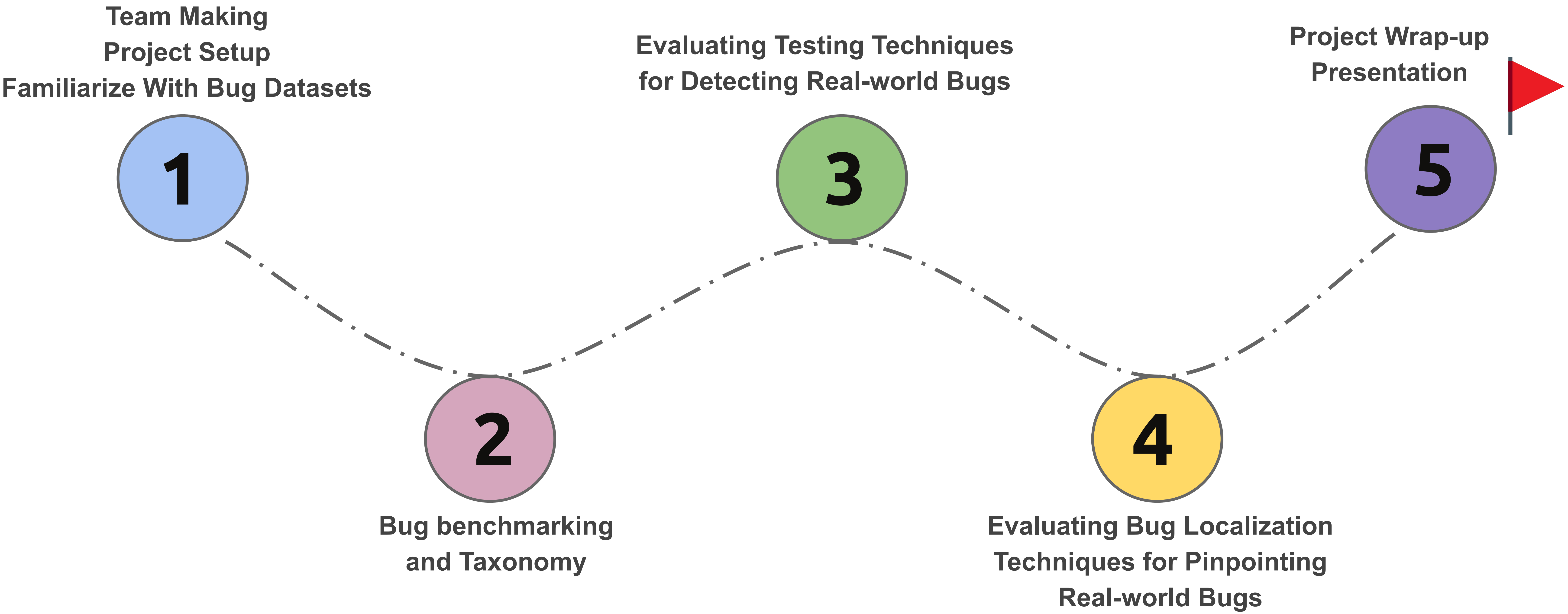
UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN

Evaluating Testing And Debugging Tools On Real-World Bugs

CS 527 ADVANCED TOPICS IN SOFTWARE ENGINEERING

TEAM 9
Andres Suazo (suazo2)
Shrushti Jagtap (sjagtap3)
Sejal Pekam (spekam2)
Shreya Chaudhary (suc2)



Team Making
Project Setup
Familiarize With Bug Datasets

Bug benchmarking
and Taxonomy

Evaluating Testing Techniques
for Detecting Real-world Bugs

Evaluating Bug Localization
Techniques for Pinpointing
Real-world Bugs

Project Wrap-up
Presentation

Milestone 1: Exploration

- Familiarized with the five bug repositories (Bears, BugSwarm, Defects4j, QuixBugs, ManySStuBs4J)
  - Number of bugs
  - Number of tests
- Collected additional information for five bugs per bug repository
  - Buggy version
  - Patched version

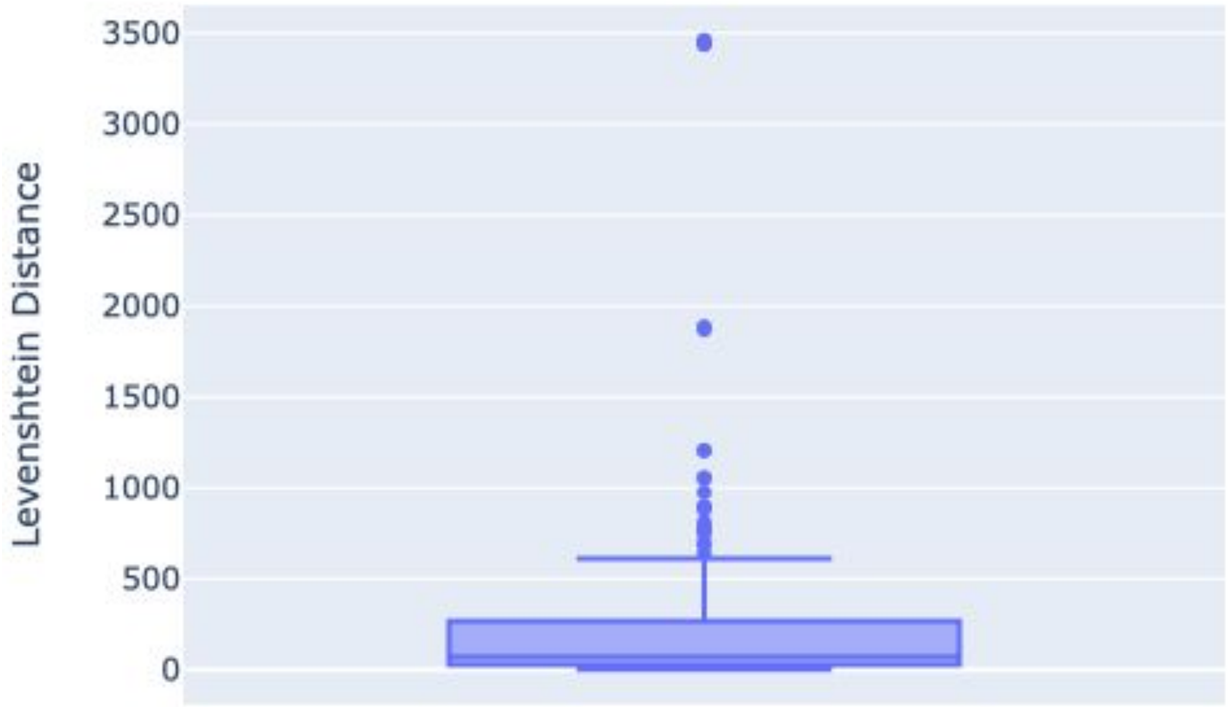
DATASET	NUMBER OF BUGS
Bears	118
Bugswarm	223
Defects4j	835 (+29)
Quixbugs	537
ManySStuBs4J	99369

Milestone 2: Benchmarking & Taxonomy

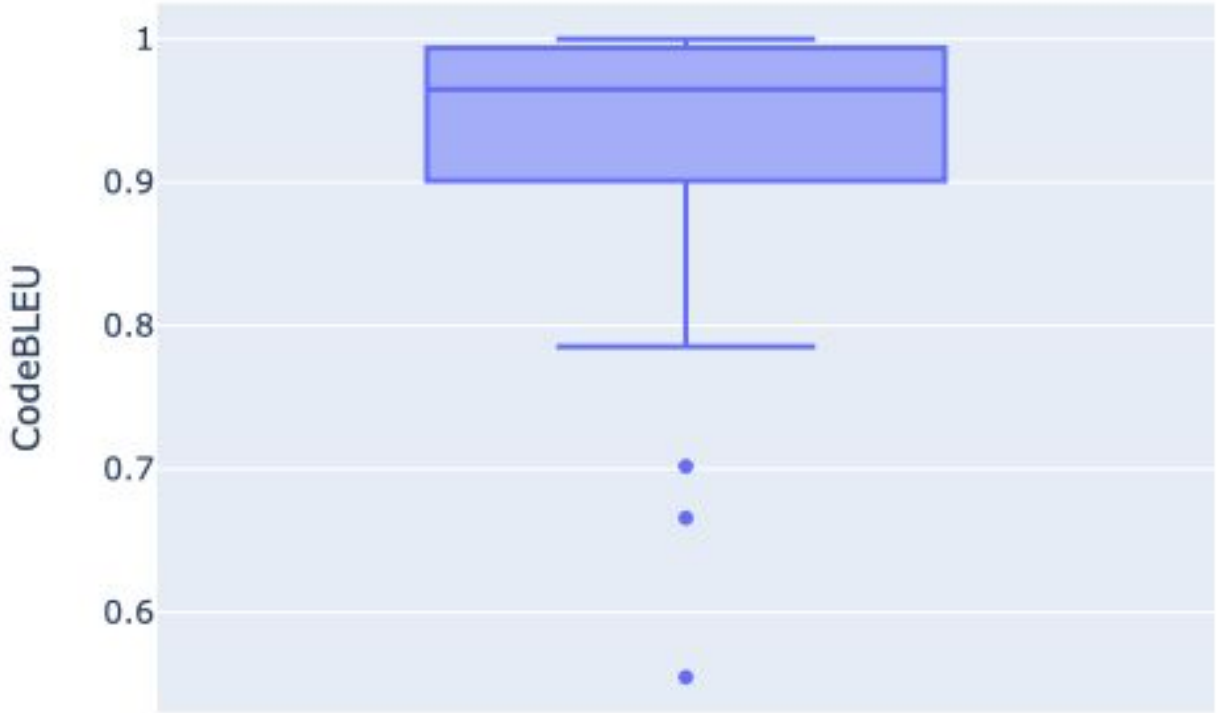
- Extracted the following from the bug repositories for 20 bugs from Bears, BugSwarm, QuixBugs each and 68 bugs from Defects4J
  - Buggy Version
  - Patched Version
  - Diff
  - List of Failed Tests
    - ran tests using maven, gradle, junit
- Benchmarking bugs

Milestone 2: Benchmarking Metrics

- Levenshtein Distance
  - Lower value -> More Similarity



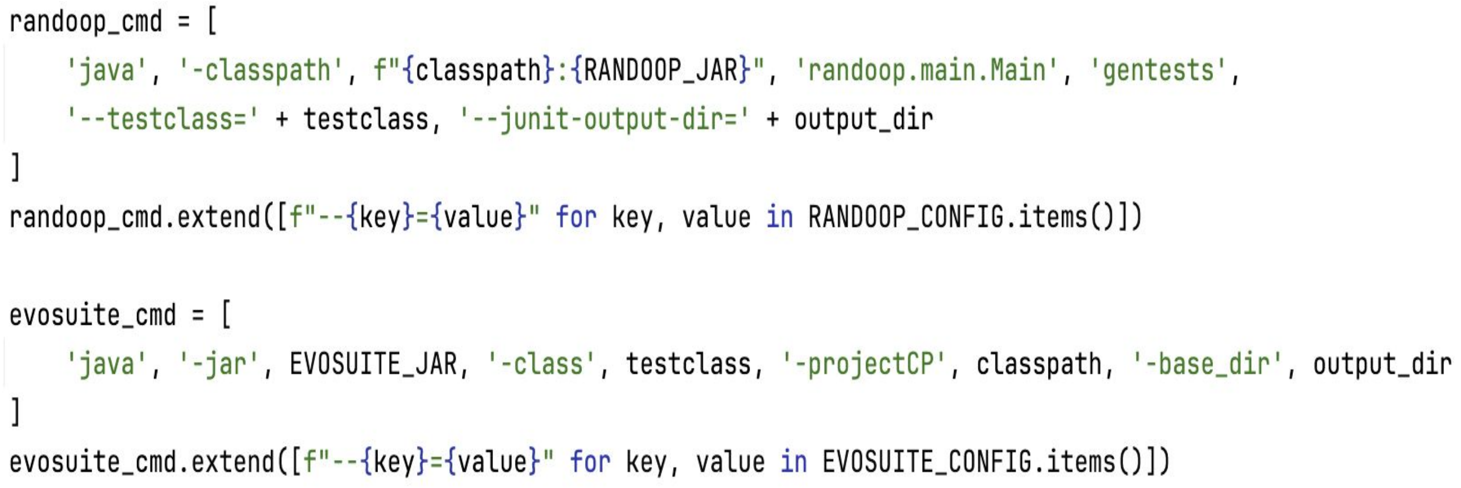
- CodeBLEU
  - Higher value -> More Similarity



Milestone 3: Test/Coverage Generation

Randoop/Evosuite

- Set proper environment for each bug (jenv)
- Created process pools to leverage parallelisation
- Extracted project dependencies and generate classpath
- Run said tools on each selected bug

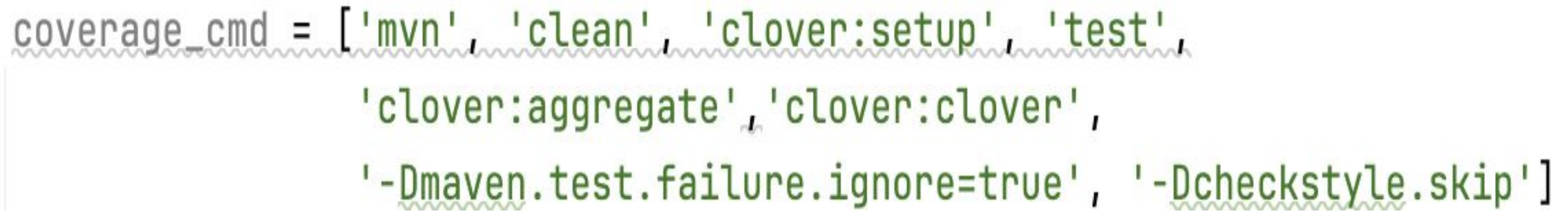


```
randoop_cmd = [
    'java', '-classpath', f"{classpath}:{RANDOOP_JAR}", 'randoop.main.Main', 'gentests',
    '--testclass=' + testclass, '--junit-output-dir=' + output_dir
]
randoop_cmd.extend([f"--{key}={value}" for key, value in RANDOOP_CONFIG.items()])

evosuite_cmd = [
    'java', '-jar', EVOSUITE_JAR, '-class', testclass, '-projectCP', classpath, '-base_dir', output_dir
]
evosuite_cmd.extend([f"--{key}={value}" for key, value in EVOSUITE_CONFIG.items()])
```

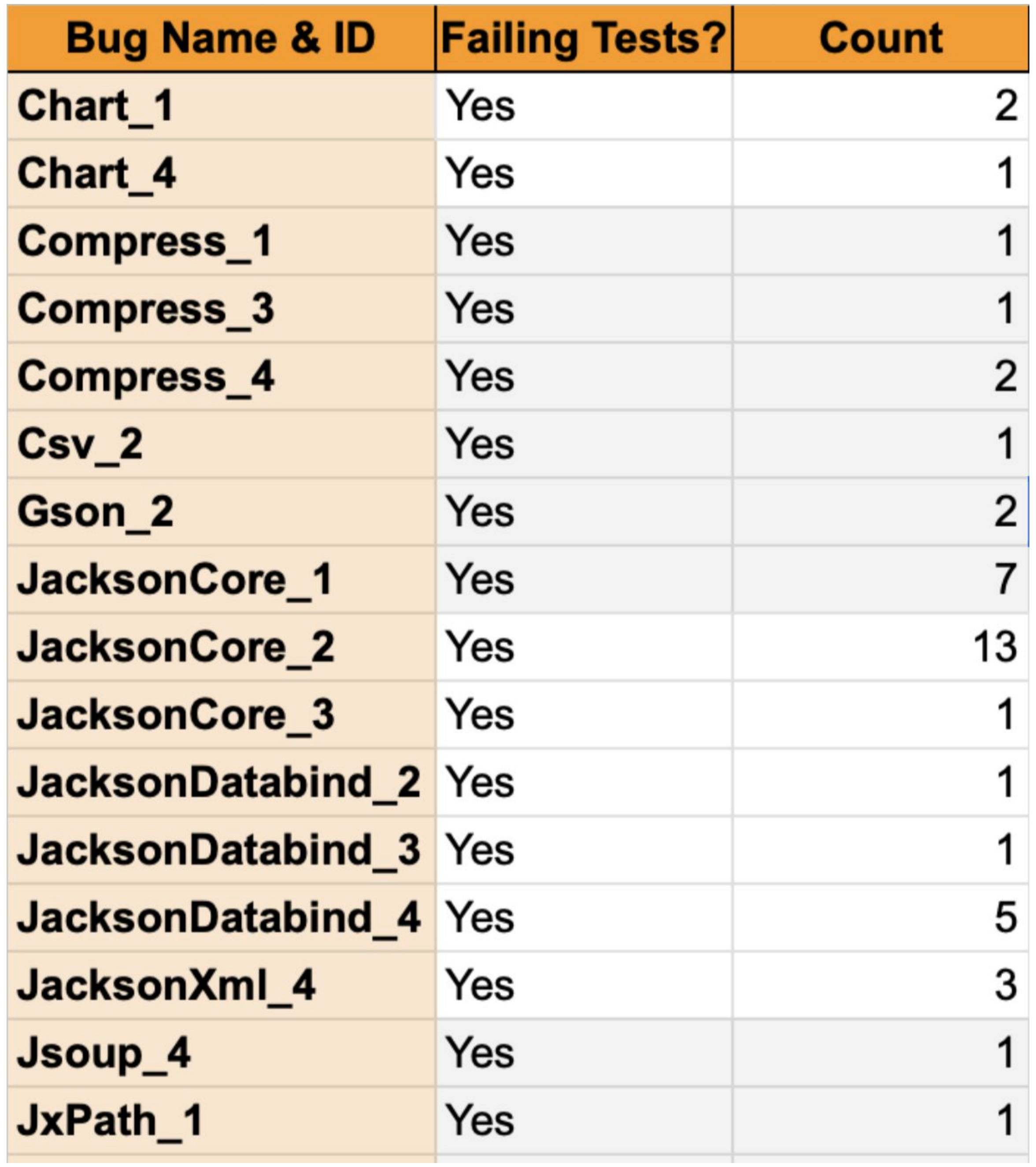
Validation & Coverage

- Create copies of the generated tests in the opposite version using shutil
- Run tests with maven to ensure tests are able to compile
- Create coverage reports with Clover/Intellj/JaCoCo
  - Used to determine if existing bug was detected or if new ones might have been uncovered



```
coverage_cmd = ['mvn', 'clean', 'clover:setup', 'test',
                'clover:aggregate','clover:clover',
                '-Dmaven.test.failure.ignore=true', '-Dcheckstyle.skip']
```

Milestone 3: Results

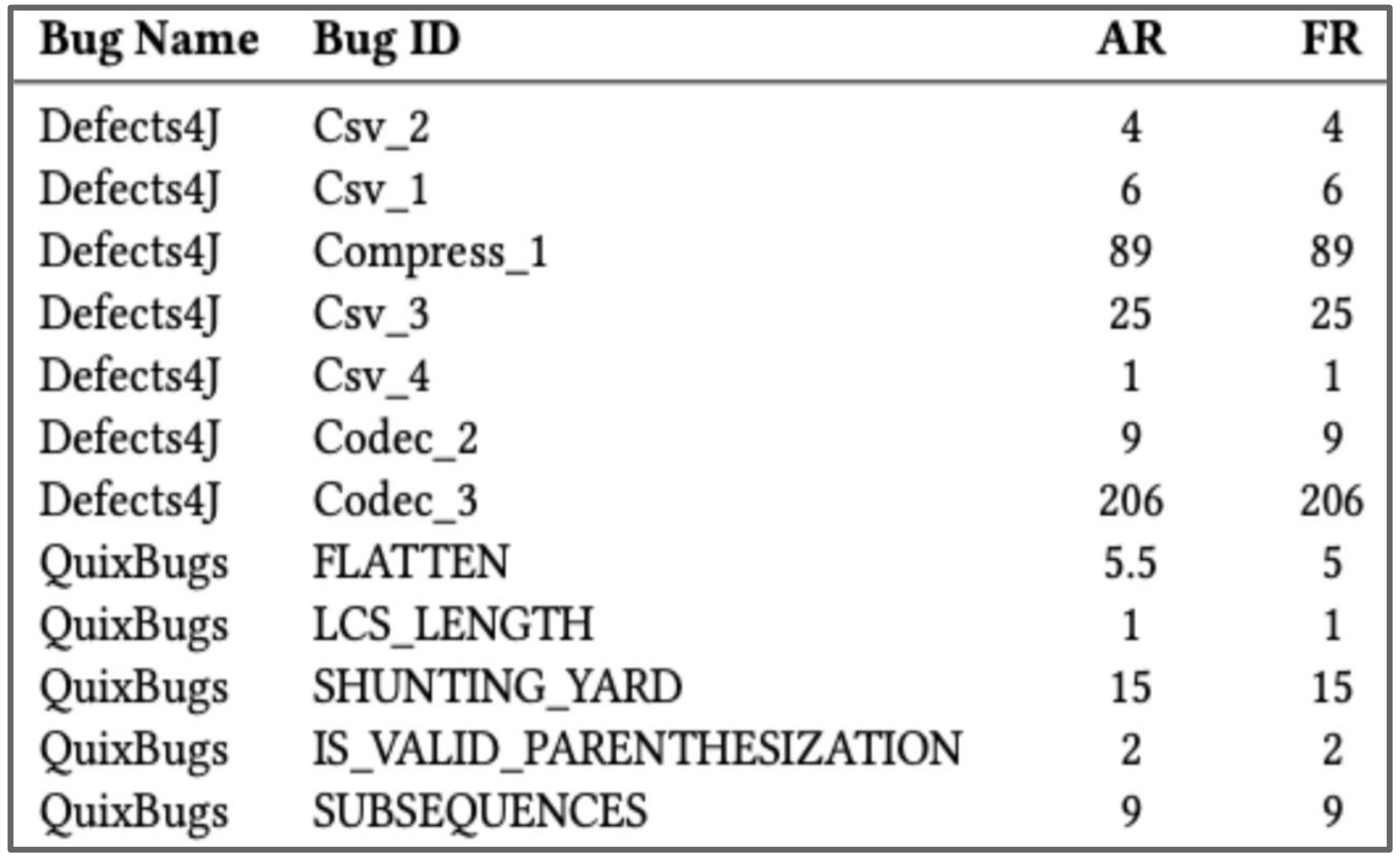


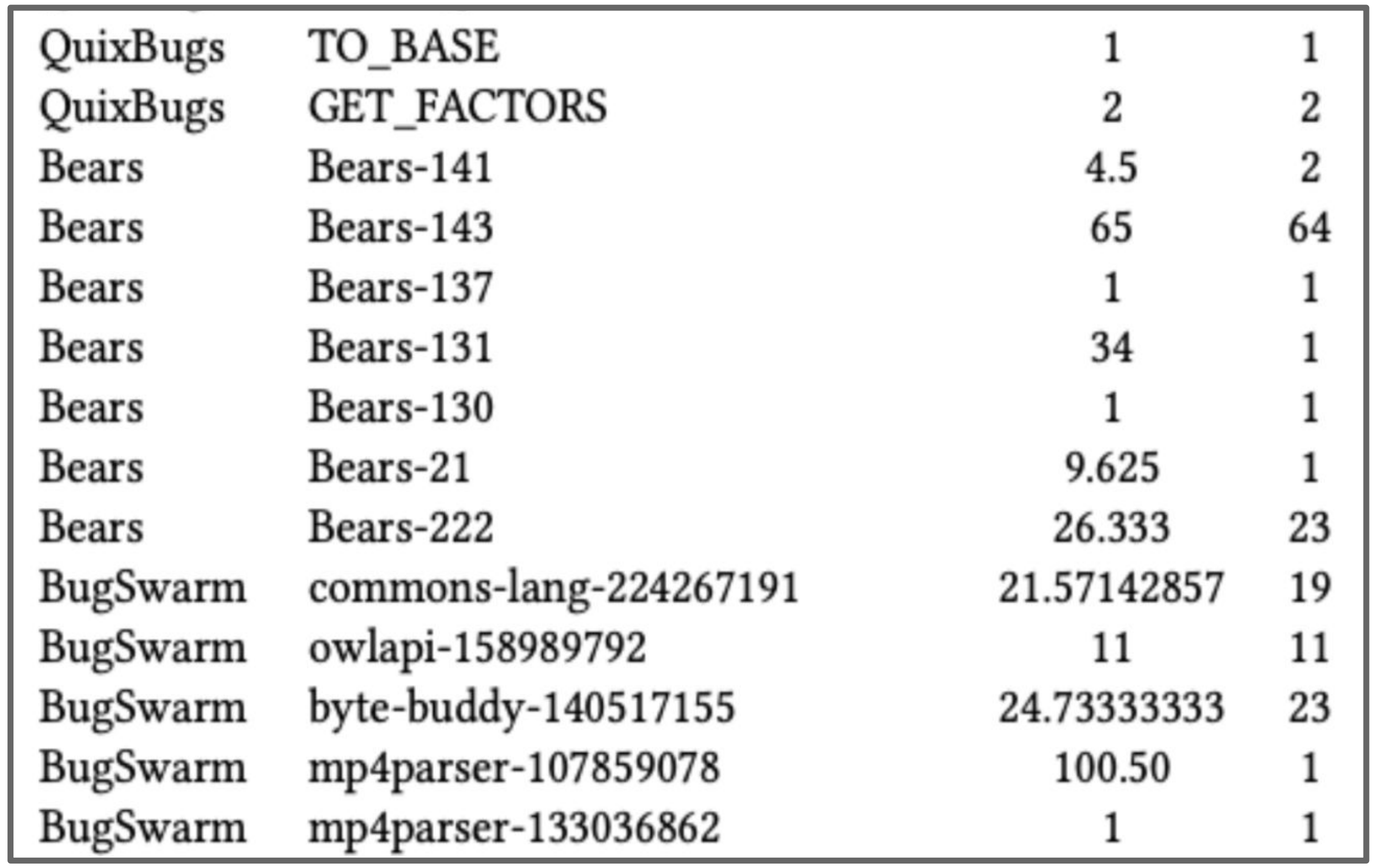
Bug Name & ID	Failing Tests?	Count
Chart_1	Yes	2
Chart_4	Yes	1
Compress_1	Yes	1
Compress_3	Yes	1
Compress_4	Yes	2
Csv_2	Yes	1
Gson_2	Yes	2
JacksonCore_1	Yes	7
JacksonCore_2	Yes	13
JacksonCore_3	Yes	1
JacksonDatabind_2	Yes	1
JacksonDatabind_3	Yes	1
JacksonDatabind_4	Yes	5
JacksonXml_4	Yes	3
Jsoup_4	Yes	1
JxPath_1	Yes	1

- Even with extensive resources (capped at 6 cores and 60 minutes) not all bug rendered new sets of randoop and evosuite tests
- Defects4J and QuixBugs were the sources of our best results
  - Close to complete generation of tests matching our criteria
- Randoop was more successful at generating error revealing tests, however evosuite produced more complete regression tests which better validate the projects stability

Milestone 4: Bug Localization

$$Suspiciousness\ (s) = \frac{fails(s)/_{totalfail}}{(fails(s)/_{totalfail}) + (pass\ (s)/_{totalpass})}$$





Bug Name	Bug ID	AR	FR
Defects4J	Csv_2	4	4
Defects4J	Csv_1	6	6
Defects4J	Compress_1	89	89
Defects4J	Csv_3	25	25
Defects4J	Csv_4	1	1
Defects4J	Codec_2	9	9
Defects4J	Codec_3	206	206
QuixBugs	FLATTEN	5.5	5
QuixBugs	LCS_LENGTH	1	1
QuixBugs	SHUNTING_YARD	15	15
QuixBugs	IS_VALID_PARENTHESIZATION	2	2
QuixBugs	SUBSEQUENCES	9	9
QuixBugs	TO_BASE	1	1
QuixBugs	GET_FACTORS	2	2
Bears	Bears-141	4.5	2
Bears	Bears-143	65	64
Bears	Bears-137	1	1
Bears	Bears-131	34	1
Bears	Bears-130	1	1
Bears	Bears-21	9.625	1
Bears	Bears-222	26.333	23
BugSwarm	commons-lang-224267191	21.57142857	19
BugSwarm	owlapi-158989792	11	11
BugSwarm	byte-buddy-140517155	24.73333333	23
BugSwarm	mp4parser-107859078	100.50	1
BugSwarm	mp4parser-133036862	1	1

Milestone 4: Findings

- **Simpler bugs = Better ranking:**
  The system ranked bugs better when the identified code closely matched the real bug.
- **QuixBugs wins:**
  Easy-to-understand bugs in QuixBugs were pinpointed well by the automated tools. Followed by Bears, Defects4J and BugSwarm.
- **Works on harder bugs too:**
  The approach worked on more complex bugs, though with less accuracy.

Conclusion

- Automated techniques hold promise for significantly accelerating bug detection in real-world software.
- By pinpointing potentially faulty code sections, these techniques can save developers valuable time and effort during the debugging process.
- This paves the way for further development and integration of such tools into the software development workflow.

Thank you!