TESTABILITY REFACTORING IN PULL REQUESTS

Patterns and Trends

Pavel Reich & Walid Maalej, University of Hamburg, Germany

HOW MANY OF YOU HAVE EVER WRITTEN A UNIT-TEST FOR LEGACY CODE?

MOTIVATING EXAMPLE 1
WIDEN ACCESS FOR INVOCATION

```
private Multimap<Cell, Long> getCellTsPairsToSweep(Multimap<Cell, Long> cellTsMappings,
                                                   PeekingIterator<RowResult<Value>> values,
                                                   long sweepTimestamp,
                                                   SweepStrategy sweepStrategy,
                                                   @Output Set<Cell> sentinelsToAdd) {
    Multimap<Cell, Long> cellTsMappingsToSweep = HashMultimap.create();
@VisibleForTesting
SweepCellsAndSentinels getStartTimestampsPerRowToSweep(
        Multimap<Cell, Long> startTimestampsPerCell,
        PeekingIterator<RowResult<Value>> values,
        long sweepTimestamp,
        SweepStrategySweeper strategySweeper) {
```

Production code

```
    @Test
    public void getTimestampsToSweep_onlyTransactionUncommitted_returnsIt() {
        Multimap<Cell, Long> timestampsPerRow = ImmutableMultimap.of(SINGLE_CELL, LOW_START_TS);
        when(mockTransactionService.get(timestampsPerRow.values())).thenReturn(ImmutableMap.of(LOW_START_TS,
TransactionConstants.FAILED_COMMIT_TS));

        SweepCellsAndSentinels sweepCellsAndSentinels = sweepTaskRunner.getStartTimestampsPerRowToSweep(timestampsPerRow,
                Iterators.peekingIterator(ClosableIterators.emptyImmutableClosableIterator()),
                HIGH_START_TS,
                conservativeStrategySweeper);
```

Test code

Github/palantir
Pull request 729

MOTIVATING EXAMPLE 2
CREATE CONSTRUCTOR

```java
public ExecutionBusiness(ApiContext apiContext) {
    WorkflowBusiness workflowBusiness = new WorkflowBusiness();
    ApplicationBusiness applicationBusiness = new ApplicationBusiness();
    this.apiContext = apiContext;
    this.simulationBusiness = new SimulationBusiness();
    this.workflowBusiness = workflowBusiness;
    this.configurationBusiness = new ConfigurationBusiness();
    this.applicationBusiness = applicationBusiness;
    this.pipelineBusiness = new PipelineBusiness(apiContext, workflowBusiness,
}

public ExecutionBusiness(ApiBusiness ab) {
    super(ab);
public ExecutionBusiness(ApiContext apiContext,
                         SimulationBusiness simulationBusiness,
                         WorkflowBusiness workflowBusiness,
                         ConfigurationBusiness configurationBusiness,
                         ApplicationBusiness applicationBusiness,
                         PipelineBusiness pipelineBusiness) {
    this.apiContext = apiContext;
    this.simulationBusiness = simulationBusiness;
    this.workflowBusiness = workflowBusiness;
    this.configurationBusiness = configurationBusiness;
    this.applicationBusiness = applicat
    this.pipelineBusiness = pipelineBus
```

Production code

```java
@Test
public void checkIfAdminCanAccessAnyExecution() throws Exception {
    ApiContext apiContext = new ApiContext(null, null, null, prepareTestUser(0, true));
    WorkflowBusiness mockedWb = prepareMockedWorkflowBusiness(EXEC_ID, new Simulation());
    ExecutionBusiness sut = new ExecutionBusiness(apiContext, null, mockedWb, null, null, null);
    sut.checkIfUserCanAccessExecution(EXEC_ID);
}
```

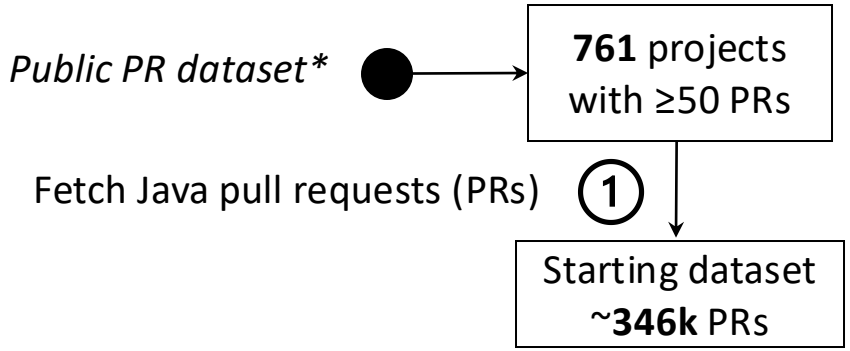
Test code

github/virtual-imaging-platform
Pull request 52

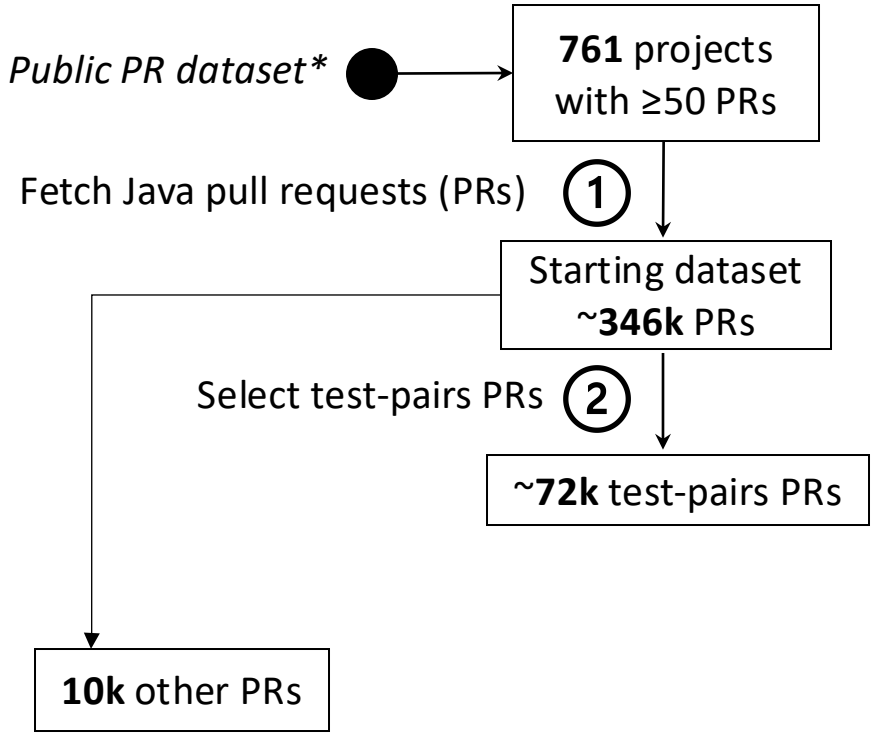
THE BIGGER PICTURE

- Developers don’t like writing unit-tests for legacy code
- Legacy code may contain hard-to-test singletons, direct operations on network, database, system time
- Need to refactor legacy production code before writing unit-tests
- What can we learn from studying these refactorings in open-source?

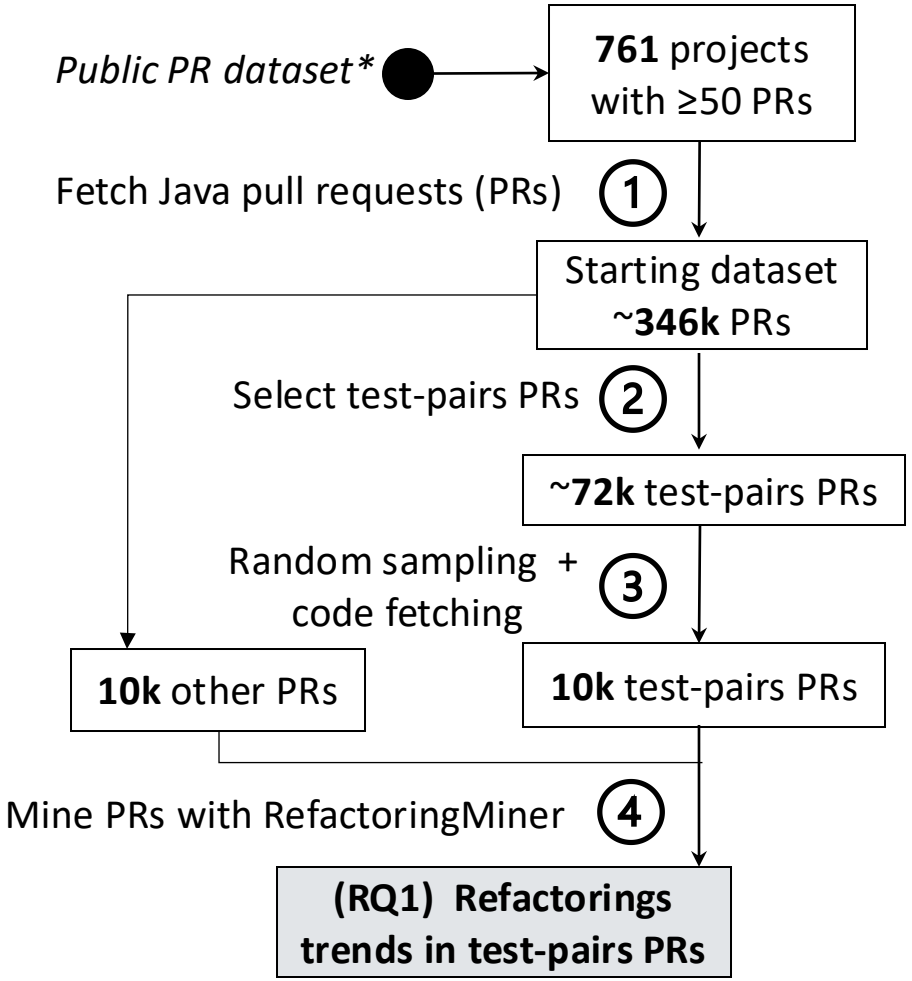
Data Collection, Preparation, and Mining

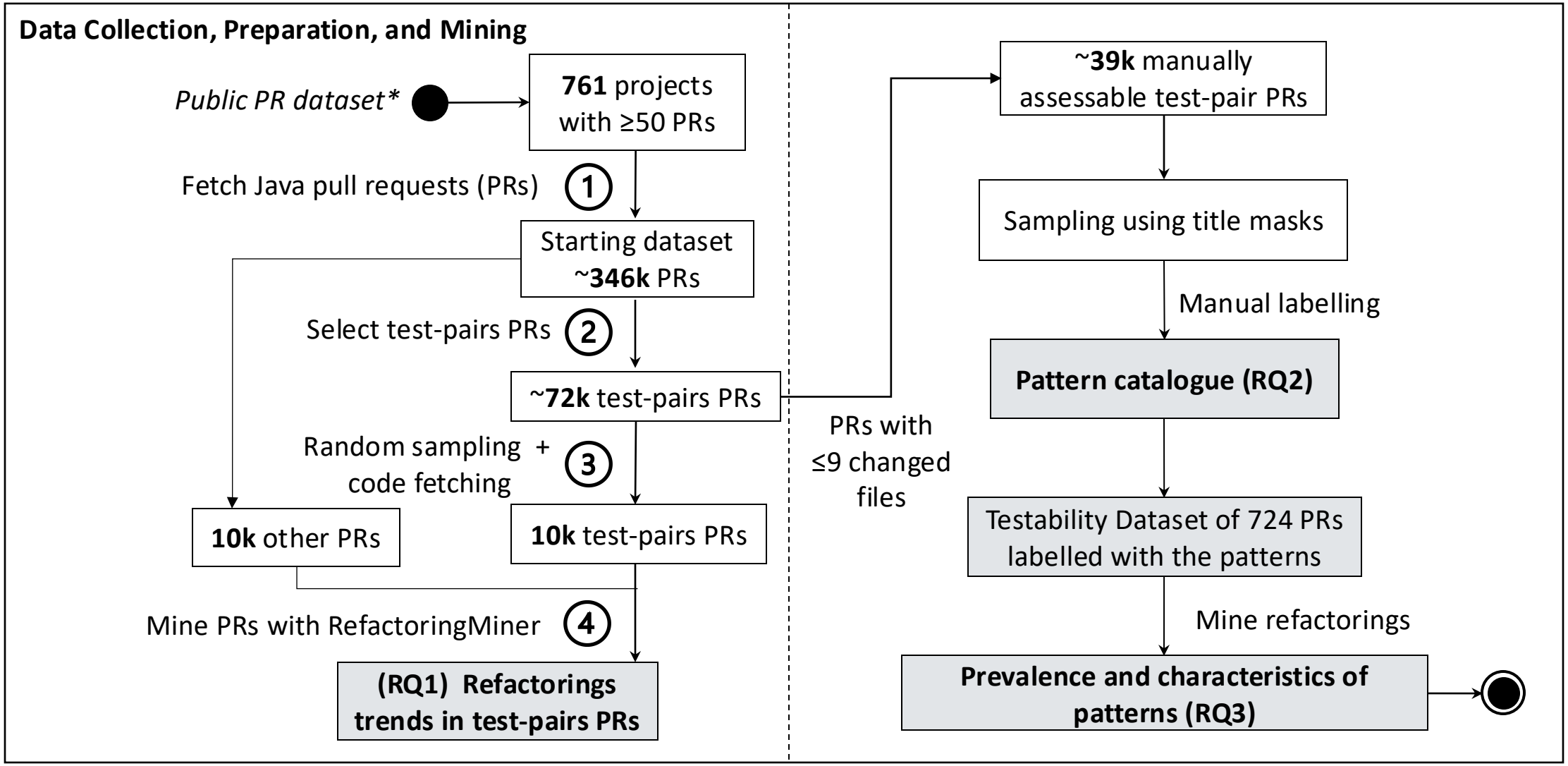


Data Collection, Preparation, and Mining



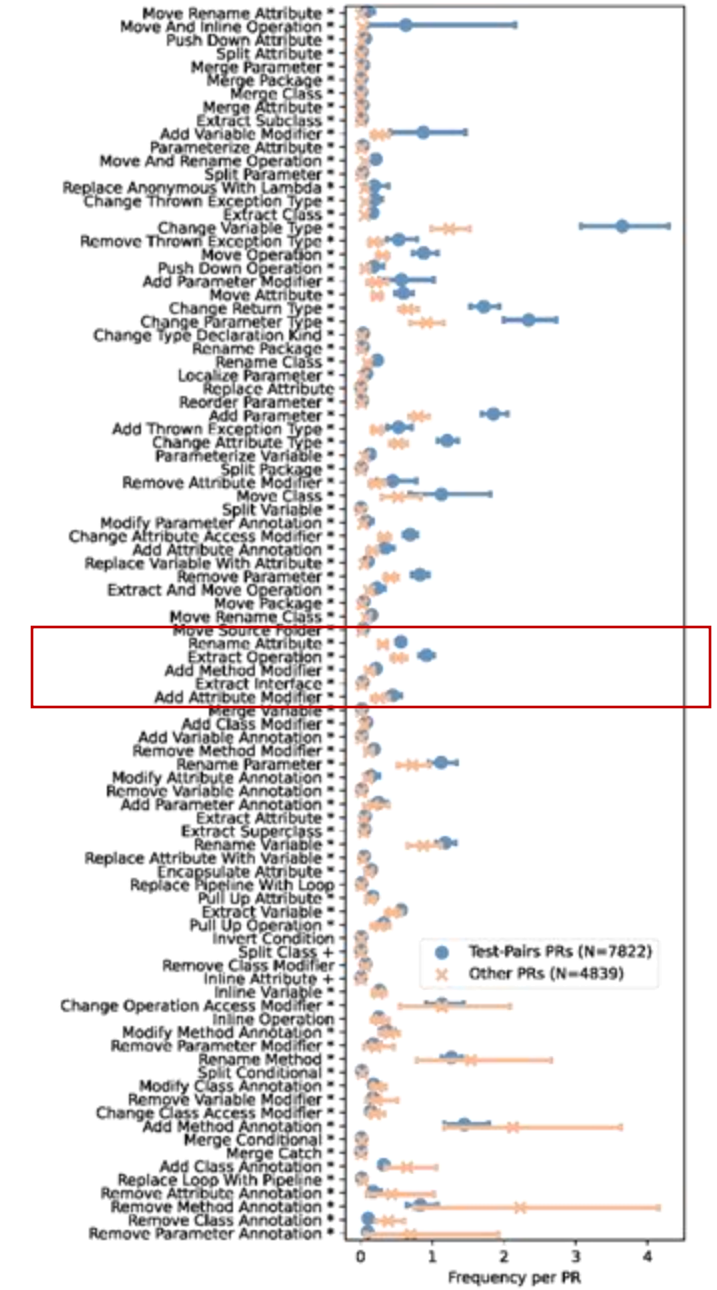
Data Collection, Preparation, and Mining




Data Collection, Preparation, and Mining
Public PR dataset*
761 projects with ≥50 PRs
Fetch Java pull requests (PRs)
1
Starting dataset ~346k PRs
Select test-pairs PRs
2
~72k test-pairs PRs
Random sampling + code fetching
3
10k other PRs
10k test-pairs PRs
Mine PRs with RefactoringMiner
4
(RQ1) Refactorings trends in test-pairs PRs
PRs with ≤9 changed files
~39k manually assessable test-pair PRs
Sampling using title masks
Manual labelling
Pattern catalogue (RQ2)
Testability Dataset of 724 PRs labelled with the patterns
Mine refactorings
Prevalence and characteristics of patterns (RQ3)

REFACTORING-MINING OF PRS

- 10k Test-Pairs PRs with changes in production and test code + 10k “Other PRs” without such changes
- We found more refactorings in production code in TP-PRs than in other PRs: 34.9 vs 23.6 refactorings per PR
- Some refactorings are more frequent: extract an operation, extract and move operation, etc.



MANUAL ANALYSIS

- PRs categorized as
  - Changes in production code solely to improve testability (only_ref_for_test)
  - Same as above + features/bugfixes (incl_ref_for_test)
  - Irrelevant for testability changes (only changes, bugfixes, refactorings in test code…)
- Testability-relevant PRs can contain one or more testability refactoring patterns

TESTABILITY REFACTORING PATTERNS IN PRS

Some patterns are similar to dependency-breaking techniques by Feather (2004)

Pattern name	Count	%
extract_method_for_override	51	22.2
extract_method_for_invocation	39	17.0
widen_access_for_invocation	35	15.2
extract_class_for_invocation	29	12.6
add_constructor_param	25	10.9
extract_class_for_override	15	6.5
create_constructor	10	4.3
widen_access_for_override	9	3.9
override_system_time	4	1.7
extract_attribute_for_assertion	3	1.3
Total	230	100

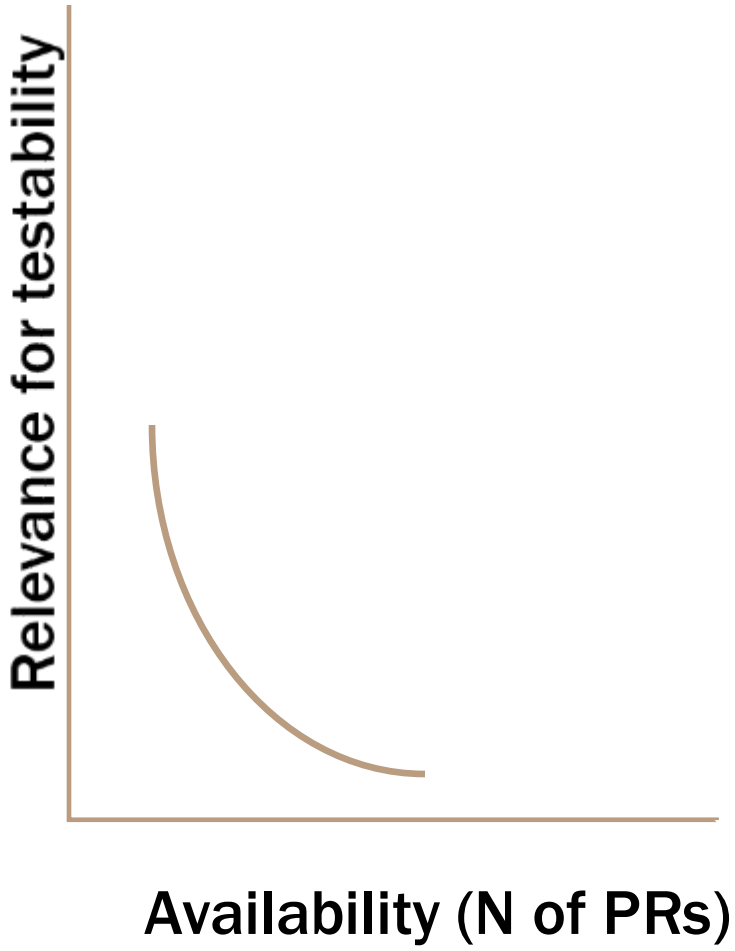
PRS WITH TESTABILITY REFACTORINGS

Pattern name	Count
extract_method_for_override	51
extract_class_for_override	15
widen_access_for_override	9
override_system_time	4
extract_attribute_for_assertion	3
Total	82

Pattern name	Count
extract_method_for_invocation	39
widen_access_for_invocation	35
extract_class_for_invocation	29
add_constructor_param	25
create_constructor	10
extract_attribute_for_assertion	3
Total	230

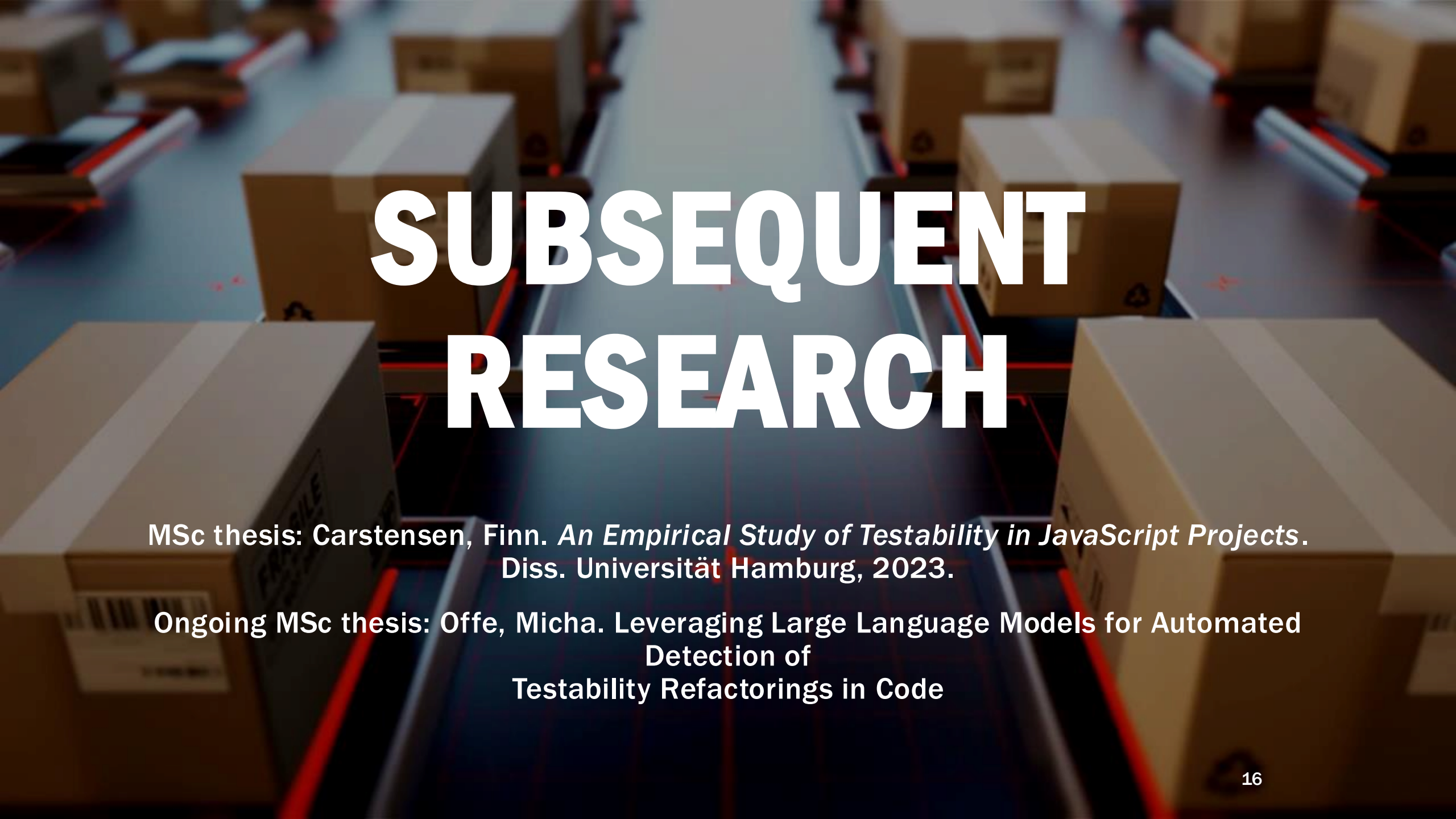
RELEVANCE OF TITLE MASKS FOR PULL REQUESTS WITH TESTABILITY

	Testability irrelevant	Testability relevant
Testability_body	49 (45.0%)	**60 (55%)**
testability	6 (33.3%)	**12 (66.7%)**
Refactor for test	38 (52.1%)	**35 (47.9%)**
test	117 (78.5%)	**32 (21.5%)**
Dependency	43 (86.0%)	7 (14%)
Concurrency	44 (88.0%)	6 (12%)
Network	45 (90.0%)	5 (10%)
Singleton	23 (95.8%)	1 (4.2%)
Inject	44 (88.0%)	6 (12%)
Other	131 (87.3%)	19 (12.7%)
Total (N=724)	540 (74.7%)	184 (25.3%)



MAIN FINDINGS

- In ~13% of test-pairs PRs, developers refactor production code to write unit-tests
- Typically, methods/classes are extracted to override or to invoke
- Refactorings improve controllability and observability (Freedman et al., 1991)
- Pattern override_system_time can be implemented in different ways



SUBSEQUENT RESEARCH

MSc thesis: Carstensen, Finn. *An Empirical Study of Testability in JavaScript Projects*. Diss. Universität Hamburg, 2023.

Ongoing MSc thesis: Offe, Micha. Leveraging Large Language Models for Automated Detection of
Testability Refactorings in Code

FINDOUT MORE AND CONTRIBUTE

ICSE2023-Paper

Datasets and catalog of patterns available on Github (e.g. for teaching or as refactoring templates in IDEs)

Pavel Reich & Walid Maalej, University of Hamburg, Germany
Contact pavel.reich@studium.uni-hamburg.de

GitHub Repo