Automatically Documenting Unit Test Cases

Boyang Li, Christopher Vendome, Mario Linares-Vasquez, Denys Poshvanyk, and Nicholas A. Kraft*

College of William and Mary, Williamsburg, VA, USA

*ABB Corporate Research Center, Raleigh, NC, USA
Unit Test Case Maintenance

Maintaining unit test cases requires comprehension of the unit test code. Comments in unit test cases are added with the purpose of making the test cases easier to understand.

Challenges:

• Unit test cases may lack comments.

• Source code changes could lead to inconsistencies in unit test case comments.
Our Contributions

- A survey of both open-source and industrial developers
- A mining-based study on a large dataset of C# projects
- An approach to automatically generate natural language descriptions to document the purpose of unit test cases

3 RQs

5 RQs
Research Questions

RQ1. To what extent do unit test cases contain comments?

RQ2. To what extent do developers update unit test case comments?

RQ3. To what extent do developers have difficulty understanding unit test cases?
Methodology

A survey

A mining-based analysis
Methodology

A survey with 212 developers

A mining-based analysis
Methodology

A survey with 212 developers

A mining-based analysis on 53,735 unit tests
Methodology

A survey with 212 developers

A mining-based analysis on 53,735 unit tests
Methodology

A survey with 212 developers on A mining-based analysis 53,735 unit tests
Methodology

A survey with 212 developers

A mining-based analysis on 53,735 unit tests

Results
Data Collection

All C# projects

GitHub

2,209 projects

4,115 developers

565 developers

212 completed

≥ 1
Data Collection

All C# projects

GitHub

≥ 1  ⭐ ≥ 1  ⚪ ≥ 1

Randomly selected

2,209 projects

1,414 projects

53,735 unit test cases
RQ1. To what extent do unit test cases contain comments?

1. How often do you write unit test cases for your project(s)?

- Always
- Fairly Often
- Sometimes
- Rarely
- Never

- Never (176, 83.01%)
- Rarely
- Sometimes
- Fairly Often
- Always (121, 52.83%)

2. How often do you write comments for unit test cases?

- Never (121, 52.83%)
- Rarely
- Sometimes
- Fairly Often
- Always
RQ1. To what extent do unit test cases contain comments?

Survey results

- Always: 176 (83.01%)
- Fairly Often: [Bar chart showing number of responses]
- Sometimes: [Bar chart showing number of responses]
- Rarely: [Bar chart showing number of responses]
- Never: [Bar chart showing number of responses]
RQ1. To what extent do unit test cases contain comments?

Survey results

“Comments need to be maintained which adds complexity to the task.”

“I use very verbose naming of tests to be the documentation, along with meaningful naming of methods and variables used in the test.”
RQ1. To what extent do unit test cases contain comments?

Survey results

- Always: 100 (47.17%)
- Fairly Often: 30
- Sometimes: 20
- Rarely: 30
- Never: 40

Repository mining results

- 3.56% have outer comments (1,914 methods)
- 14.02% have inner comments (7,534 methods)
3. How often do you find outdated comments in unit tests?

4. When you make changes to the unit tests, how often do you comment the changes?

- **Never**
- **Rarely**
- **Sometimes**
- **Fairly Often**
- **Always**

**RQ2. To what extent do developers update unit test case comments?**

- **Always**: 120 (52.36%)
- **Fairly Often**: 117 (55.19%)
- **Sometimes**
- **Rarely**
- **Never**
RQ2. To what extent do developers update unit test case comments?

Survey results

- Always: 117 (55.19%)
- Fairly Often: 100
- Sometimes: 50
- Rarely: 25
- Never: 10

Repository mining results

- Projects with release: 246
- Unit method changes: 1,075,076
- 1.54% have outer comment updates (16,561 changes)
- 15.23% have inner comment updates (163,737 changes)
RQ3. To what extent, do developers have difficulty understanding unit test cases?

5. Maintaining good unit test cases and documentation is important to the quality of a system?

6. How difficult is it to understand a unit test (i.e., identifying focal methods)?
Lessons learnt from RQ1 – RQ3

(i) Documenting unit test cases is not a common practice in Github projects

(ii) Developers do not update comments when changes are made to unit test cases

(iii) Understanding unit test cases is generally not an easy task
UnitTestScribe

- Natural language descriptions
- Focal methods
- Assertion description
- Internal data dependencies
**UnitTestScribe**

- Natural language descriptions
- **Focal methods**
- Assertion description
- Internal data dependencies
UnitTestScribe

- Natural language descriptions
- Focal methods
- Assertion description
- Internal data dependencies
UnitTestScribe

- Natural language descriptions
- Focal methods
- Assertion description
- Internal data dependencies
UnitTestScribe Architecture

1. Unit Test Cases Detector
2. Focal Method Detector
3. SWUM.NET
4. Program Slicing Analyzer
5. Description Generator
1) Unit Tests Detector

- Unit test annotations
  
  [Test], [TestMethod], [TestCase], [Fact] etc.

- The test case detector detects unit tests based on the annotations
2) Focal Method Detector

- Focal method detection (Ghafari et al. SCAM’15)

- The last mutator/collaborator function(s) that modifies the variable(s) examined in a given assertion.
3) SWUM.NET

- SWUM (Hill et al. ICSE’09) is an approach that can extract and tag NL phrases for source code.

- SWUM.NET by ABB

```
public SimpleApiClientFactory {
    public void ClearFromCache(ISimpleApiClient c) {
        ....
    }
}
```

“Clear simple api client from cache”
4) Variable Slicing Analyzer

- Internal data dependencies for the variables in assertions
5) Description Generator

- Using predefined templates
- Documentations in HTML format
UnitTestScribe HTML Report

### UnitTestScribe report - Sando

#### Methods
- TestFileParserTest, ParseXMLFile()
- TestFileParserTest, ParseXMLFile2()
- XAMLFileParserTest, ParsePerformanceTests()
- XAMLFileParserTest, TestLengthOfEachXamlElement
- UnitTests, CamelCaseSplittersTest, SplitTest
- UnitTests, CamelCaseSplittersTest, SplitBadCamelC
- UnitTests, CamelCaseSplittersTest, SplitTestLowercase
- UnitTests, CamelCaseSplittersTest, SplitTestUppercase
- UnitTests, CamelCaseSplittersTest, SplitTestUpperCase
- UnitTests, SwumDataRecordTests, QueryRecommend
- UnitTests, SwumDataRecordTests, TestRoundTrip
- UnitTests, SwumManagerTests, TestAddSourceFile()
- UnitTests, SwumManagerTests, TestAddSourceFile_C
- UnitTests, SwumManagerTests, TestAddSourceFile_D
- UnitTests, SwumManagerTests, TestCacheRoundTrip
- UnitTests, SwumManagerTests, TestCurrentConcurrentRequests
- UnitTests, SwumManagerTests, TestReadingRealCelsius
- UnitTests, SwumManagerTests, TestRemoveSourceFile
- UnitTests, SwumManagerTests, TestUpdateSourceFile
- JUnitTests, CodeSearcherFixture, PerformBasicSearch
- JUnitTests, CodeSearcherFixture, PerformBasicSearch
- JUnitTests, CodeSearcherFixture, TestCreateCodeSearch
- JUnitTests, CodeSearcherFixture, TestCreateCodeSearch
- HighlightConverterTests, HandleWithHeadingAndTable
- HighlightConverterTests, HighlightedWordAtBeginningTri
- HighlightConverterTests, MakeSureEmptyLinesAreHand
- ViewControllerTests, RemoveEscapeFromQueryy
- translatorTest, Translator_GetTranslationReturnsValidT


This unit test case method tests parse cpp constructor test.

4. This unit test case includes following focal methods:
   (1) var elements = parser.Parse("TestFiles\Event.txt")
   This focal method is related to assertions at line 180
   This focal method is related to assertions at line 181

This unit test case includes following focal methods:
(1) var elements = parser.Parse("WeirdStructFile")
   This focal method is related to assertions at line 150
   This focal method is related to assertions at line 151

This unit test case includes following focal methods:
(1) elements.Count == 2 is true.
   elements is obtained from
   1) variable WeirdStructFile through slicing path.
(2) structElement is not null.
   structElement is obtained from
   1) variable pe through slicing path.
(3) "LangMenuItems" is equal to structElement.Name.
(4) structId is equal to methodElement.ClassId.
   structId is obtained from
   1) variable pe through slicing path.
(5) hasStruct is true.
public AddWordsSeveralTimes()
{
    int listLength = 20;
    int coocurrenceCount = 3;
    var words = GenerateRandomWordList(listLength);
    for (int i = 0; i < coocurrenceCount; i++)
    {
        matrix.HandleCoOcurrentWordsSync(words);
    }
    for (int i = 0; i < listLength - 1; i++)
    {
        var word1 = words.ElementAt(i);
        var word2 = words.ElementAt(i + 1);
        var count = matrix.GetCoOccurrenceCount(word1, word2);
        Assert.IsTrue(count > 0);
    }
}

- Natural language descriptions
- Focal methods
- Assertion description
- Internal data dependencies
public AddWordsSeveralTimes()
{
    int listLength = 20;
    int coocurrenceCount = 3;
    var words = GenerateRandomWordList(listLength);
    for (int i = 0; i < coocurrenceCount; i++)
    {
        matrix.HandleCoOcurrentWordsSync(words);
    }
    for (int i = 0; i < listLength - 1; i++)
    {
        var word1 = words.ElementAt(i);
        var word2 = words.ElementAt(i + 1);
        var count =
            matrix.GetCoOccurrenceCount(word1, word2);
        Assert.IsTrue(count > 0);
    }
}
public AddWordsSeveralTimes()
{
    int listLength = 20;
    int cooccurrenceCount = 3;
    var words = GenerateRandomWordList(listLength);
    for (int i = 0; i < cooccurrenceCount; i++)
        matrix.HandleCoOcurrentWordsSync(words);
    for (int i = 0; i < listLength - 1; i++)
    {
        var word1 = words.ElementAt(i);
        var word2 = words.ElementAt(i + 1);
        var count = matrix.GetCoOccurrenceCount(word1, word2);
        Assert.IsTrue(count > 0);
    }
}
public AddWordsSeveralTimes()
{
    int listLength = 20;
    int coocurrenceCount = 3;
    var words = GenerateRandomWordList(listLength);
    for(int i = 0; i < coocurrenceCount; i++)
    {
        matrix.HandleCoOcurrentWordsSync(words);
    }
    for(int i = 0; i < listLength - 1; i++)
    {
        var word1 = words.ElementAt(i);
        var word2 = words.ElementAt(i + 1);
        var count = matrix.GetCoOccurrenceCount(word1, word2);
        _Assert.IsTrue(count > 0);
    }
}

- Natural language descriptions
- Focal methods
- Assertion description
- Internal data dependencies
Empirical Study

• We evaluate UnitTestScribe on four open source systems:
  
  1) SrcML.NET, 2) Sando, 3) Glimpse, and 4) Google-api-dotnet

• We then measure the quality of descriptions generated by UnitTestScribe according to three criteria (Sridhara et al. ASE’10; Moreno et al. ICPC’13; Cortes-Coy et al. SCAM’14):

  1) Completeness  2) Conciseness  3) Expressiveness
Research Questions

RQ4. How complete are the descriptions generated by UnitTestScribe?

RQ5. How concise are the descriptions generated by UnitTestScribe?

RQ6. How expressive are the descriptions generated by UnitTestScribe?

RQ7. How important are focal methods and program slicing for understanding unit test cases?

RQ8. How well can UnitTestScribe help developers understand unit test cases?
Empirical Study

• Group 1: ABB

  7 researchers/interns from the ABB corporate research center

  2 systems from ABB

• Group 2:

  19 students/researchers from universities

  2 systems from Github
Empirical Study

• Randomly selected 5 unit tests for each system

• Each participant evaluates 2 systems

• For each attribute (Completeness, Conciseness, and Expressiveness)
  - We had $5 \times 2 \times 7$ (participants) = 70 answers in group 1
  - We had $5 \times 2 \times 19$ (participants) = 190 answers in group 2
RQ4. How complete are the descriptions generated by UnitTestScribe?

<table>
<thead>
<tr>
<th>Completeness</th>
<th>Group 1 (Answers)</th>
<th>Group 2 (Answers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does not miss any important info</td>
<td>33 (47.14%)</td>
<td>132 (69.47%)</td>
</tr>
<tr>
<td>Misses some important info</td>
<td>28 (40.00%)</td>
<td>50 (26.32%)</td>
</tr>
<tr>
<td>Misses the majority of the important info</td>
<td>9 (12.86%)</td>
<td>8 (4.21%)</td>
</tr>
</tbody>
</table>

Overall, UnitTestScribe is able to cover most essential information in most of the cases.
RQ5. How concise are the descriptions generated by UnitTestScribe?

<table>
<thead>
<tr>
<th>Conciseness</th>
<th>Group 1 (Answers)</th>
<th>Group 2 (Answers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains no redundant info</td>
<td>36 (51.43%)</td>
<td>100 (52.63%)</td>
</tr>
<tr>
<td>Contains some redundant info</td>
<td>25 (35.71%)</td>
<td>77 (40.53%)</td>
</tr>
<tr>
<td>Contains a lot of redundant info</td>
<td>9 (12.86%)</td>
<td>13 (6.84%)</td>
</tr>
</tbody>
</table>

Overall, UnitTestScribe can generate descriptions with little redundant information.
RQ6. How *expressive* are the descriptions generated by UnitTestScribe?

<table>
<thead>
<tr>
<th>Expressiveness</th>
<th>Group 1 (Answers)</th>
<th>Group 2 (Answers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is easy to read</td>
<td>43 (61.43%)</td>
<td>114 (60.00%)</td>
</tr>
<tr>
<td>Is somewhat readable</td>
<td>16 (22.86%)</td>
<td>53 (27.89%)</td>
</tr>
<tr>
<td>Is hard to read</td>
<td>11 (15.71%)</td>
<td>23 (12.11%)</td>
</tr>
</tbody>
</table>

(UnitTestScribe) descriptions are easy to read in most of the cases.
RQ7. How important are focal methods and program slicing for understanding unit test cases?

Identifying **focal method** would help developers to understand the unit test.

<table>
<thead>
<tr>
<th></th>
<th>Group 1 (Answers)</th>
<th>Group 2 (Answers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>7 (100.00%)</td>
<td>17 (89.00%)</td>
</tr>
<tr>
<td>No</td>
<td>0 (0%)</td>
<td>2 (11%)</td>
</tr>
</tbody>
</table>

Identifying **slicing path** would help developers to understand the unit test.

<table>
<thead>
<tr>
<th></th>
<th>Group 1 (Answers)</th>
<th>Group 2 (Answers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>6 (86.00%)</td>
<td>13 (68.00%)</td>
</tr>
<tr>
<td>No</td>
<td>1 (14%)</td>
<td>6 (32%)</td>
</tr>
</tbody>
</table>

Focal methods and program slicing are important.
RQ7. How well can UnitTestScribe help developers understand unit test cases?

<table>
<thead>
<tr>
<th>Are UnitTestScribe description useful for understanding the unit tests in the system?</th>
<th>Group 1 (Answers)</th>
<th>Group 2 (Answers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>4 (57%)</td>
<td>17 (89.00%)</td>
</tr>
<tr>
<td>No</td>
<td>3 (43%)</td>
<td>2 (11%)</td>
</tr>
</tbody>
</table>

For developers who are not familiar with an application, UnitTestScribe is very useful for understanding unit test methods.
Summary RQ4 – RQ8

- Completeness
- Conciseness
- Expressiveness
- Importance of focal methods and program slicing
- Usefulness
Acknowledgment

• ABB colleagues: Vinay Augustine and Patrick Francis

• Anonymous reviewers

• All students, developers and researchers who responded to our survey
Summary

Lessons learnt from RQ1 – RQ3

(i) Documenting unit test cases is not a common practice

(ii) Developers do not update comments when changes are done to unit test cases

(iii) Understanding unit test cases is not generally an easy task

http://www.cs.wm.edu/semeru/data/ICST16-UnitTestScribe/
Summary

Methodology

- A survey with 212 developers
- A mining-based analysis on 53,735 unit tests
  - Results
Summary

Methodology

A survey with 212 developers → Results

A mining-based analysis on 53,735 unit tests → Results

Lessons learnt from RQ1 – RQ3

(i) Documenting unit test cases is not a common practice

(ii) Developers do not update comments when changes are done to unit test cases

(iii) Understanding unit test cases is not generally an easy task

http://www.cs.wm.edu/semuru/data/ICST16-UnitTestScribe/
Summary

Methodology

A survey with 212 developers on a mining-based analysis of 53,735 unit tests results in the following:

Lessons learnt from RQ1 – RQ3

(i) Documenting unit test cases is not a common practice

(ii) Developers do not update comments when changes are done to unit test cases

(iii) Understanding unit test cases is not generally an easy task

UnitTestScribe Architecture

http://www.cs.wm.edu/semeru/data/ICST16-UnitTestScribe/
Summary

Methodology

A survey with 212 developers on 53,735 unit tests led to the following results:

- A mining-based analysis
- 53,735 unit tests
- Results

Lessons learnt from RQ1 – RQ3

(i) Documenting unit test cases is not a common practice

(ii) Developers do not update comments when changes are done to unit test cases

(iii) Understanding unit test cases is not generally an easy task

UnitTestScribe Architecture

(Unit Test Cases Detector)

SWUM.NET

Program Slicing Analyzer

Description Generator

Empirical Study

- Group 1: ABB
  - 7 researchers/interns from ABB corporate research center
  - 2 systems from ABB

- Group 2: University
  - 19 students/researchers from universities
  - 2 systems from Github

http://www.cs.wm.edu/semeru/data/ICST16-UnitTestScribe/
Summary

Methodology

A survey on 212 developers
A mining-based analysis of 53,735 unit test cases

Lessons learnt from RQ1 – RQ3

1. Documenting unit test cases is not a common practice
   - Documentation
2. Developers sometimes update comments when unit test cases
   - Automation
3. Writing unit test cases is not generally an

Summary RQ4 – RQ8

- Completeness
- Concision
- Expressiveness
- Importance of focal methods and program slicing
- Usefulness

UnitTestScribe Architecture

Unit Test Cases Detector
Focal Method Detector
SWUM.NET
Description Generator
Program Slicing Analyzer

Study

- 2 systems from ABB
- 2 systems from Github

- Group 2: 🧠
  - 19 students/researchers from universities

http://www.cs.wm.edu/semeru/data/ICST16-UnitTestScribe/
BACKUP Slides...
UnitTestScribe Architecture

Source Codes → Unit Test Cases Detector → Unit Test Cases

Stereotype Analyzer → SWUM.NET  → Program Slicing Analyzer

Focal Method Detector → SWUM.NET Description → Variable Slicing Information

Focal Methods Information → Description Generator → Unit Test Case Documentation

Templates
RQ7. How well can UnitTestScribe help developers understand unit test cases?

<table>
<thead>
<tr>
<th>Are UnitTestScribe description useful for understanding the unit tests in the system?</th>
<th>Group 1</th>
<th>Group 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>4 (57%)</td>
<td>17 (89.00%)</td>
</tr>
<tr>
<td>No</td>
<td>3 (43%)</td>
<td>2 (11%)</td>
</tr>
</tbody>
</table>

“I see the SrcML.NET system, I know what’s going on. Its usefulness drops off if you’re talking to someone experienced with the code base, though. So I suppose this depends on who this is aimed at.” - From a participant in group 1

“It is useful if I am not familiar with an application.” - From a participant in group 2

For developers who are not familiar with an application, UnitTestScribe is very useful for understanding unit test methods.
<table>
<thead>
<tr>
<th>Type</th>
<th>Category</th>
<th>Description</th>
<th>Modified rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getter</td>
<td>Accessor</td>
<td>Returns the value of a data member</td>
<td>No class field is changed &amp;&amp; Return type is not void &amp;&amp; Only return one class field.</td>
</tr>
<tr>
<td>Predicate</td>
<td>Accessor</td>
<td>Returns a Boolean result based on a data member(s)</td>
<td>No data member is changed &amp;&amp; Return type is bool &amp;&amp; Do not directly return any data member</td>
</tr>
<tr>
<td>Property</td>
<td>Accessor</td>
<td>Returns information about a data member</td>
<td>No data member is changed &amp;&amp; Return type is not bool or not void. &amp;&amp; Do not directly return any data member</td>
</tr>
<tr>
<td>Setter</td>
<td>Mutator</td>
<td>Changes the value of a data member</td>
<td>Only 1 data member is changed &amp;&amp; Return type is void or 0/1</td>
</tr>
<tr>
<td>Command</td>
<td>Mutator</td>
<td>Executes complex changes on data members</td>
<td>More than 1 class field is changed &amp;&amp; Return type is void or 0/1</td>
</tr>
<tr>
<td>Collaborator</td>
<td>Collaborator</td>
<td>Works on objects of classes different from the method</td>
<td>At least one of the method’s parameters or local variables is an object</td>
</tr>
<tr>
<td>Factory</td>
<td>Creator</td>
<td>Creates an object and returns it</td>
<td>Not returns primitive type Local &amp;&amp; (A local variable is instantiated and returned</td>
</tr>
</tbody>
</table>

Accessor

Predicate methods

Setter methods

Classe.DropClass(Student st)

Student.SetName(String name)

Student.IsGraduate()
• Focal method detection (Ghafari et al. SCAM’15)

• The last mutator/collaborator function(s) that modifies the variable(s) used as arguments to a given assertion call.

```java
public void foo(){
    Student a = new Student();
a.setName("myname");
    assertTrue(a.Name == "myname");
}
```

```java
public void foo(){
    Student a = new Student();
a.setAge(28);  
    int age = a.getAge();
    assertTrue(age == 28);
}
```

• Kamimura and Murphy ICPC’13. The approach identified the focal method based on how many times the test method invokes the function.
• SWUM (Hill et al.) is an approach which can extract and tag NL phrases for source codes.

• The SWUM.NET tool implemented by ABB in C#