How Disabled Test Manifests in Test Maintainability Challenges?



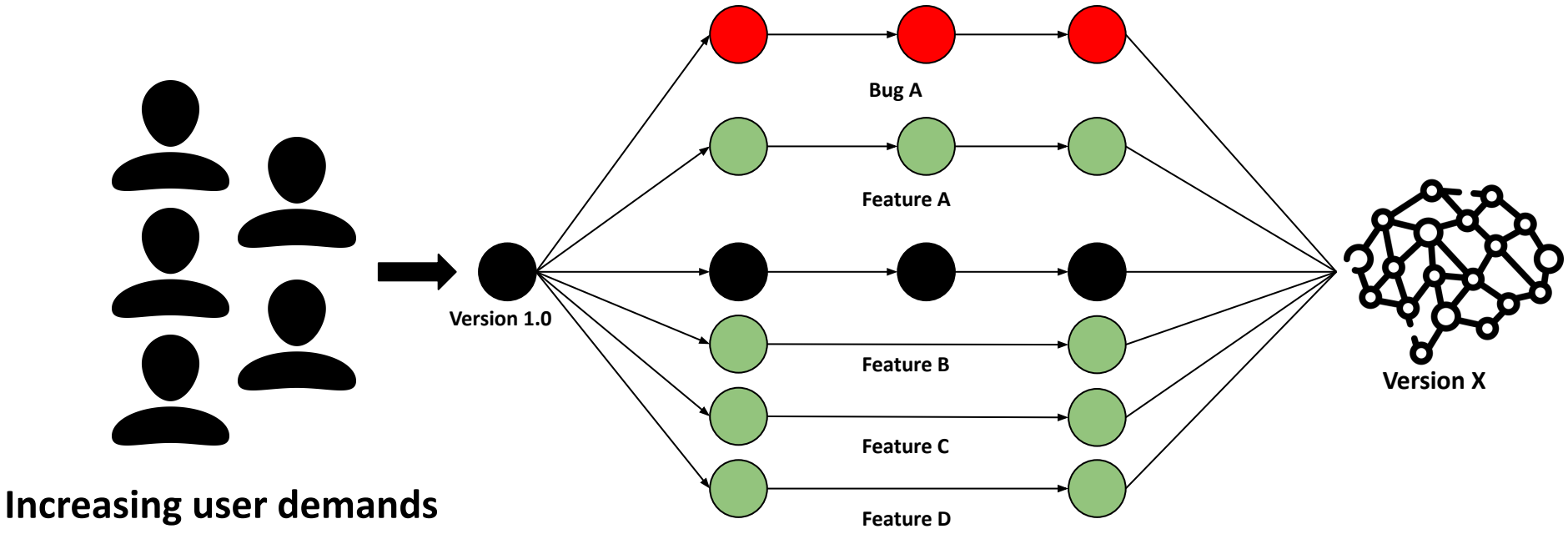

UNIVERSITÉ Concordia UNIVERSITY

SPEAR Software PErformance, Analysis, and Reliability

Content list (delete this page at last)

1. Tests are important,
    a. regression tests
2. A comic about the developers problem in tests and the motivation of disabling tests
3. It may not always be a good thing
    a. list some side effects
4. Motivations of our paper, tech debts and maintenence
5. What and how did we do
    a. the tool
    b. manual analysis
6. The results
    a. quantitative
    b. qualitative
7. Implications

Software becomes increasingly complex overtime



Software testing plays a vital role in software quality

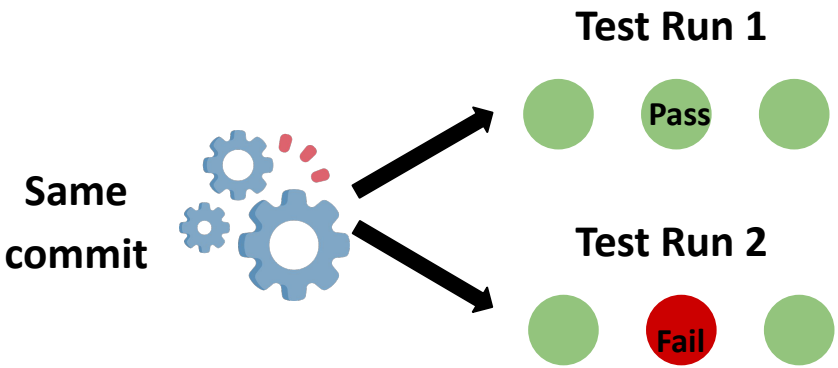
Detect bugs early

Reduce later cost to fix defect

Customer Satisfaction

Test code is also prone to age and quality issues

"Flaky Test"



"Test Smell"

Negatively impact program comprehension and maintainability

Increasing popularity of test maintenance frameworks in Java

TestNG
JUnit

Increasing Junit and TestNG usage

Benchmark Unit Test

Reset Fixture

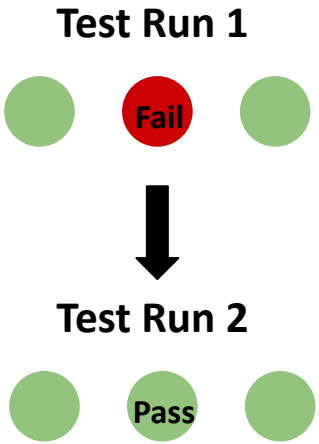
```
@Test (timeout = 120000)
public void methodName()
throws Exception {
    ……
}

public class className throws Exception {
 @Before
 public static void setupPerMethod(){
        ……
   }
}
```

Test disabling practice is introduced to alleviate maintainability difficulties

```
@Ignore
public void methodName()
throws Exception {
    ……
}
```

Meet project deliverables



There are other ways to disable test code

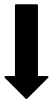
Utilize Annotation

```
@Ignore
public void methodName()
throws Exception {
    ……
}
```

Comment out annotation

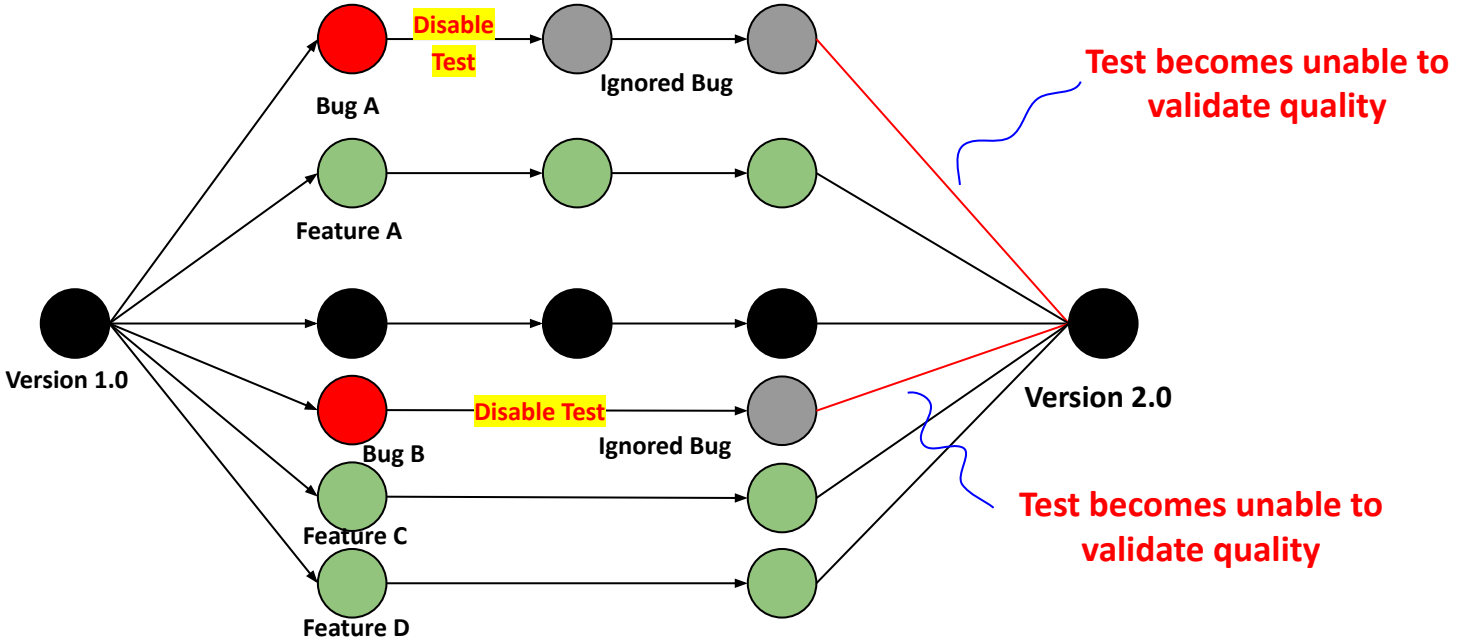
```
//@Test
public void methodName()
throws Exception {
    ……
}
```

Comment out test code

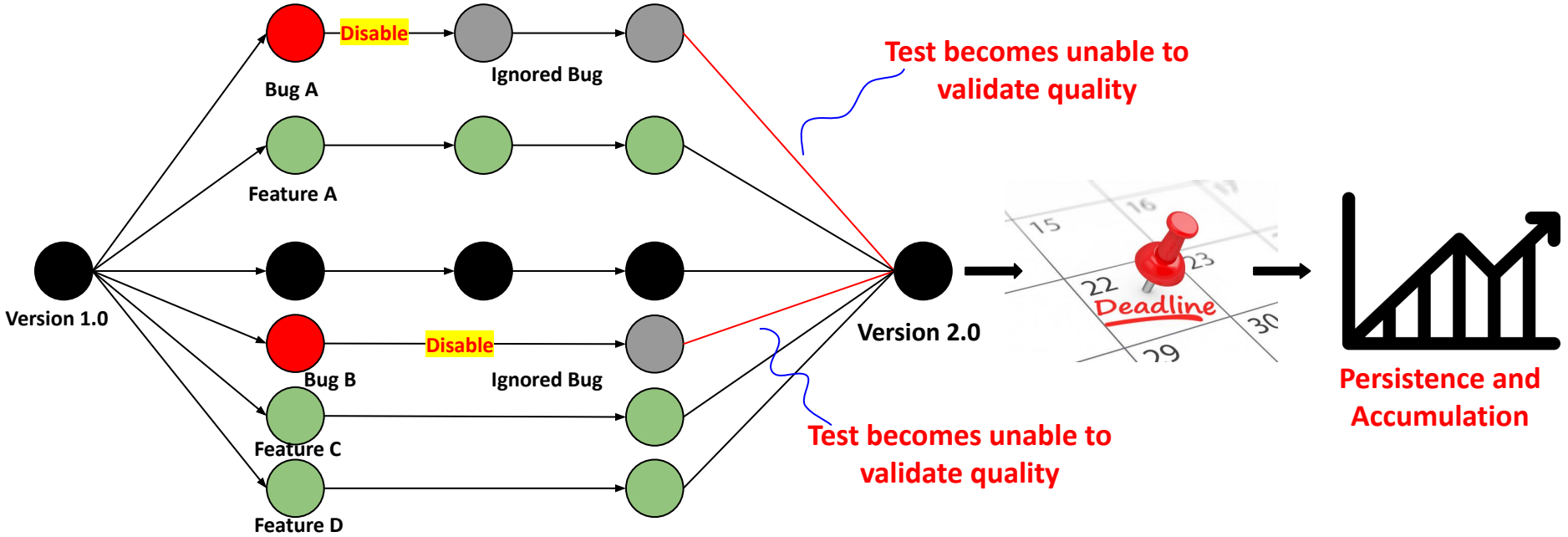


```
/*@Test
public void methodName()
throws Exception {
    ……
}
*/
```

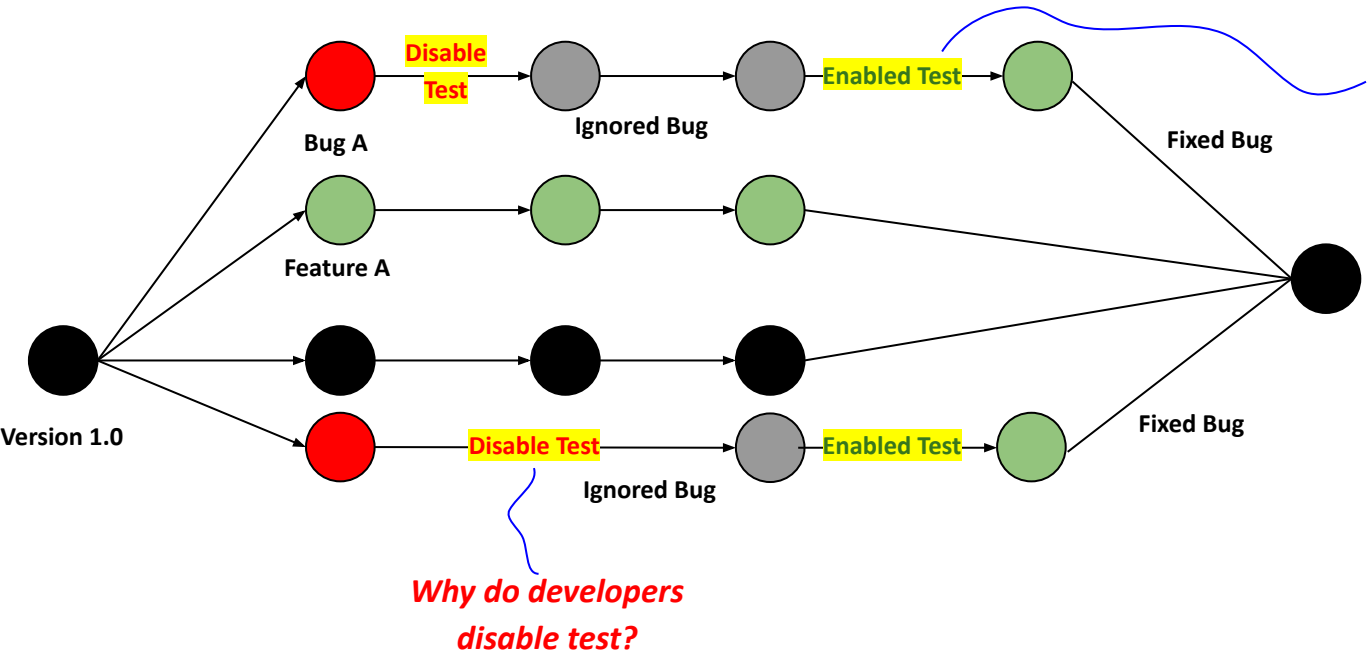
Test disabling practice may become a source of technical-debt



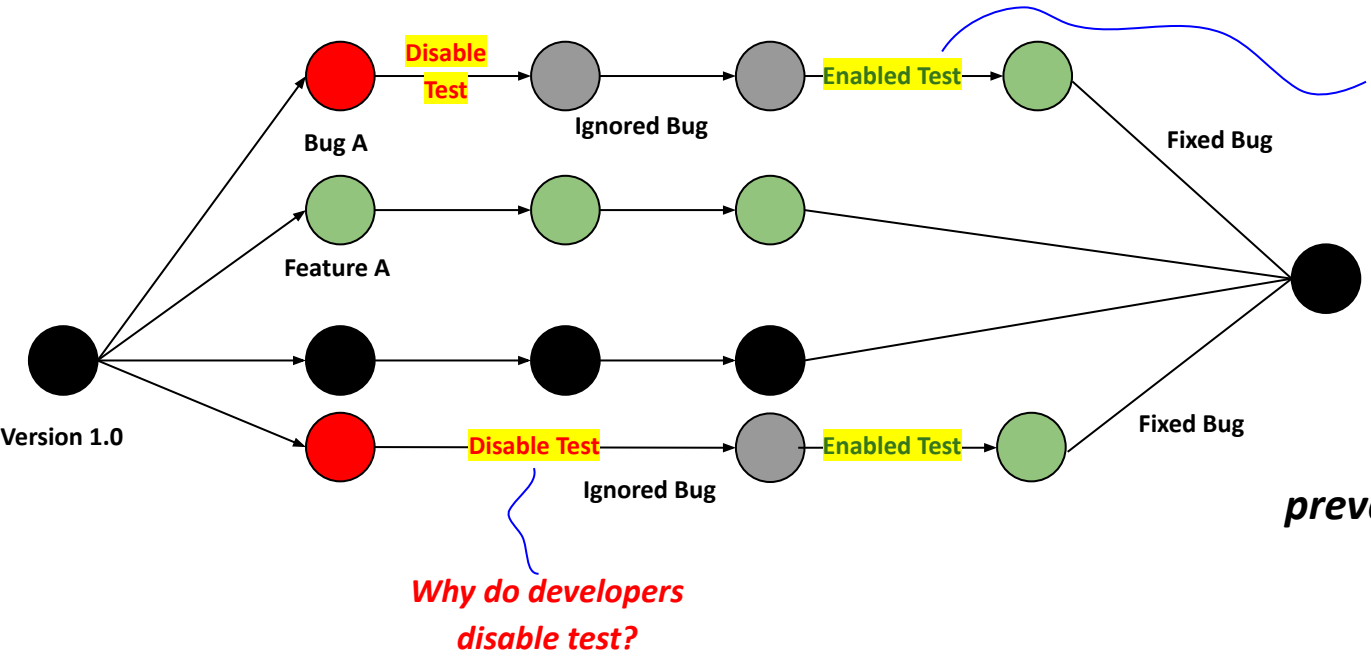
Test disabling practice may become a source of technical-debt



Understanding test disabling practice is vital to providing better repair strategies



Understanding test disabling practice is vital to providing better repair strategies



Limited study on the prevalence and maintenance of disabling test practice

Research Questions?

RQ1: How common are test disabling changes?

Study the prevalence of the test disabling practices.

RQ2: What is the change pattern of disabled tests?

Study the change patterns, durations between disabled and re-enabled or deleted and final destinations of the disabled tests.

RQ3: Why do developers utilize test disabling practice?

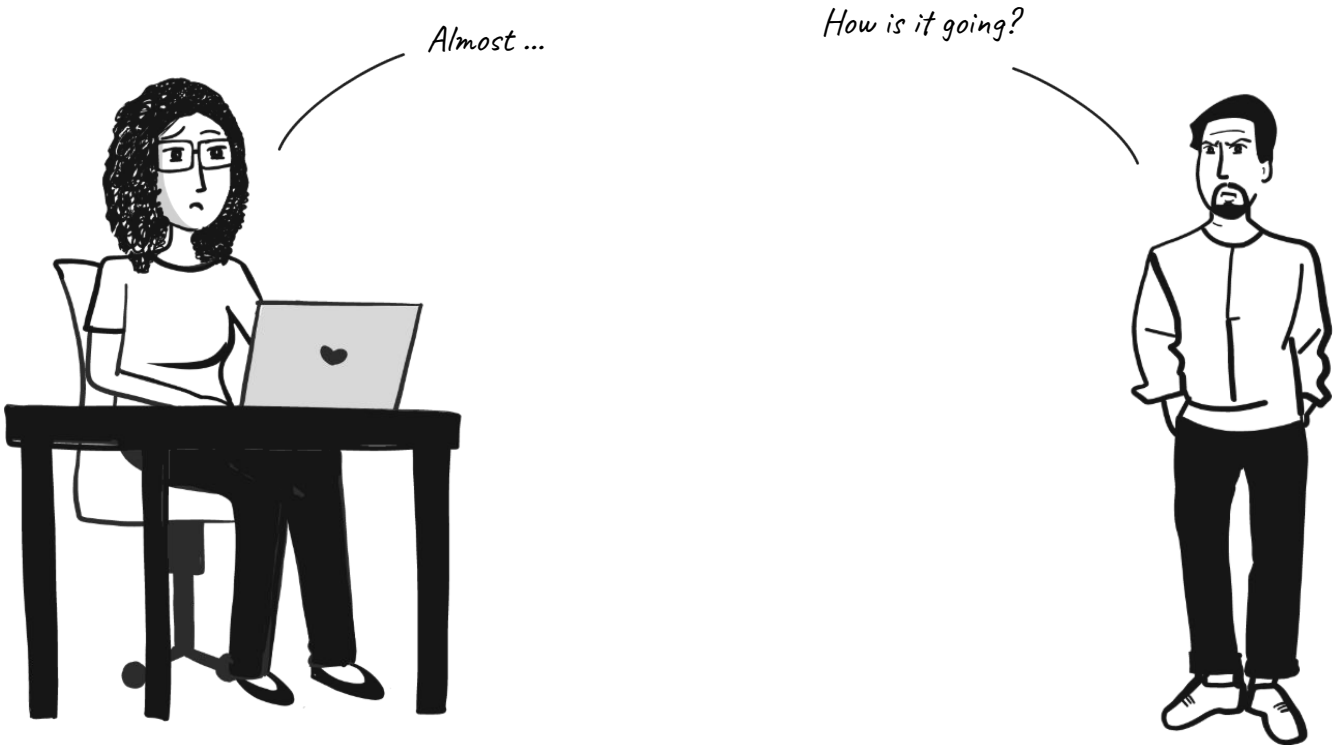
Study the motivations behind the test disabling, re-enabling and deleting. Further study the possible maintenance challenges of the disabled tests.

Test Disabling Is The Salvation

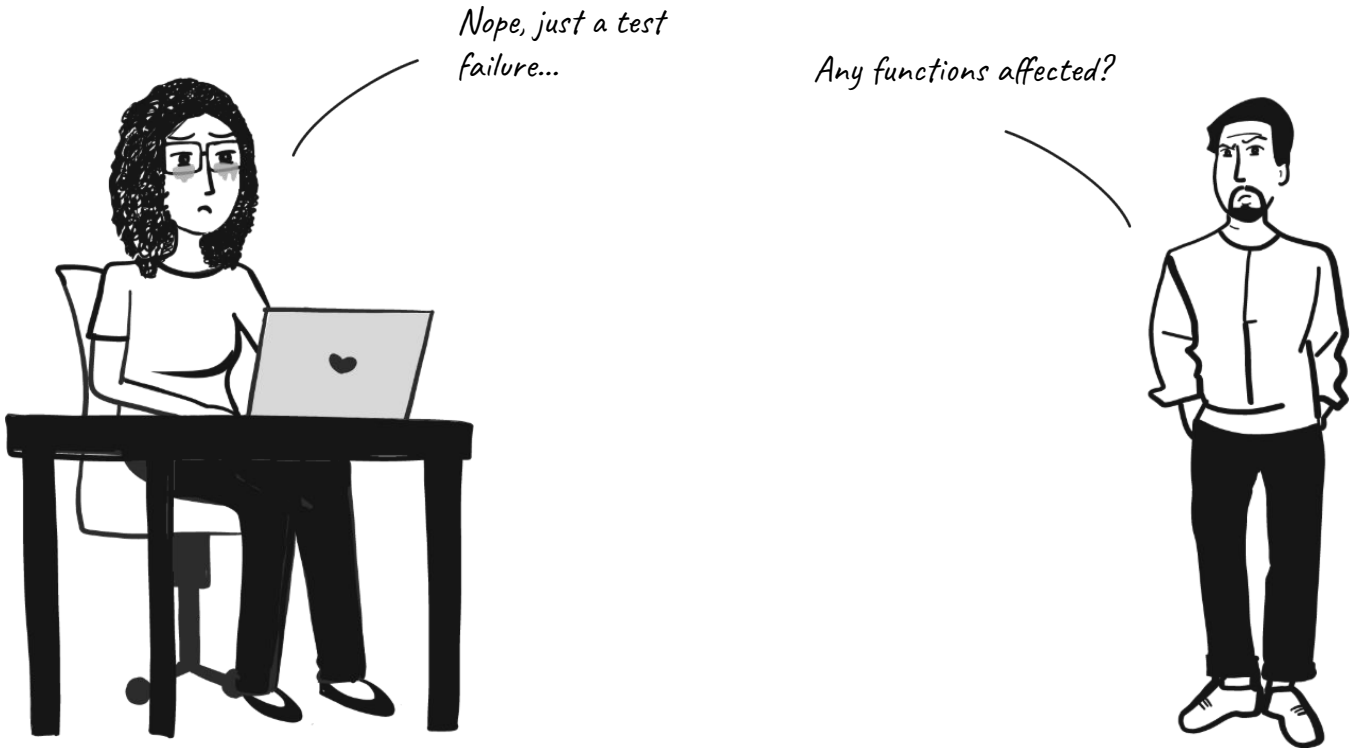
Test Disabling Is The Salvation

55 minutes later...

Test Disabling Is The Salvation



Test Disabling Is The Salvation



Test Disabling Is The Salvation

Test Disabling Is The Salvation

To meet the demand of faster delivery of applications, test disabling becomes the salvation to developers. ***However, is it always a blessing?***

Comment: Some possible bad side effects

Comment: So in this paper, we investigated the prevalence, lifecycle, reasons of disabling tests and finally propose the implications to researchers and developers and indicate the future direction.

Mine the test disabling practices

We consider all the methods that ever have an active or commented out @Test annotation as a test.

Mine the test disabling practices

Different strategies to disable tests:

1. Adding an @Ignore annotation.
2. Setting @Test(enabled=false).
3. Deleting @Test annotation.
4. Commenting out the entire test method or class.

How to mine the test disabling practices?

1. Identify the enabled and disabled tests in each commit.

What is a test?

We consider all the methods that ever have a @Test annotation as a test.

```
@Test
public void testDownloadFile() throws Exception {
    final Map<String, Object> headers = new HashMap<>();
    // parameter type is String
    headers.put("CamelBox.fileId", testFile.getID());
    // parameter type is java.io.OutputStream
    ByteArrayOutputStream output = new ByteArrayOutputStream();
    headers.put("CamelBox.output", output);
    // parameter type is Long
    headers.put("CamelBox.rangeStart", null);
    // parameter type is Long
    headers.put("CamelBox.rangeEnd", null);
    // parameter type is com.box.sdk.ProgressListener
    headers.put("CamelBox.listener", null);

    final java.io.OutputStream result = requestBodyAndHeaders("direct://DOWNLOADFILE", null, headers);

    assertNotNull("downloadFile result", result);
    LOG.debug("downloadFile: " + result);
}
```

Types of disabled tests

Add @Ignore

```
@Test @Ignore
public void methodName()
throws Exception {
    ……
}
```

Set enabled to false

```
@Test(enabled=false)
public void methodName()
throws Exception {
    ……
}
```

Comment out annotation

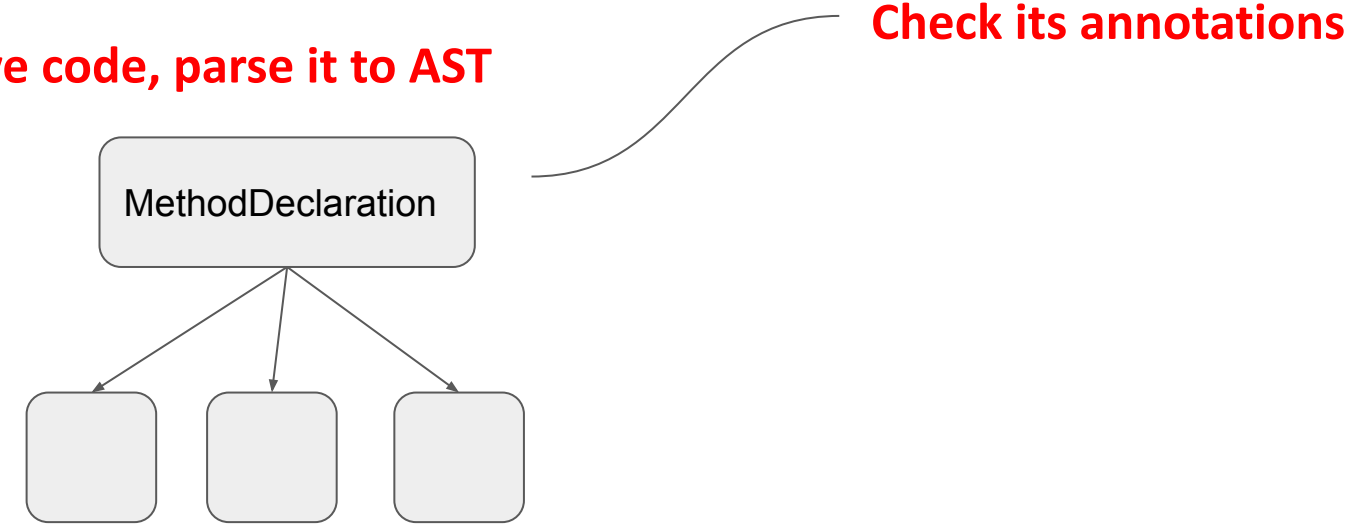
```
//@Test
public void methodName()
throws Exception {
    ……
}
```

Comment out test code

```
/*@Test
public void methodName()
throws Exception {
    ……
}
*/
```

Mine the disabled tests

If it is live code, parse it to AST



Check its annotations

Mine the disabled tests

To mine the tests in comments

1. Extract the content from the consecutive statement comments or a single block comment.

```
@Test(dataProvider = "testSortedFiltersDataProvider")
public void testSortedFilters(String filters, String expectedFilters) {
    String sortedFilters = ThirdEyeUtils.getSortedFilters(filters);
    Assert.assertEquals(sortedFilters, expectedFilters);
}
```

Mine the disabled tests

To mine the tests in comments

1. Extract the content from the consecutive statement comments or a single block comment as a single target.
2. Detect the annotations in the comments.

```
@Test(dataProvider = "testSortedFiltersDataProvider")
public void testSortedFilters(String filters, String expectedFilters) {
    String sortedFilters = ThirdEyeUtils.getSortedFilters(filters);
    Assert.assertEquals(sortedFilters, expectedFilters);
}
```

Mine the disabled tests

To mine the tests in comments

1. Extract the content from the consecutive statement comments or a single block comment as a single target.
2. Detect the annotations in the comments.
3. Find the first ‘{’ after the detected annotations.

```
@Test(dataProvider = "testSortedFiltersDataProvider")
public void testSortedFilters(String filters, String expectedFilters) {
    String sortedFilters = ThirdEyeUtils.getSortedFilters(filters);
    Assert.assertEquals(sortedFilters, expectedFilters);
}
```

Mine the disabled tests

To mine the tests in comments

1. Extract the content from the consecutive statement comments or a single block comment as a single target.
2. Detect the annotations in the comments.
3. Find the first ‘{’ after the detected annotations.
4. Try to parse the content between the first detected annotation and the first detected ‘{’ as a method.

```
@Test(dataProvider = "testSortedFiltersDataProvider")
public void testSortedFilters(String filters, String expectedFilters) {}
    String sortedFilters = ThirdEyeUtils.getSortedFilters(filters);
    Assert.assertEquals(sortedFilters, expectedFilters);
}
```

Mine the disabled tests

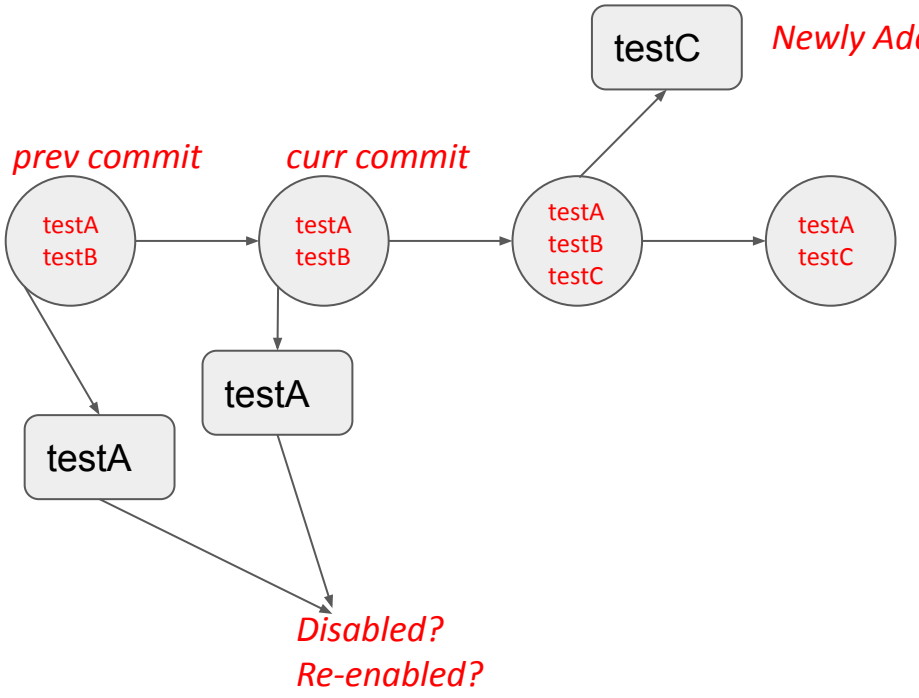
To mine the tests in comments

1. Extract the content from the consecutive statement comments or a single block comment as a single target.
2. Detect the annotations in the comments.
3. Find the first '{' after the detected annotations.
4. Try to parse the content between the first detected annotation and the first detected '{' as a method.
5. Detect the body of the test by pairing the '{' and '}'.

```
@Test(dataProvider = "testSortedFiltersDataProvider")
public void testSortedFilters(String filters, String expectedFilters) {
    String sortedFilters = ThirdEyeUtils.getSortedFilters(filters);
    Assert.assertEquals(sortedFilters, expectedFilters);
}
```

2. Decide if the status has changed between previous and current commit.

Mine the disabling changes



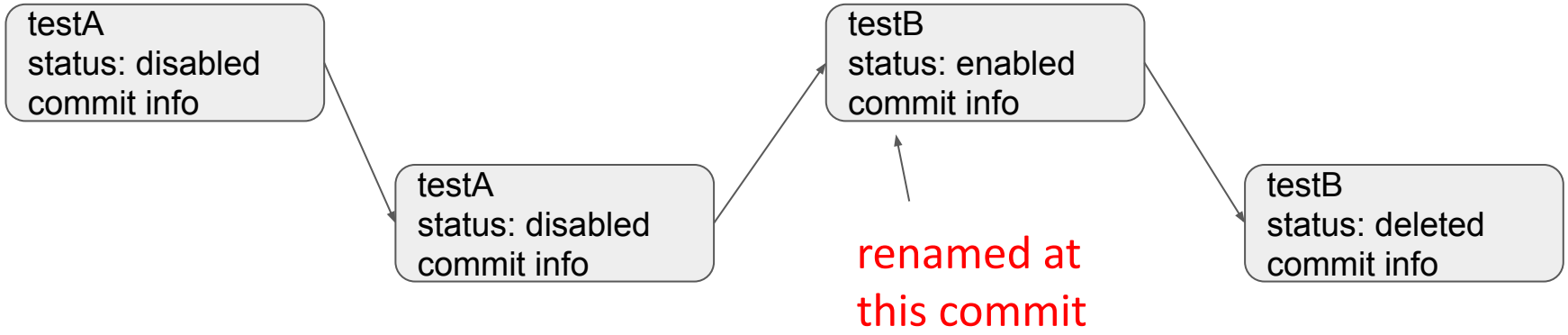
Mine the disabling changes

1. identification of a test: package.class.method(param, param, ...)

2. RefactoringMiner ->

- renamed package?
- renamed class?
- moved method?
- renamed method?
- changed parameters?
- …

Track the disabling/re-enabling lifecycle



RQ1: How common are test disabling changes?

Disabling and re-enabling tests are frequent comparing to rename refactorings on method and class levels.

	Method Level		Class Level		Total Changes	Total Commits
	Total	Per Commit	Total	Per Commit		
Test Disabling changes	2,581	2.7	530	0.6	3,111	949
Refactoring	2,495	1.9	120	0.1	2,651	1,334
Δ % Percent	**+3.4%**	**+42%**	**+341%**	**+500%**	**+19%**	**-29%**

RQ1: How common are test disabling changes?

Many tests remain disabled and most of re-enabled tests did not do any modifications.

	Disabling Tests	Re-enabling Tests		Deleting Disabled Tests
		Modified	Unmodified	
Method	2,581	314	486	762
Class	530	115	96	87
Total	**3,111(62.5%)**	**429(8.6%)**	**582(11.7%)**	**849(17.1%)**

RQ2: What is the Change Pattern of Disabled Tests?

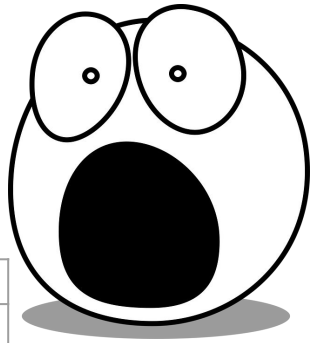
Most tests do not have many disabling/re-enabling changes in our study period.

Change patterns	Frequency
Unresolved tests	
Disabled	1,229
Disabled->Re-enabled->Disabled	21
Disabled->Deleted->Disabled	1
Total	**1,251**
Resolved tests	
Disabled->Re-enabled	871
Disabled->Deleted	824
Disabled->Re-enabled->Disabled->Re-enabled	46
Disabled->Re-enabled->Disabled->Deleted	24
Disabled->Re-enabled->Disabled->Re-enabled->Disabled->Re-enabled	1
Total	**1,766**

RQ2: What is the Change Pattern of Disabled Tests?

It takes a longer time to be deleted than to be re-enabled.

Many tests remain disabled for many years.



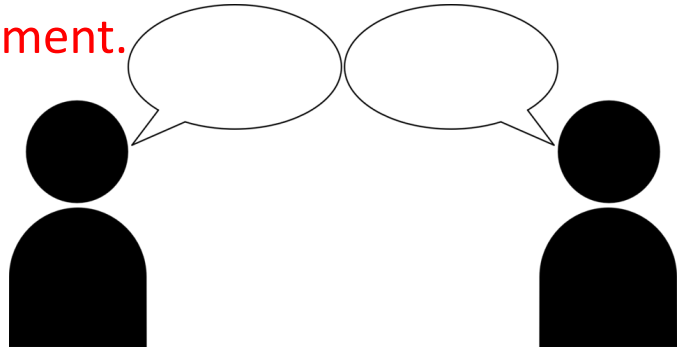
	Time (in days)					
	Min	25%	50%	75%	Max	Mean
Re-enabled	8.4	17.1	39.4	65.9	431.2	61.8
Deleted	4.8	19.9	92.3	222.1	696.6	158.7
Remain Disabled	142.0	508.0	793.2	1096.4	1475.7	797.4

RQ3: Why do developers utilize test disabling practice?

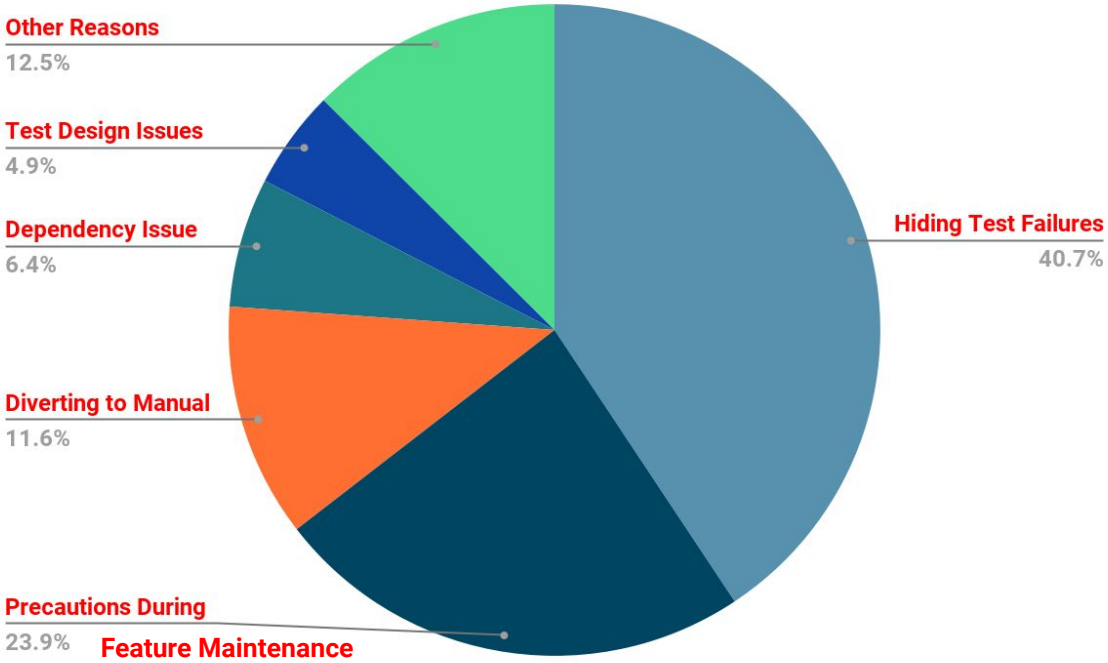
1. Stratified sampling 349 cases out of 3,111 mined cases.

2. First 2 authors study these cases independently.

3. First 2 authors unified the results, solved disagreements and finally achieved a substantial level of agreement.



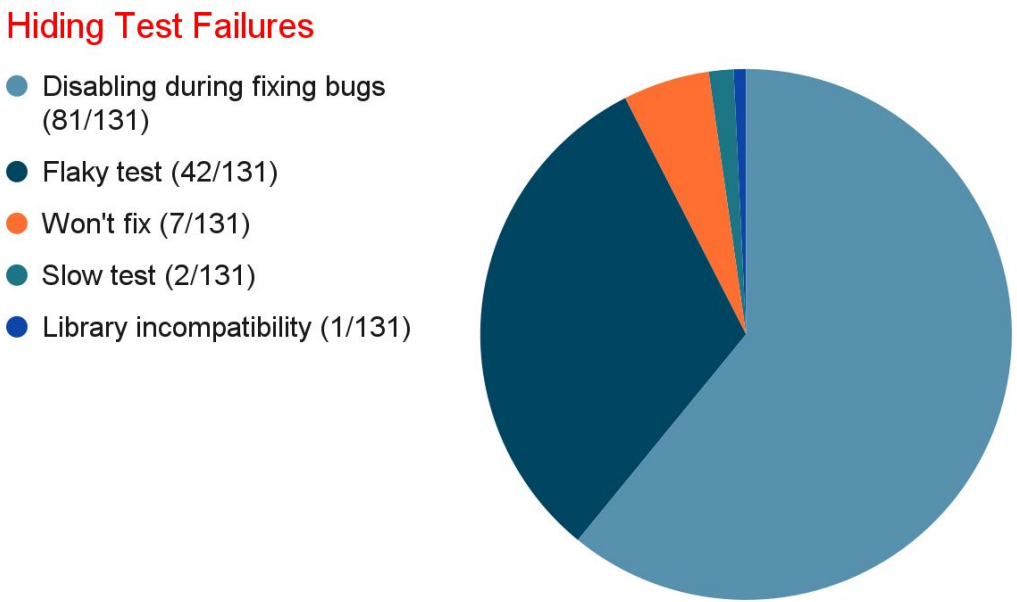
RQ3: Why do developers utilize test disabling practice?



RQ3: Why do developers utilize test disabling practice?

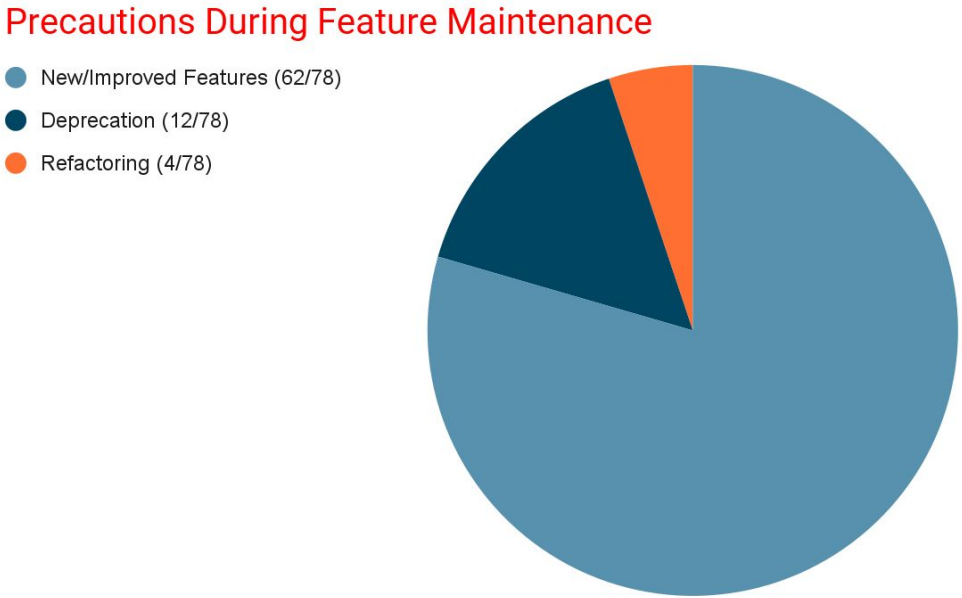
Only 32/131 tests were re-enabled after bug-fixing 😳

Others were either deleted or disabled indefinitely. And most of the deleted tests lack of traceability.

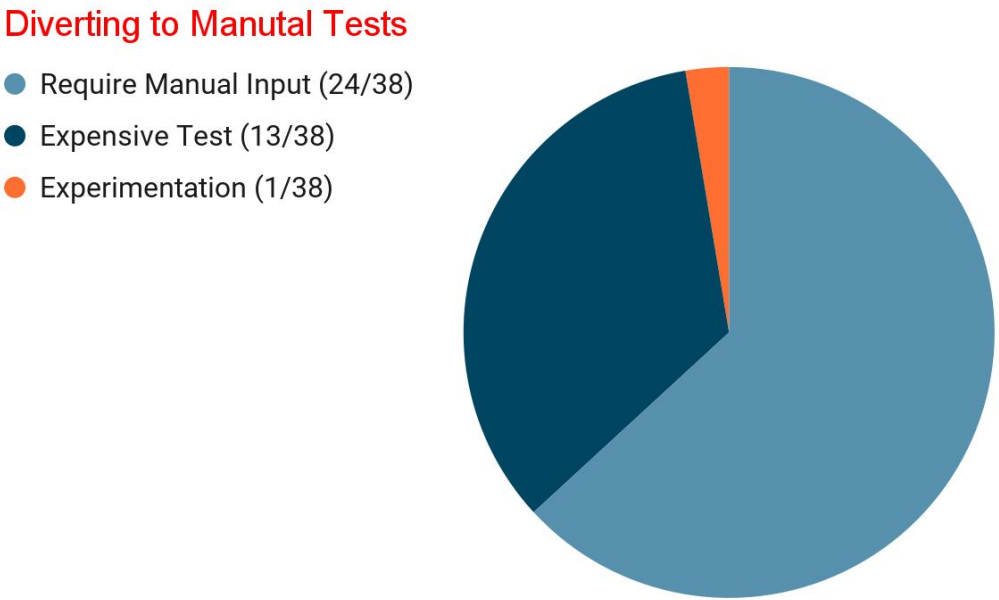


RQ3: Why do developers utilize test disabling practice?

There are still 50% tests remaining disabled after the maintenance.



RQ3: Why do developers utilize test disabling practice?



RQ3: Why do developers utilize test disabling practice?

Dependency Issues:

1. Difficulties in maintaining external dependencies. (17/21)
2. Waiting for functionality update in external dependencies. (4/21)

RQ3: Why do developers utilize test disabling practice?

Test Design Issues:

1. Selective test inheritance. (14/16)
2. Redundant tests. (2/16)

```
24 + public class RabbitMQRequeueQpidTest extends RabbitMQRequeueIntTest {
25 +     @BeforeClass
26 +     public static void startBroker() throws Exception {
27 +         systemLauncher.startup(createQpidSystemConfig());
28 +     }
29 +
30 +     @AfterClass
31 +     public static void stopBroker() {
32 +         systemLauncher.shutdown();
33 +     }
34 +
35 +     @Ignore
36 +     @Override
37 +     public void testNoRequeueHeaderCausesReject() throws Exception {
38 +     }
39 +
40 +     @Ignore
41 +     @Override
42 +     public void testNonBooleanRequeueHeaderCausesReject() throws Exception {
43 +     }
44 +
45 +     @Ignore
46 +     @Override
47 +     public void testFalseRequeueHeaderCausesReject() throws Exception {
48 +     }
49 + }
```

RQ3: Why do developers utilize test disabling practice?

Other reasons:

1. Unknown. (32/41)
2. Obsolete tests. (5/41)
3. By mistake. (4/41)

Implications to Researchers

Many tests are disabled to hide failures and remain disabled for a long period.

Future research should study the **impacts** of the disabled tests **to the software quality**.

Implications to Researchers

Many disabled tests remain disabled for a long time, even if the related issues were marked as solved.

Future research may provide automated approaches to **track the disabled tests** and **remind developers to re-enable them**.

Implications to Researchers

Some tests have to be run manually due to the needs of personal information or manual input.

Future research may consider helping developers automate them.

Implications to Researchers

Many disabled tests are deleted directly because of being obsolete after a long disabled duration.

Future research may study the test obsolescences and the co-evolution between source code and test code.

Implications to Developers

Many tests remain disabled in the code repo for a long period. Though many of them related to Jira issues, most of them are not re-enabled after the issues solved. Non-trivial of them do not have any documentations.

Developers should consider better documentation to **track the disabled tests for good software quality.**

Thank You!

What Are Disabled Tests?

Developers may bypass some tests due to many reasons. There exists many different approaches to do so:

1. Use @Ignore annotation (JUnit4).