Ricardo Hernandez-Montoya,
ricardo.hernandez@rwth-aachen.de

Localizing Error-inducing Commits in CI Environments

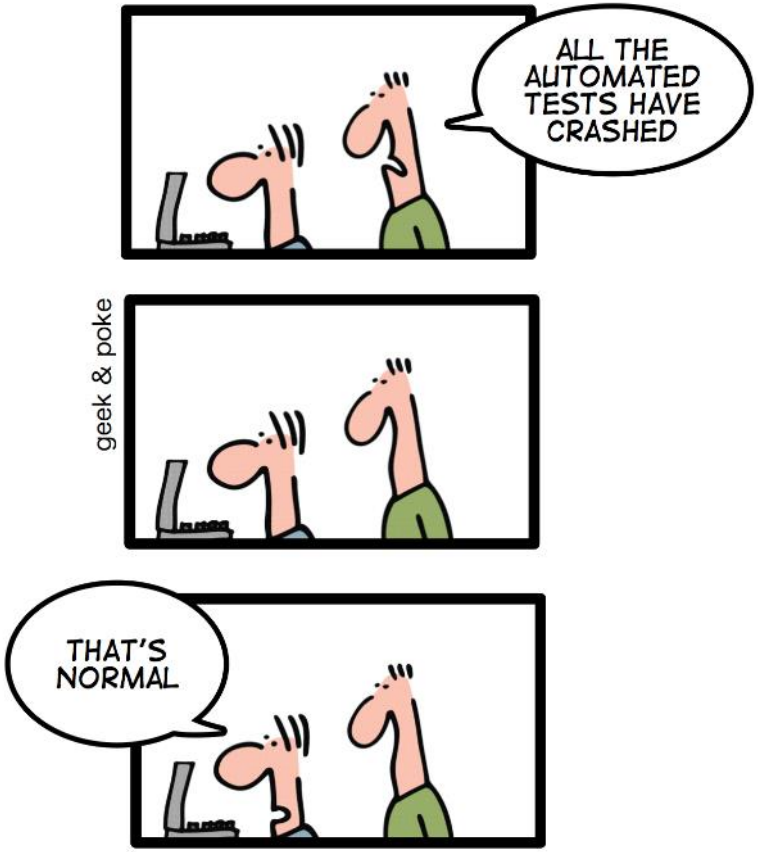
Master Thesis Final Presentation

April 26th, 2016

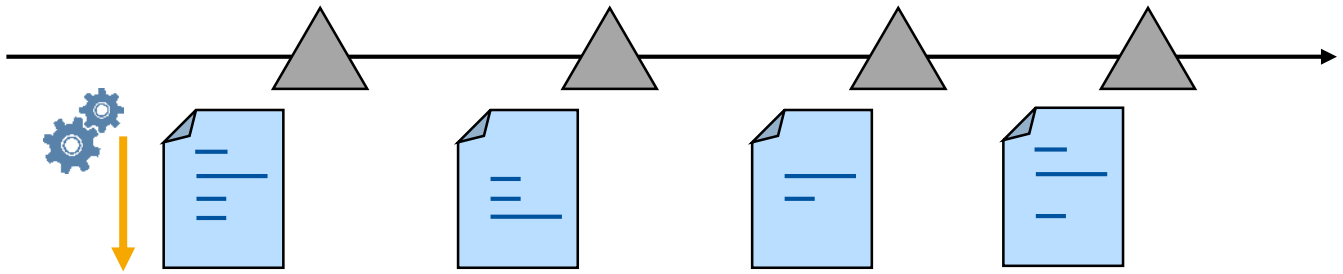
SWC Software Construction | RWTH AACHEN UNIVERSITY

GEEK & POKE'S LIST OF BEST PRACTICES

TODAY: CONTINUOUS INTEGRATION GIVES YOU THE COMFORTING FEELING TO KNOW THAT EVERYTHING IS NORMAL

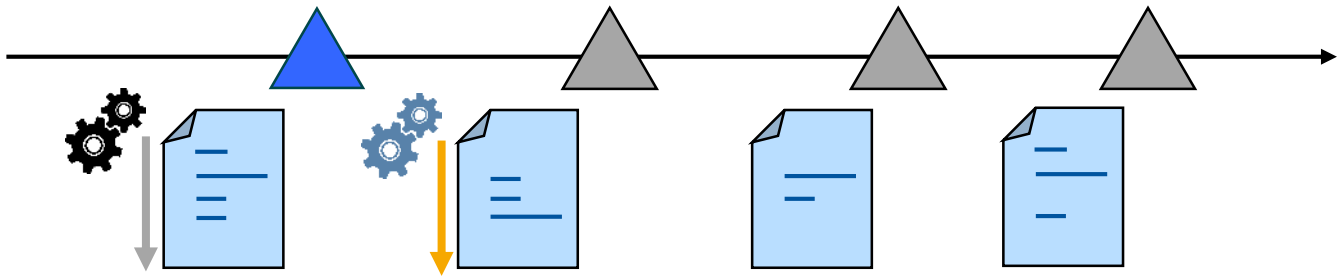


Continuous Integration (Ideal Scenario)



Healthy commit?

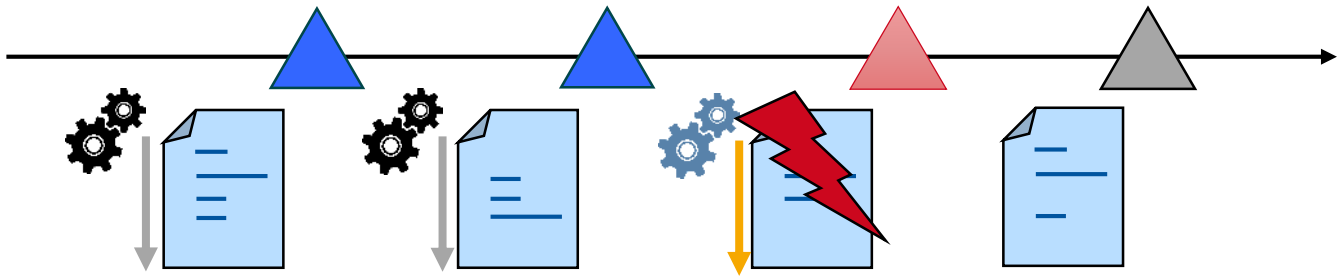
Continuous Integration (Ideal Scenario)



Healthy commit?

Healthy commit

Continuous Integration (Ideal Scenario)



Healthy commit?

Healthy commit

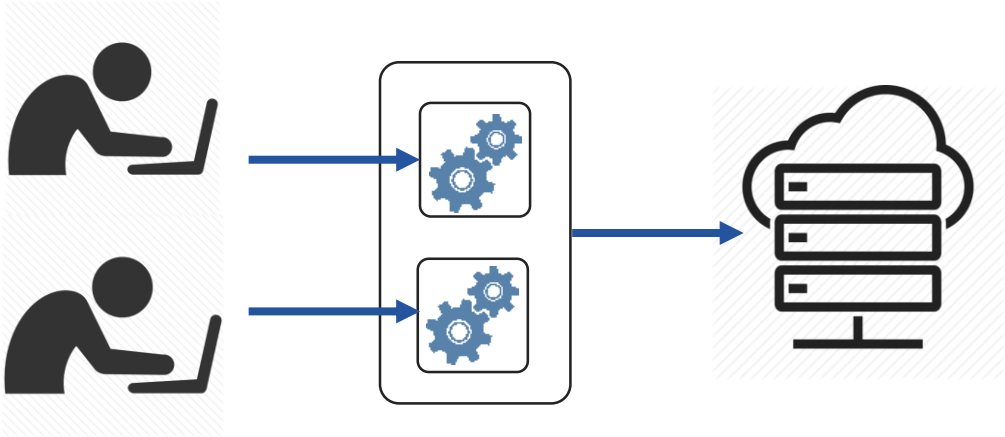
Broken build

Continuous Integration

How to avoid breaking the codebase?

Staged commit:

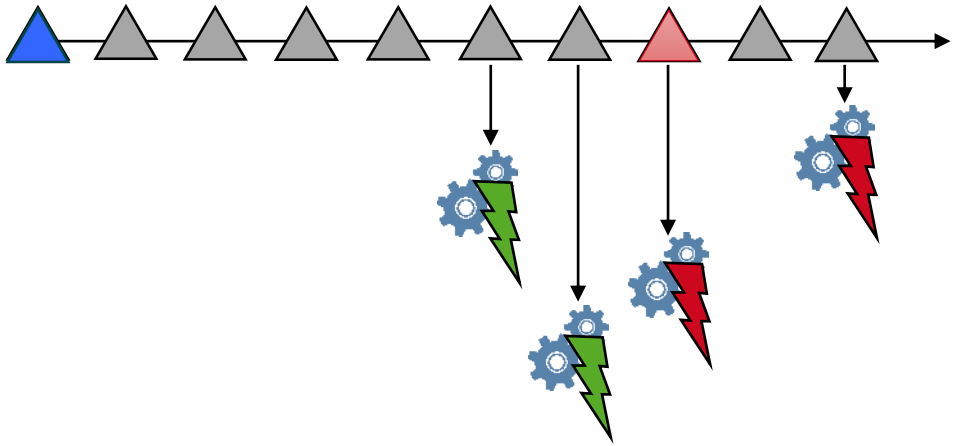
- Developers commit changes *blindly* to CI server.
- CI server determines if they can be integrated to the main line of work.
- Improve evaluation by testing changes in parallel.
- Example: OpenStack Zuul.



State of the Art

Identify failed commits using a **binary approach**:

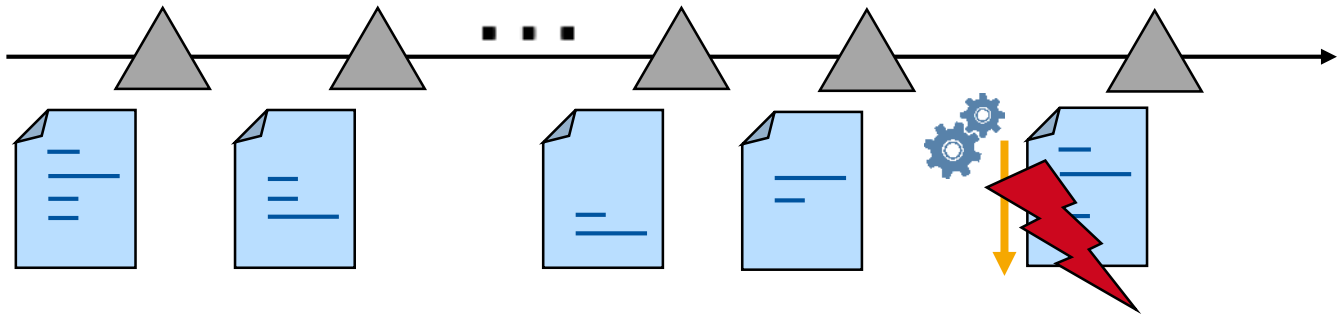
- Define initial evaluation commit.
- Visit each commit in the history on a binary basis.
- Evaluate **each** commit and mark them as good or bad.
- Automation possible with help of scripts.
- Examples: git-bisect, CI features (TeamCity), Facebook Sandcastle, Google Test tools.



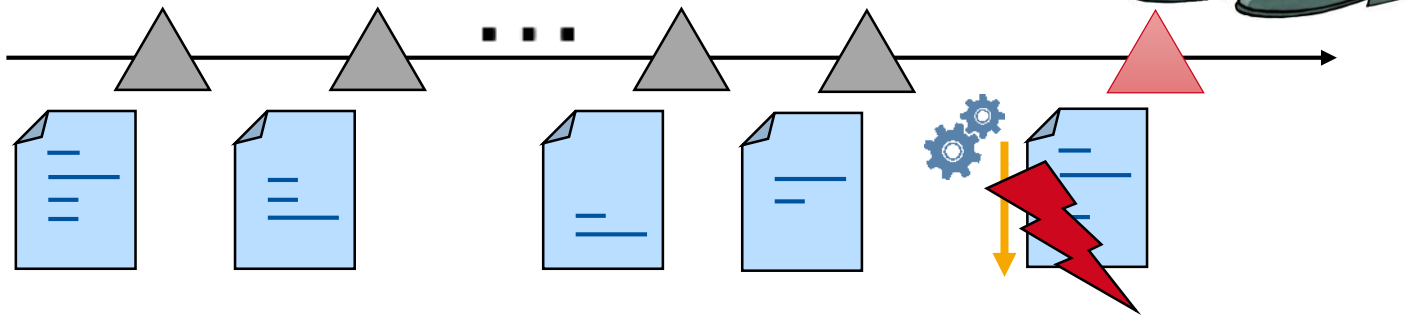
Motivation

- Fast-evolving software
  - Google example[1]:
    - 20+ changes per minute; 50% codebase changes every month.
    - 120K Test suites in the codebase.
    - 7.5M test suites run per day.
  - Facebook's VCS receives up to 100K+ commits per week.[2]

Continuous Integration (In Practice)



Continuous Integration (In Practice)

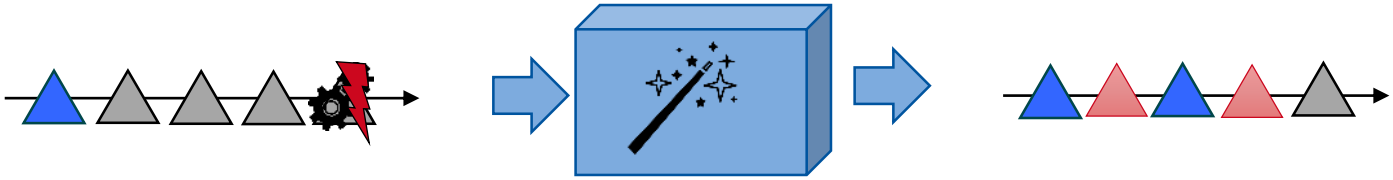


Healthy commit?

Broken build

Objective

Conceive strategies that help to identify possible *error-inducing commits* in the presence of failed tests.



Error-inducing Commits

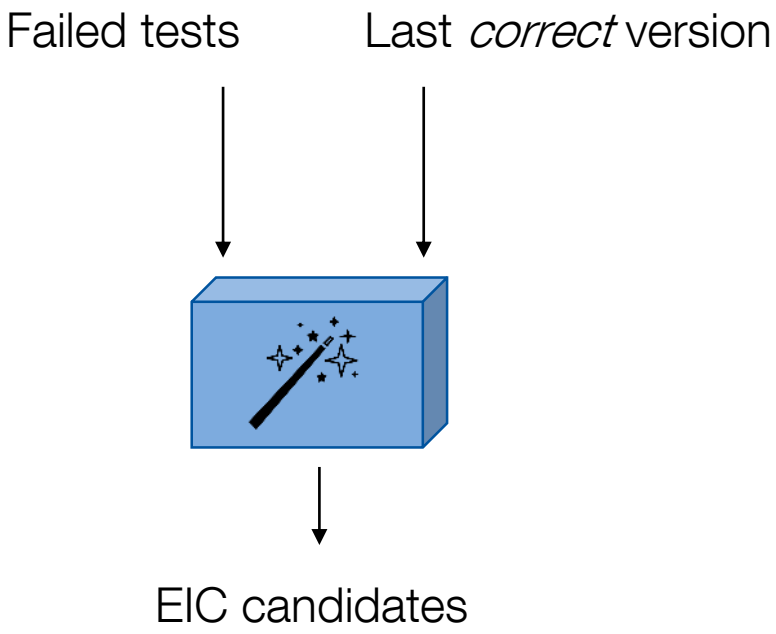
- Introduction of an error into a project's codebase.
- Evidenced by failed tests.
- Relies on the quality of the test suite.

Tracing EIC Concept

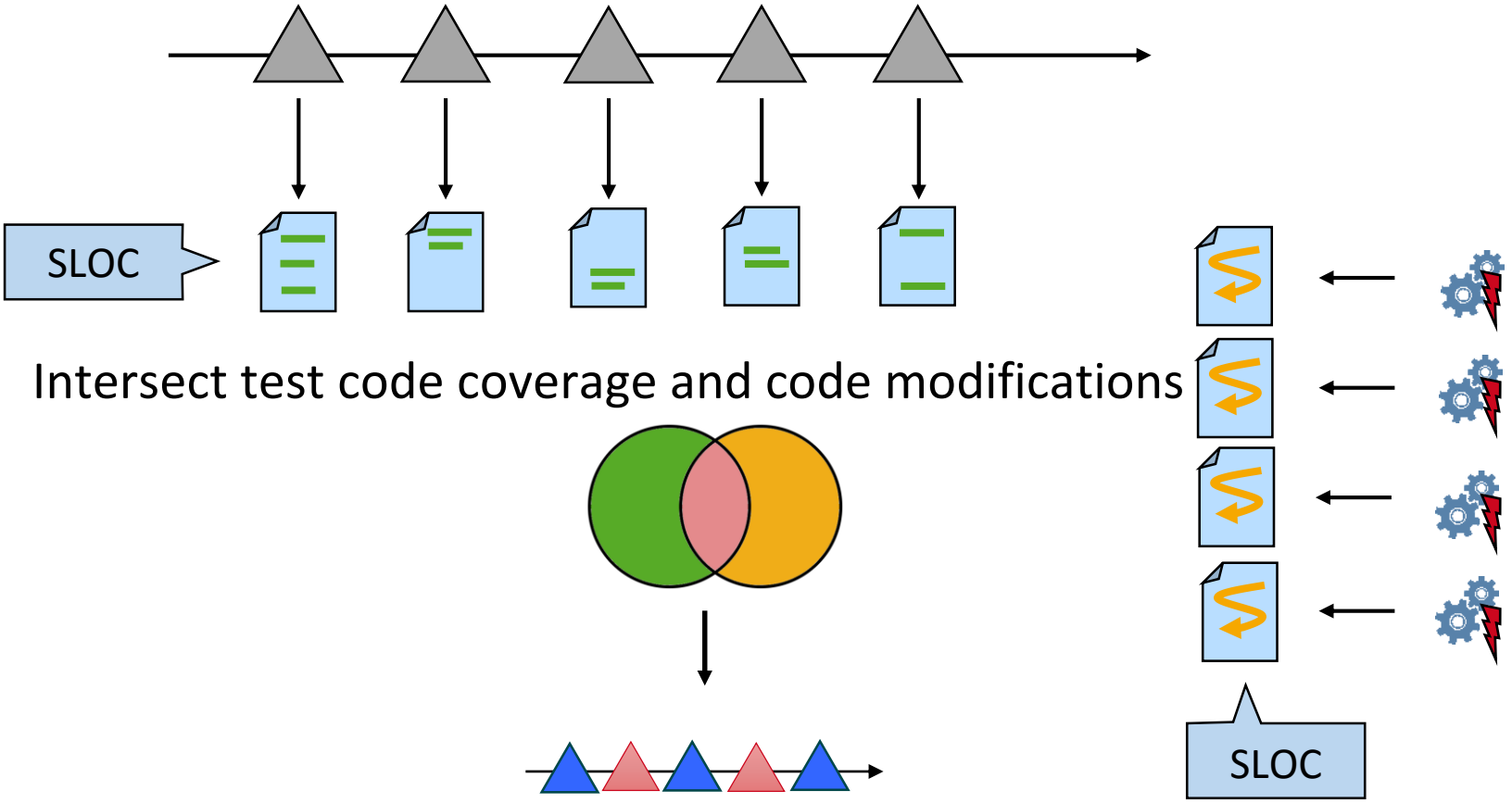
Background:

- Similar concepts (e.g., *bug-introducing change*, *fix-inducing change*) revealed properties around error introduction.
- Change impact identification techniques:
  - Identify program behavior affected by changes.
  - Used in regression testing to select and prioritize tests.
  - *Static* and *dynamic* techniques (test coverage, program slicing, call graphs, program dependency graphs, between others).

Tracing Strategies Framework



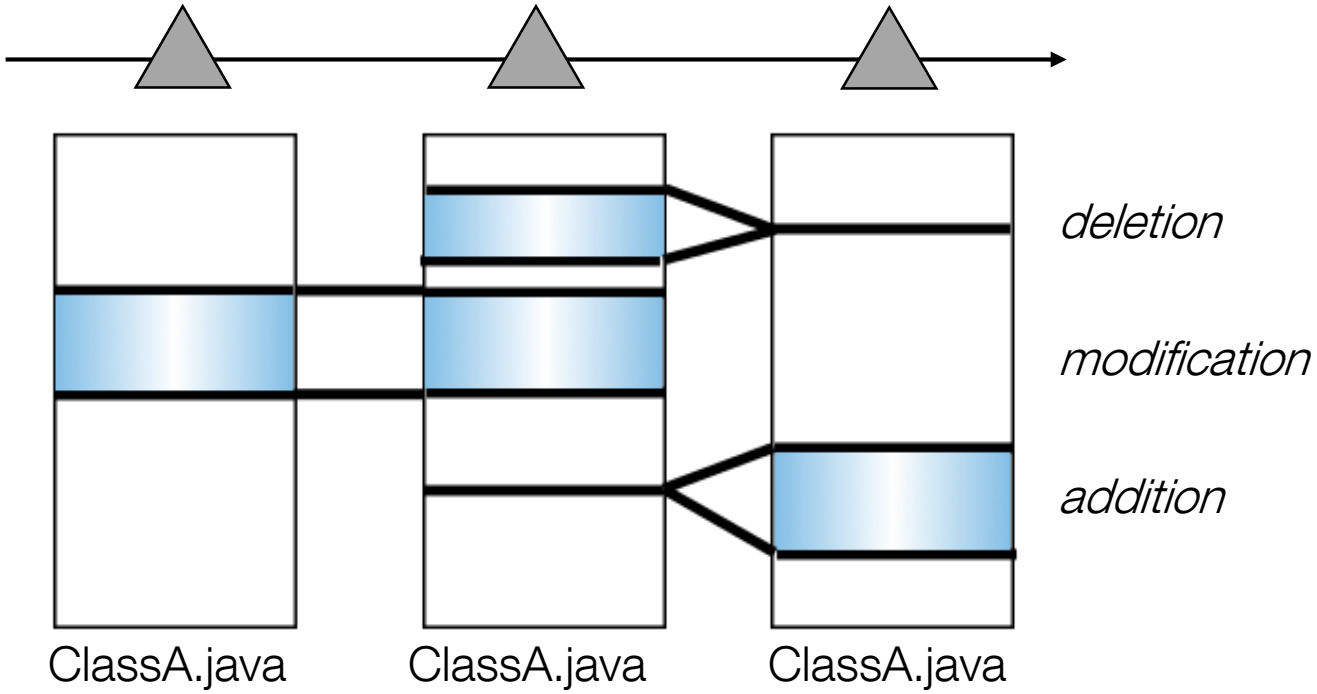
1st strategy: Statement-test-coverage



Yield possible error-inducing commits

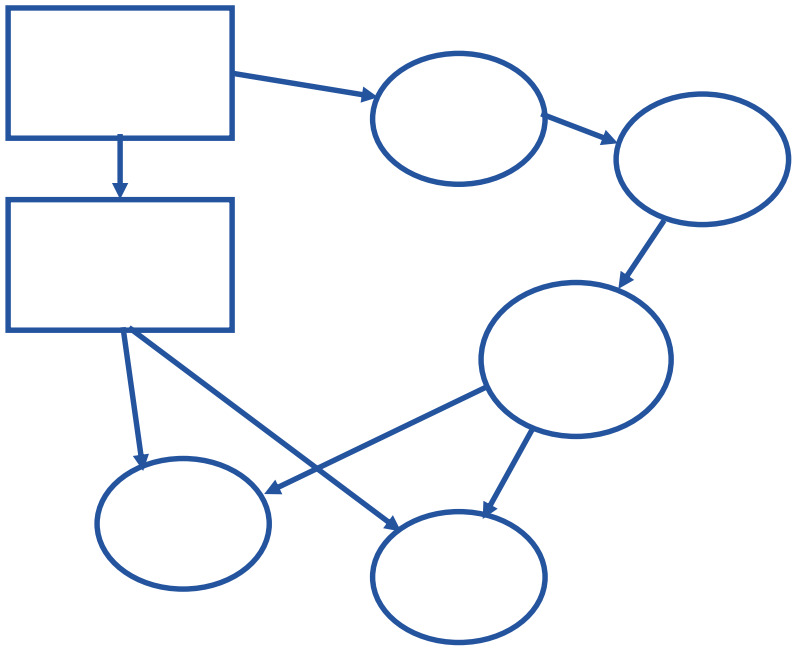
Challenge 1 + refinement

- Line count changes along the code evolution.
- Solution: *Delta Spots* to track the line count positions.



Challenge 2

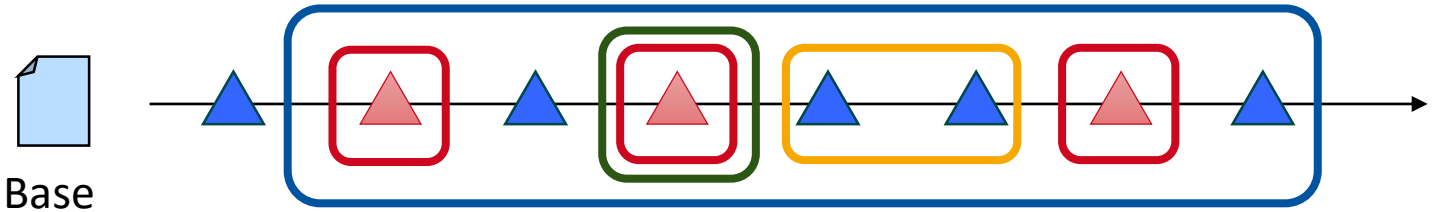
- Text comparison doesn’t recognize semantic changes.
- Solution: alternative techniques.



Properties schema

Properties of Strategies

- Safe strategy: Discovers all possible error-inducing commits.
- Precise strategy: Avoids all non-error-inducing commits.
- Reliable strategy: Delivers at least one possible error-inducing commit.



	Safe	Precise	Reliable
	X	X	X
	X	-	X
	-	X	X
	-	-	-

Error-inducing commit

Non-error-inducing commit

STC strategy properties

Safe ⟶ Changes hidden by deleted lines in history

Precise → No semantic data. Comments represent changes.

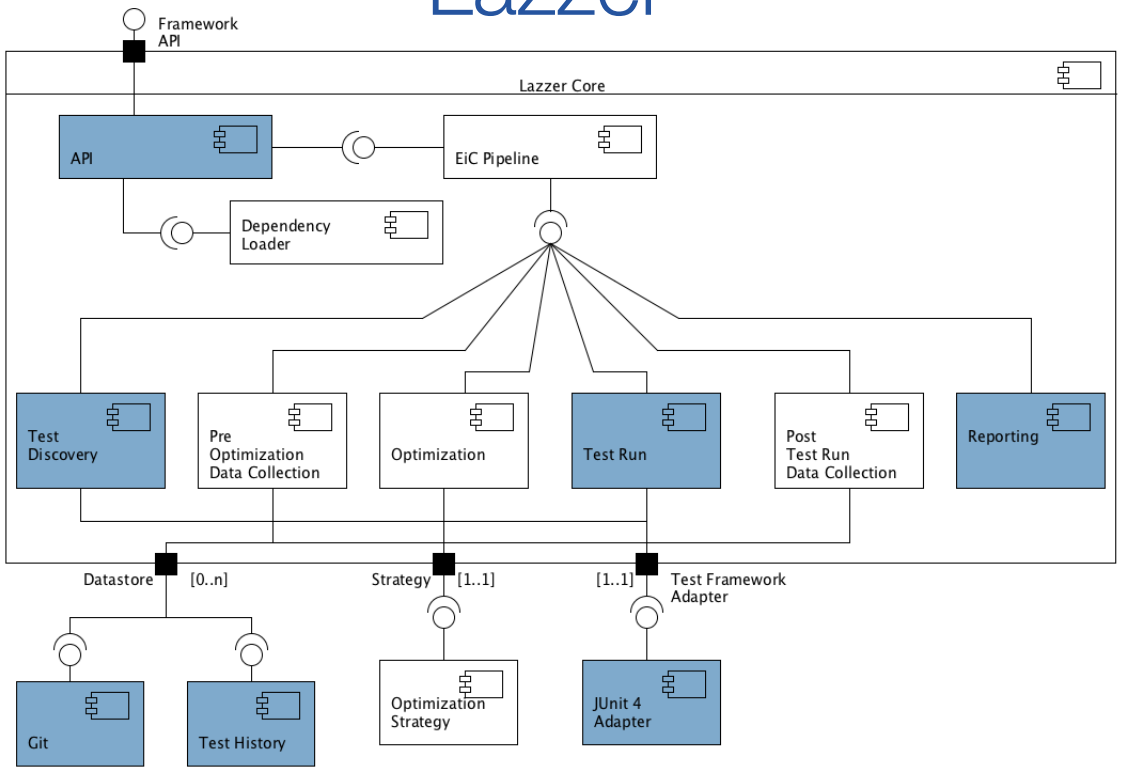
Reliable → No systematic approach enforced by strategy to discover EICs.

Realization

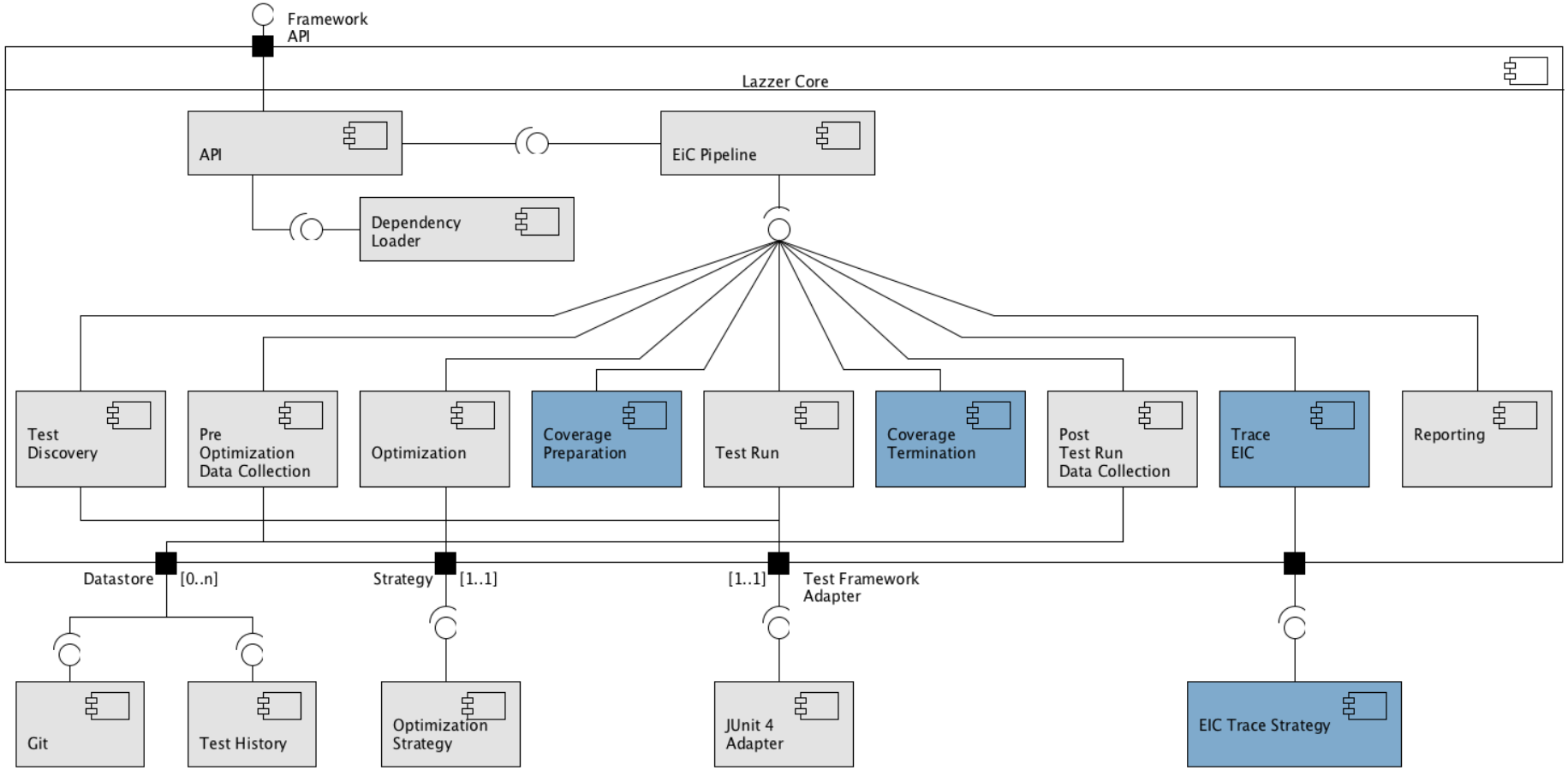
Realization requirements

- Test discovery
- Test execution (coverage gathering)
- Test result history
- Code versions access
- Reporting
- Build configuration

Lazzer

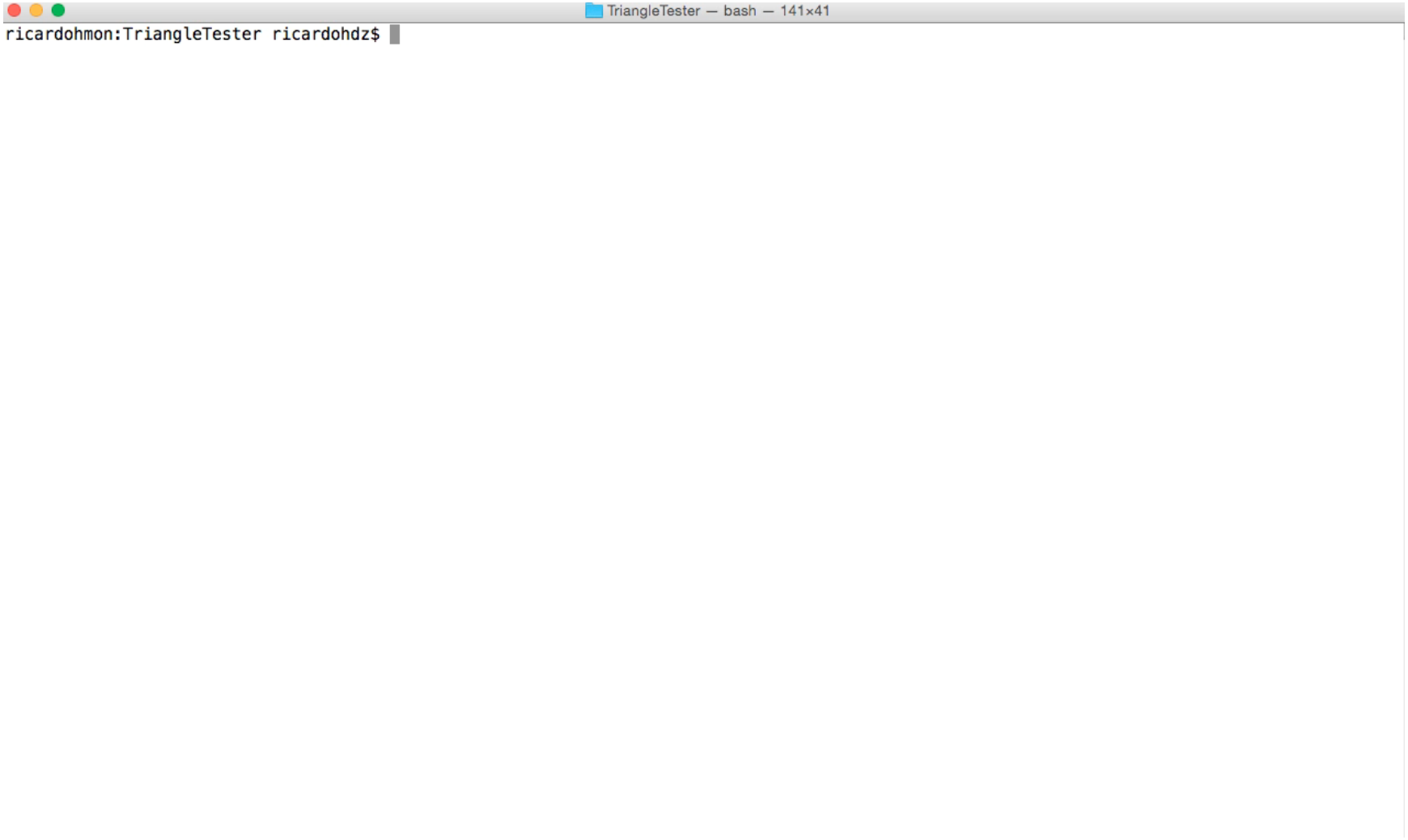


Lazzer framework extensions



Demo

Demo Case: Single line modification



Demo Case: Displaced error line

```
TriangleTester — bash — 141×41
[INFO] == Lazzer Results Report ============================================================
[INFO] ~~ Optimization Strategies ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[INFO] [1/1] AlphabeticPrioritisation
[INFO]
[INFO] ~~ Tracing Strategies ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[INFO] [1/1] StatementTestCoverage
[INFO]
[INFO] ~~ Data Stores ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[INFO] [1/1] TestHistoryDataStore
[INFO]
[INFO] ~~ Tests ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[INFO] [ 1/13] BoundaryValueTest.minPNomNom ..... SUCCESS .... 1 ms
[INFO] [ 2/13] BoundaryValueTest.nomMaxNom ..... SUCCESS .... 0 ms
[INFO] [ 3/13] BoundaryValueTest.nomMinNom ..... SUCCESS .... 0 ms
[INFO] [ 4/13] BoundaryValueTest.nomNomMax ..... SUCCESS .... 0 ms
[INFO] [ 5/13] BoundaryValueTest.nomNomMin ..... SUCCESS .... 0 ms
[INFO] [ 6/13] BoundaryValueTest.nomNomNom ..... SUCCESS .... 0 ms
[INFO] [ 7/13] BoundaryValueTest.maxNomNom ..... SUCCESS .... 1 ms
[INFO] [ 8/13] BoundaryValueTest.minNomNom ..... SUCCESS .... 0 ms
[INFO] [ 9/13] BoundaryValueTest.nomMaxMNom ..... SUCCESS .... 0 ms
[INFO] [10/13] BoundaryValueTest.nomMinPNom ..... SUCCESS .... 0 ms
[INFO] [11/13] BoundaryValueTest.maxMNomNom ..... SUCCESS .... 0 ms
[INFO] [12/13] BoundaryValueTest.nomNomMaxM ..... SUCCESS .... 0 ms
[INFO] [13/13] BoundaryValueTest.nomNomMinP ..... SUCCESS .... 1 ms
[INFO]
[INFO] ~~ Test statistics ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[INFO] Tests succeeded: 13        Tests failed: 0           Tests ignored: 0
[INFO]
[INFO]
[INFO] ------------------------------------------------------------------------------
[INFO] Result: SUCCESS
[INFO] ------------------------------------------------------------------------------
[INFO] HHH000030: Cleaning up connection pool [jdbc:h2:~/lazzer-h2-db;MODE=PostgreSQL]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5.428 s
[INFO] Finished at: 2016-04-26T00:55:59+02:00
[INFO] Final Memory: 37M/451M
[INFO] ------------------------------------------------------------------------
ricardohmon:TriangleTester ricardohdz$
```

Demo Case: Method extraction refactoring

```
TriangleTester — bash — 141×41
[INFO] == Lazzer Results Report ============================================================
[INFO] ~~ Optimization Strategies ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[INFO] [1/1] AlphabeticPrioritisation
[INFO]
[INFO] ~~ Tracing Strategies ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[INFO] [1/1] StatementTestCoverage
[INFO]
[INFO] ~~ Data Stores ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[INFO] [1/1] TestHistoryDataStore
[INFO]
[INFO] ~~ Tests ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[INFO] [ 1/13] BoundaryValueTest.minPNomNom ..... SUCCESS .... 1 ms
[INFO] [ 2/13] BoundaryValueTest.nomMaxNom ..... SUCCESS .... 0 ms
[INFO] [ 3/13] BoundaryValueTest.nomMinNom ..... SUCCESS .... 0 ms
[INFO] [ 4/13] BoundaryValueTest.nomNomMax ..... SUCCESS .... 1 ms
[INFO] [ 5/13] BoundaryValueTest.nomNomMin ..... SUCCESS .... 0 ms
[INFO] [ 6/13] BoundaryValueTest.nomNomNom ..... SUCCESS .... 0 ms
[INFO] [ 7/13] BoundaryValueTest.maxNomNom ..... SUCCESS .... 0 ms
[INFO] [ 8/13] BoundaryValueTest.minNomNom ..... SUCCESS .... 0 ms
[INFO] [ 9/13] BoundaryValueTest.nomMaxMNom ..... SUCCESS .... 0 ms
[INFO] [10/13] BoundaryValueTest.nomMinPNom ..... SUCCESS .... 0 ms
[INFO] [11/13] BoundaryValueTest.maxMNomNom ..... SUCCESS .... 0 ms
[INFO] [12/13] BoundaryValueTest.nomNomMaxM ..... SUCCESS .... 0 ms
[INFO] [13/13] BoundaryValueTest.nomNomMinP ..... SUCCESS .... 0 ms
[INFO]
[INFO] ~~ Test statistics ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[INFO] Tests succeeded: 13        Tests failed: 0           Tests ignored: 0
[INFO]
[INFO]
[INFO] ====================================================================================
[INFO] Result: SUCCESS
[INFO] ====================================================================================
[INFO] HHH000030: Cleaning up connection pool [jdbc:h2:~/lazzer-h2-db;MODE=PostgreSQL]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5.431 s
[INFO] Finished at: 2016-04-26T01:01:27+02:00
[INFO] Final Memory: 37M/456M
[INFO] ------------------------------------------------------------------------
ricardohmon:TriangleTester ricardohdz$
```

EIC

Summary

Contributions:

- Concept of Error-inducing Commits in CI.
- Three strategies for localizing EICs.
- Properties schema for categorizing tracing strategies.
- Realization of one strategy.

Conclusion

- Pioneering concept of EICs in CI.
- Realized strategy not very strong but gives initial hints, despite reduced computing overhead.
- Alternative promising techniques for localizing EICs.
- Relevance in the industry.

Further Work

- Strategies realization:
  - Further implementation of tracing strategies.
- Conceived techniques:
  - Based on statement-test-coverage.
  - Based on program slicing.
  - Based on atomic changes.
  - Re-run failed tests on all commits.
- Strategies evaluation. Using real-world projects.
- Strategy conception:
  - Explore further techniques (e.g., symbolic execution).

References

- [1] Kumar, Ashish. *Development at the speed and scale of google.* (2010).
- [2] Facebook Developers. *F8 2015 - Big Code: Developer Infrastructure At Facebook's Scale.* (2015). Available at: https://www.youtube.com/watch?v=X0VH78ye4yY. Accessed April 25, 2016.

Thank you

2nd strategy: Based on program slicing

Forward program slicing

```
1  input (i , j);
2  a = i;
3  b = 0;
4  if (a > 0)
5     b = a
6  if (b > 0)
7     o = j
8  output(o)
```

2nd strategy: Based on program slicing

Forward program slicing

```
1  input (i , j);
2  a = i + j;
3  b = 0;
4  if (a > 0)
5     b = a
6  if (b > 0)
7     o = j
8  output(o)
```

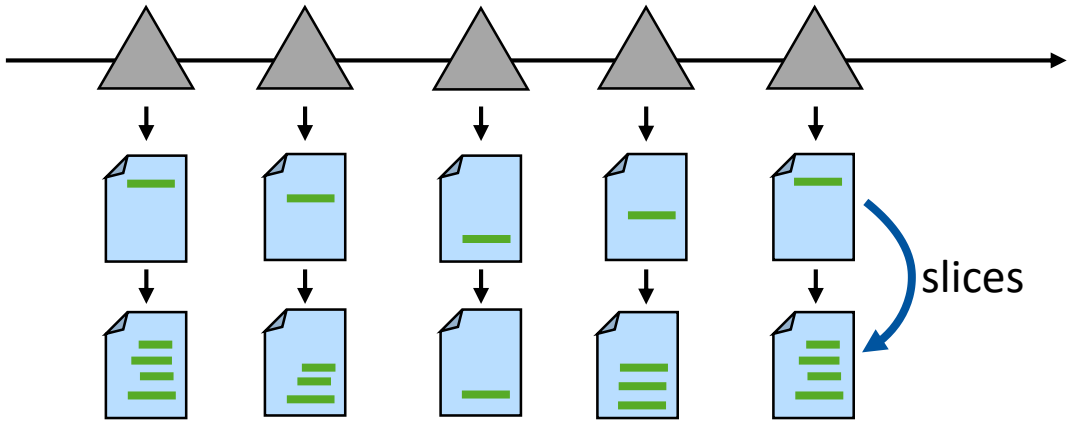
2nd strategy: Based on program slicing

Forward program slicing

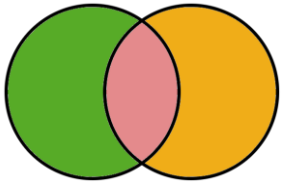
```
1  input (i , j);
2  a = i + j;
3  b = 0;
4  if (a > 0)
5     b = a
6  if (b > 0)
7     o = j
8  output(o)
```

Forward slice on modified line 2
4, 5, 6, 7, 8

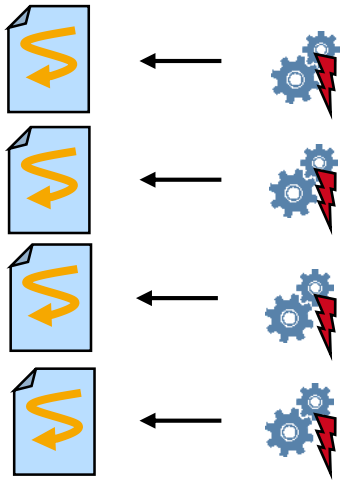
2nd strategy: Based on program slicing



Identify slices covered by failed tests



Yield possible error-inducing commits

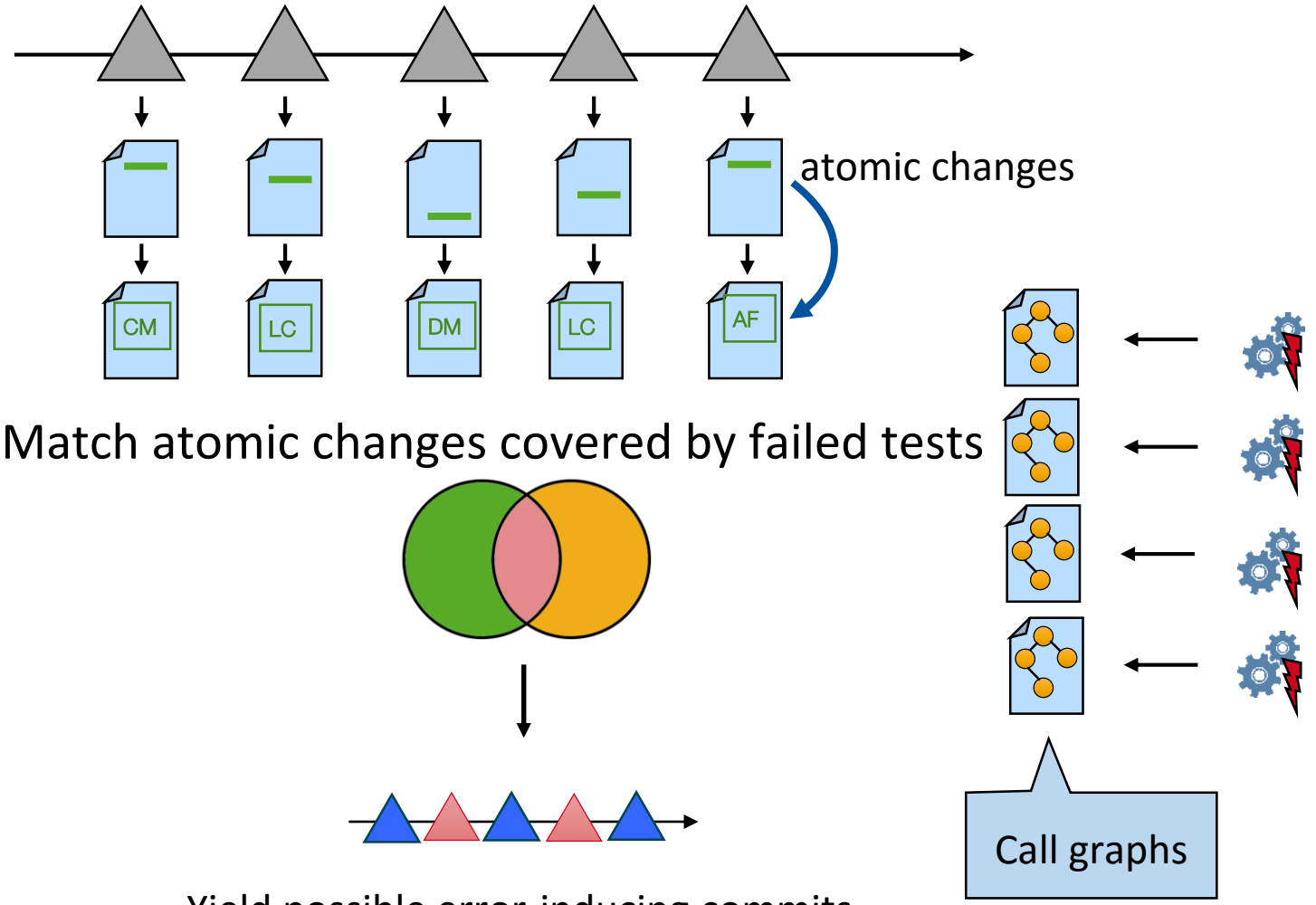


Statement level

3rd strategy: Based on atomic changes

- Classification of the code changes into atomic changes. Possible types:
  - Added Method (**AM**)
  - Changed Method (**CM**)
  - Deleted Method (**DM**)
  - Added Field (**AF**)
  - Deleted Field (**DF**)
  - Lookup Change (**LC**)
- Obtain semantic difference from the atomic changes.
- Identify affected test methods by the code changes.

3rd strategy: Based on atomic changes



4th strategy: Re-run failed tests on all commits

- Naïve approach.
- Rerun failed tests on all commits and stop until all failed tests responsible commits are found.
- Make sure to deliver at least one EIC with certainty.