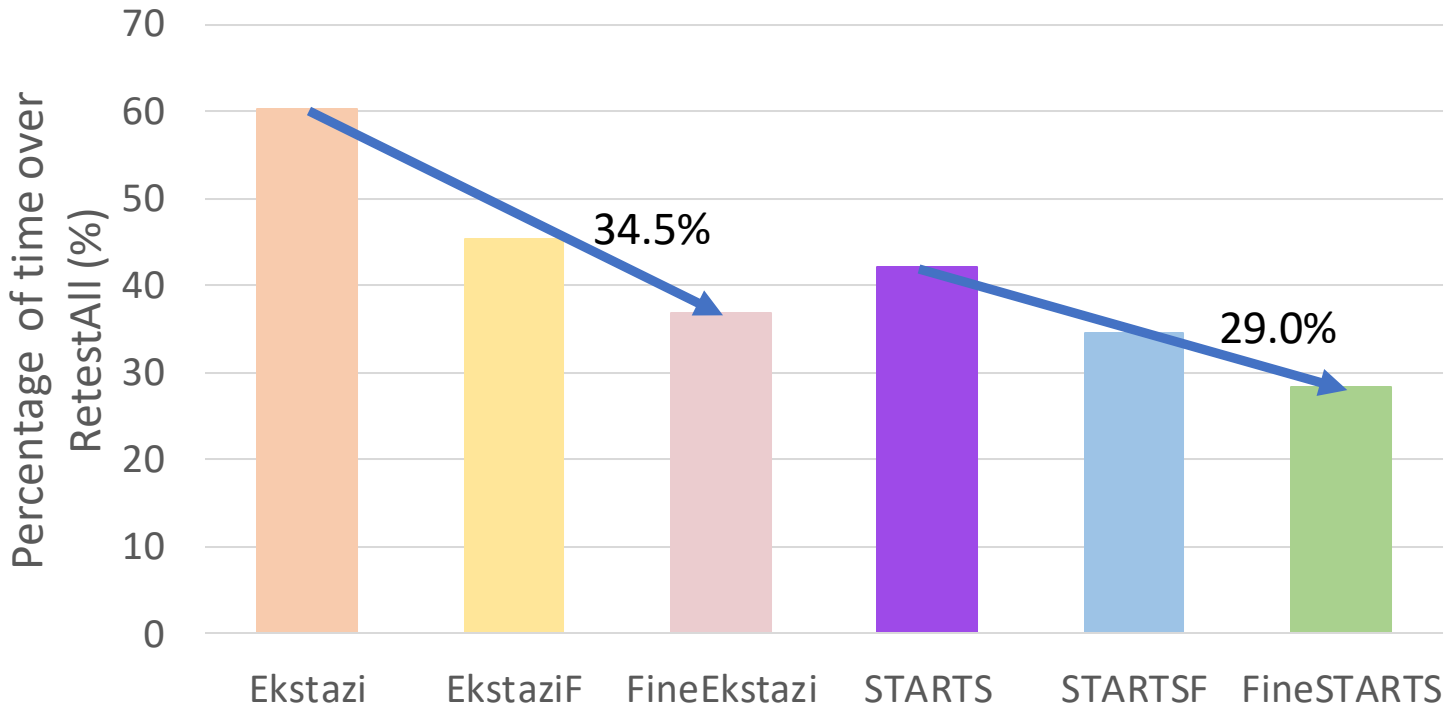


# RQ2: Reduction in end-to-end time



# RQ2: Reduction in end-to-end time

On average, FineEkstazi reduces the end-to-end time of Ekstazi by **34.5%**;
FineSTARTS reduces the end-to-end time of STARTS by **29.0%**.



# RQ3 (safety) and RQ4 (re-occurrence)

- RQ3: Impact on safety

  RTSCheck[1] did not find more violations in FineEkstazi and FineSTARTS compared to Ekstazi and STARTS. (More details in the paper.)

- RQ4: Re-occurence of manual analysis findings in other projects

  ~60% of revisions in projects that we did not manually analyze contain the kinds of findings that we leverage. (More details in the paper.)

[1] Zhu, Chenguang, Owolabi Legunsen, August Shi, and Milos Gligoric. "A framework for checking regression test selection tools." ICSE 2019

# Conclusion

- Goal: improve RTS precision without sacrificing safety
- Approach: find and leverage semantics-modifying changes
- Outcomes:
  - develop FineEkstazi and FineSTARTS
  - reduce selected tests by 19% (FineEkstazi), 15% (FineSTARTS)
  - reduce end-to-end time by 35% (FineEkstazi), 29% (FineSTARTS)
  - FineEkstazi and FineSTARTS are as safe as Ekstazi and STARTS

**https://github.com/EngineeringSoftware/FineRTS**

**yuki.liu@utexas.edu**

# Algorithm

**Algorithm 1** Embedding mRTS in FineEkstazi

**Require:** test $t$
**Require:** Ekstazi metadata *.ekstazi*
**Require:** mRTS metadata *.mrts*
**Ensure:** true if the test should run; false otherwise

```
1:  function Affected(t, .ekstazi, .mrts)
2:      cg ← FineEkstazi.GetModifiedClasses(t, .ekstazi)
3:      if cg = ∅ then                          ▷ Nothing is modified
4:          return false
5:      end if
6:      mg ← mRTS.GetModifiedClasses(t, .mrts)
7:      if cg ⊊ mg then                ▷ Reflection or third-party class
8:          return true
9:      end if
10:     for clz : cg do
11:         if mRTS.IsModified(t, clz, .mrts) then
12:             return true
13:         end if
14:     end for
15: return false
16: end function
```

# Algorithm

**Inputs:** $T$: a set of test classes, $M$: $t \rightarrow D$ ▷ Section 4.1 describes $D$
**Outputs:** $T^a \subseteq T$ : affected test classes

```
1:  procedure getAffectedTests(T, M)
2:    T^a ← {}
3:    for all test in T do
4:      for all C in M[test] do
5:        if getModified(test, C, M) then          ▷ test should be re-run
6:          T^a ← T^a ∪ {test}; break
7:    return T^a
9:  procedure getModified(test, C, M)
10:   M^new ← getNewMetadata(C)
11:   if M[test][C] == NULL then return true        ▷ C is a new dep
12:   else if M[test][C] == M^new then return false  ▷ C did not change
13:   else   ▷ did fields, constructors, initializers, or methods in C change?
14:     for all f in getFieldData(M[test][C]) do
15:       if fldChanged(M[test][C][f], M^new[f]) then return true
16:     for all c in getConstructorAndInitData(M[test][C]) do
17:       if conChanged(M[test][C][c], M^new[c]) then return true
18:     for all m in getMethodData(M[test][C]) do
19:       if mtdChanged(M[test][C][m], M^new[m], C) then return true
20:     return false
```