DDU

Master's Thesis

Aaron Ang

DDU

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Aaron Ang
born in Amstelveen, The Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

PARC, a Xerox company
3333 Coyote Hill Road
Palo Alto, CA 94304
www.parc.com

©2017 Aaron Ang. All rights reserved.

DDU

Author: Aaron Ang
Student id: 4139194
Email: `a.w.z.ang@student.tudelft.nl`

Abstract

ABSTRACT

Thesis Committee:

Chair:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
Company supervisor:	Prof. Dr. R. Maranhao, University of Lisbon

Preface

This is where you thank people for helping you etc.

Aaron Ang
Delft, The Netherlands
August 21, 2017

Contents

Preface iii

Contents v

List of Figures vii

1 Introduction 1
- 1.1 Problem Definition . . . 2
- 1.2 Goal . . . 2
- 1.3 Structure of Report . . . 2

2 Background 3
- 2.1 Spectrum-Based Reasoning (SBR) . . . 3
- 2.2 Evaluation of Diagnosis . . . 3
- 2.3 Diagnosability . . . 4
- 2.4 Error detection . . . 4

3 Research Questions 5

4 DDU vs. Diagnosability 7
- 4.1 Experimental Setup . . . 7
- 4.2 Experimental Results . . . 9

5 DDU vs. Error Detection 15

6 Conclusion 17

Bibliography 19

List of Figures

4.1 An activity matrix $\mathcal{A}$ is generated for a particular class. Then, 10 fault candidates of cardinality 2 are generated with a corresponding activity matrix $\mathcal{A}_k$. For each generated matrix, we perform fault diagnosis with *BARINEL* resulting in diagnostic report $\mathcal{D}_k$ and compute the wasted effort. Finally, we compute the average wasted effort. This process is repeated 10 times. 8
4.2 Two activity matrices $\mathcal{A}$ and $\mathcal{A}'$ are generated for a particular class based on two different test suites. We generate 10 fault candidates of cardinality 2 and accordingly generate 10 activity matrices. Then, we use *BARINEL* to perform fault diagnosis and compute the wasted effort. . . 8
4.3 Scatterplot of density, diversity, uniqueness, and DDU against effort. . . 10
4.4 11

Chapter 1

Introduction

Software systems are complex and error-prone, likely to expose failures to the end user. When a failure occurs, the developer has to debug the system to eliminate the failure. This debugging process can be described in three phases [4]. In the first phase, the developer has to pinpoint the fault, also known as the root cause, in code that causes the failure. In the second phase, the developer has to develop an understanding of the root cause and its context. Finally, in the third phase, the developer has to implement a patch that corrects the behavior of the system. This process is time-consuming and can account for 30% to 90% of the software development cycle [7, 2, 3].

Traditionally, developers use four different approaches to debug a software system, namely program logging, assertions, breakpoints and profiling [8]. Program logging is the act of inserting *print* statements in the code to observe program state information during execution. Assertions are constraints that can be added to a program that have to evaluate to true during execution time. Breakpoints allow the developer to pause the software system during execution, and observe and modify variable values. Profiling is used to perform runtime analysis and collect metrics on, for example, execution speed and memory usage. These techniques provide an intuitive approach to localize the root cause of a failure, but, as one might expect, are less effective in the massive size and scale of software systems today.

Therefore, in the last decades a lot of research has been performed on improving and developing *advanced* fault localization techniques [8] such that they are applicable to the software systems of today. Specifically, a prominent fault localization technique is spectrum-based fault localization (SBFL). SBFL techniques pinpoint faults in code based on execution information of a program, also known as a program spectrum [6]. It does this by outputting a list of suspicious components, for example statements or methods, ranked by their suspiciousness. Intuitively, if a statement is executed primarily during failed executions, then this statement might be assigned a higher suspiciousness score. Similarly, if a statement is executed primarily during successful executions, then this statement might be assigned a lower suspiciousness score.

While SBFL techniques are promising for debugging purposes, these techniques are dependent on the quality of a test suite. Currently, test suites are optimized with respect to adequacy measurements that focus on error detection, e.g. branch coverage, line coverage. However, Perez *et al.* [5] show evidence that optimizing a test suite with respect to DDU — a metric to quantify the test suite's diagnosability — improves the diagnostic performance of SBFL by 34% compared to a test suite optimized with respect to branch coverage. The goal of DDU is to capture diagnosability and to serve as a complementary metric to code coverage for developers to use to improve the test suite's diagnosability.

1.1 Problem Definition

Currently, when the DDU is computed for a given test suite, its value is in the domain $[0,1]$, where 0 suggests that the test suite's diagnosability is low, and 1 suggests that the test suite's diagnosability is high. The problem with this value is that the developer does not know how to extend or update the test suite given a DDU value. For example, when the test suite's DDU is equal to 0.1, the developer does not know how to write tests that improve the DDU. In other words, time spent on software debugging cannot be reduced using DDU because its practical implications are unclear to the developer.

1.2 Goal

Although DDU is currently not usable in practice, Perez *et al.* [5] have shown that optimizing a test suite with respect to DDU can yield a 34% gain in diagnostic performance using SBFL. In addition, having a test suite with a high diagnosability could possibly reduce the time spent debugging because the fault is easier to find manually. Therefore, the goal of this thesis is to find ways to make DDU usable in practice. In other words, we explore possibilities to convey DDU to the developer such that the developer knows what kind of tests to write to improve the system's diagnosability. To be able to

1.3 Structure of Report

The structure of this report is as follows. TODO: Update once all chapters are done.

Chapter 2

Background

In this chapter, we discuss topics that are relevant to understanding the following chapters. First, we discuss spectrum-based reasoning, which is used in the experiments to compute the diagnostic performance. Second, we discuss the metric used to evaluate the diagnostic performance of spectrum-based fault localization techniques. Then, we explain the definition of diagnosability and diagnosability metrics. Finally, we briefly discuss error detection.

2.1 Spectrum-Based Reasoning (SBR)

TODO: Describe how SR works and Barinel.

Spectrum-based reasoning is a spectrum-based fault localization technique that leverages a Bayesian reasoning framework to diagnose fault candidates that could potentially be the root cause for a given software failure [1]. In SBR, we define a finite set $\mathcal{C} = \langle c_1, c_2, \ldots, c_M \rangle$ of M system components, and the finite set $\mathcal{T} = \langle t_1, t_2, \ldots, t_N \rangle$ of N system transactions, such as test executions. The outcomes of all system transactions are defined as an error vector $e = \langle e_1, e_2, \ldots, e_N \rangle$, where $e_i = 1$ indicates that transaction t_i has failed and $e_i = 0$ otherwise. To keep track of which system components were executed during which system transactions, we construct a $N \times M$ activity matrix $\mathcal{A}$, where $\mathcal{A}_{ij} = 1$ indicates that component c_j was hit during transaction t_i. The pair $(\mathcal{A}, e)$ is also known as a program spectrum, which was first coined by Reps *et al.* [6].

SBR distinguishes itself from SBFL techniques by leveraging a reasoning framework. More specifically, the diagnostic report is generated by reasoning about the program spectrum instead of using a so-called similarity coefficient.

2.2 Evaluation of Diagnosis

TODO: Describe wasted effort

2.3 Diagnosability

TODO: Explain concept and definition of diagnosability: the property of faults to be easily and precisely located

2.3.1 Diagnosability Metric: Entropy

2.3.2 Diagnosability Metric: DDU

Density

Diversity

Uniqueness

2.4 Error detection

Chapter 3

Research Questions

To explore possibilities to convey DDU to the developer with usability in mind, we define four research questions that are relevant to investigate. Note that the nature of this study will be exploratory because little research has been performed on software diagnosability. Moreover, DDU is a metric that has been proposed recently [5] and, therefore, to the best of our knowledge, no research has investigated this metric.

RQ1: What kind of values do density, diversity, uniqueness, and DDU take on?

To make recommendations based on DDU, it is necessary to obtain a better understanding of DDU and its individual components: density, diversity, and uniqueness. Specifically, we are interested in what common values are for DDU and its individual components in software systems.

RQ2: What is the relation between density, diversity, uniqueness, and DDU and diagnosability?

We would like to validate that DDU and diagnosability are positively correlated, i.e. the higher DDU, the better the diagnosability, and vice versa. This question is important to answer because DDU was proposed to quantify the diagnosability. Therefore, answering this question will function as a complementary study to Perez *et al.*'s work [5].

RQ3: What is the relation between density, diversity, uniqueness, and DDU and test coverage?

The intention of test coverage is to optimize for error detection. Perez *et al.* propose DDU as a complementary metric to test coverage [5] because DDU is meant to capture the diagnosability and not error detection. However, if there is a strong correlation between DDU and test coverage, then DDU could possibly replace test coverage as a test adequacy metric, which is a use case that the authors have not thought of. Additionally, assuming that DDU is positively correlated with diagnosability and test coverage is representative for error detection, answering this question will give us a better understanding on the relation between diagnosability and error detection.

RQ4: What kinds of tests have positive or negative effects on density, diversity, uniqueness, and DDU?

Chapter 4

DDU vs. Diagnosability

In prior work Perez *et al.* [5] show that optimizing test suite generation with respect to DDU results in better fault diagnosis. Therefore, in this chapter, we perform experiments to verify the correlation between DDU and diagnosability, that is, answering RQ2. We do this by discussing the experimental setup followed by the experimental results.

RQ2: What is the relation between density, diversity, uniqueness, and DDU and diagnosability?

4.1 Experimental Setup

First, we have to define the subjects of interest that will be used during the experiments. For the experiments, we use the open source projects Commons Codec, Commons Compress, and Commons Math, which were also used in prior work by Alex *et al.* [5]. In addition, we include the open source projects Guice and Jsoup due to their popularity on GitHub; both roughly have 5000 stars.

To test the correlation between DDU and diagnosability, we generate 10 artificial multiple components faults of cardinality 2 for each class that has at least 8 components, i.e. method branches. For each generated fault set, we construct an activity matrix. We determine for each test that exercises the faulty components whether it is failing according to an *oracle quality probability* of 0.75, which was also used in prior work [5]. Then, for each generated activity matrix, we use *STACCATO* to generate fault candidates and *BARINEL* to generate a diagnosis report. Based on the diagnosis report we compute the wasted effort which is a measurement for diagnosability. To account for randomness of generating fault sets, we repeat this process 10 times. Note that we do not generate single component faults because in this case the optimal matrix for diagnosability is a diagonal activity matrix, i.e. each component each tested individually by a unit test. Additionally, *STACCATO* can sometimes take hours or days to generate fault candidates. Hence, we discard classes when gener-

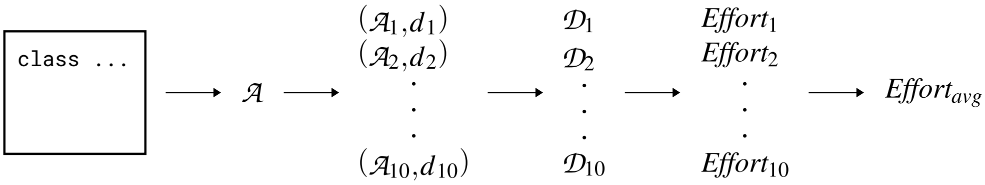

class ...
$\mathcal{A}$
$(\mathcal{A}_1, d_1)$
$(\mathcal{A}_2, d_2)$
$(\mathcal{A}_{10}, d_{10})$
$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_{10}$
$Effort_1$
$Effort_2$
$Effort_{10}$
$Effort_{avg}$

Figure 4.1: An activity matrix $\mathcal{A}$ is generated for a particular class. Then, 10 fault candidates of cardinality 2 are generated with a corresponding activity matrix $\mathcal{A}_k$. For each generated matrix, we perform fault diagnosis with *BARINEL* resulting in diagnostic report $\mathcal{D}_k$ and compute the wasted effort. Finally, we compute the average wasted effort. This process is repeated 10 times.

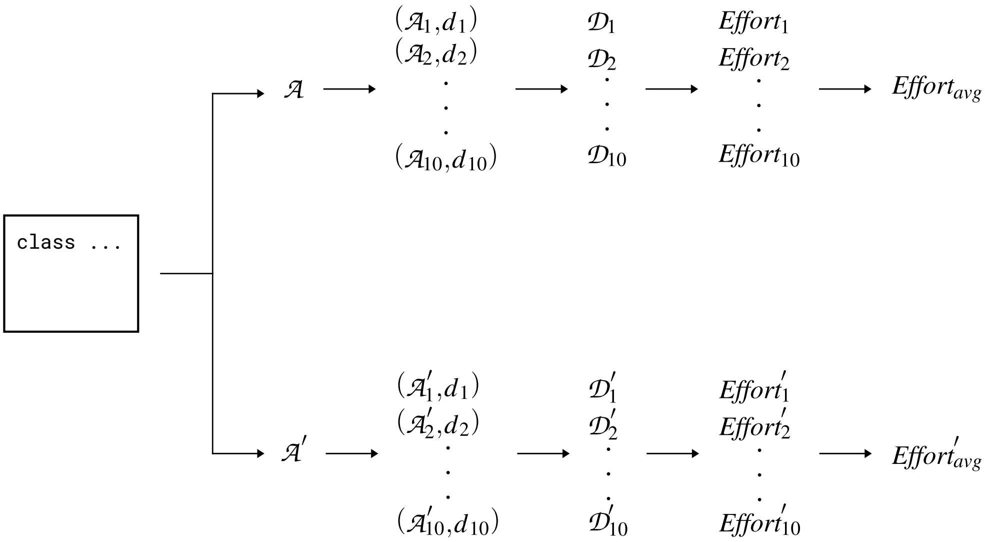

class ...
$\mathcal{A}$
$(\mathcal{A}_1, d_1)$
$(\mathcal{A}_2, d_2)$
$(\mathcal{A}_{10}, d_{10})$
$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_{10}$
$Effort_1$
$Effort_2$
$Effort_{10}$
$Effort_{avg}$
$\mathcal{A}'$
$(\mathcal{A}'_1, d_1)$
$(\mathcal{A}'_2, d_2)$
$(\mathcal{A}'_{10}, d_{10})$
$\mathcal{D}'_1$
$\mathcal{D}'_2$
$\mathcal{D}'_{10}$
$Effort'_1$
$Effort'_2$
$Effort'_{10}$
$Effort'_{avg}$

Figure 4.2: Two activity matrices $\mathcal{A}$ and $\mathcal{A}'$ are generated for a particular class based on two different test suites. We generate 10 fault candidates of cardinality 2 and accordingly generate 10 activity matrices. Then, we use *BARINEL* to perform fault diagnosis and compute the wasted effort.

ating fault candidates takes longer than 10 seconds; this resulted in 16 classes being discarded.

In the construction of the activity matrix we use the branch granularity, that is, every component of the activity matrix represents a method branch; this granularity is also used by Perez *et al.* [5]. To construct the activity matrix of a class we use Perez' DDU Maven plugin[1] using the `basicblock` granularity, which represents branch granularity. The steps after the obtaining the activity matrix in Figure 4.1 are performed using Python scripts[2].

This experiment is different from Perez *et al.*'s work because we do not improve the DDU of a fixed system. Specifically, in Perez *et al.*'s study, the authors improved the DDU of a fixed system under test by generating new test cases using *EvoSuite*. However, in this experiment, we compute the DDU for each class and measure for each class its diagnosability using the aforementioned approach. Essentially, the difference is that we do not improve the DDU of a fixed system but we are simply measuring the DDU.

---

[1] `https://github.com/aperez/ddu-maven-plugin`

[2] `https://github.com/aaronang/ddu`

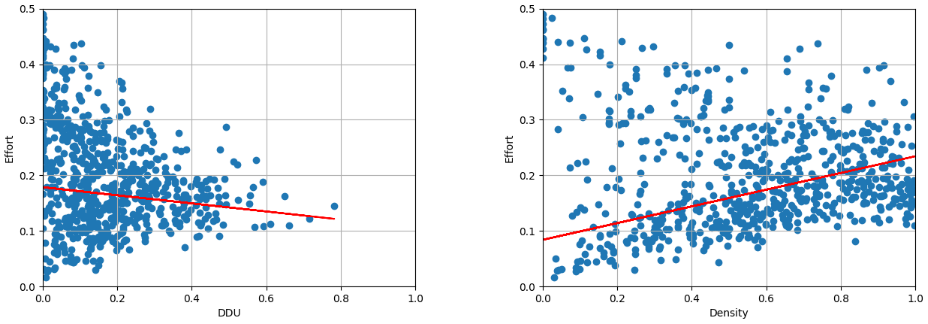
For this reason, we perform another experiment where we generate two test suites for each class with at least 8 components and at least 10 unit tests. In addition, we perform the same experiment but for all classes with at least 8 components. We generate two test suites by using all test cases and 50% of the test cases. For both test suites we compute the DDU and randomly generate 10 multiple components faults of cardinality 2 to compute the wasted effort. Similar to previous experiment we perform this process 10 times to account for randomness of generating fault sets. The intuition behind this experiment is when we improve the DDU of a fixed system, its diagnosability should improve too. The setup of this experiment is illustrated in Figure 4.2.

4.2 Experimental Results

In the first experiment, we measure for each class the density, diversity, uniqueness, DDU, and diagnosability. The results of this experiment are shown in Figure 4.3. Note that in Figure 4.3 the population comprises all classes of all projects. Each datapoint in Figure 4.3 represents a class for which 100 fault candidates are generated in (potentially overlapping) sets of 10 fault candidates as described in Figure 4.1. We observe the that there is a positive correlation between density and effort, a negative correlation between diversity and effort, a negative correlation between uniqueness and effort, and a statistically non-significant weak correlation between DDU and effort.

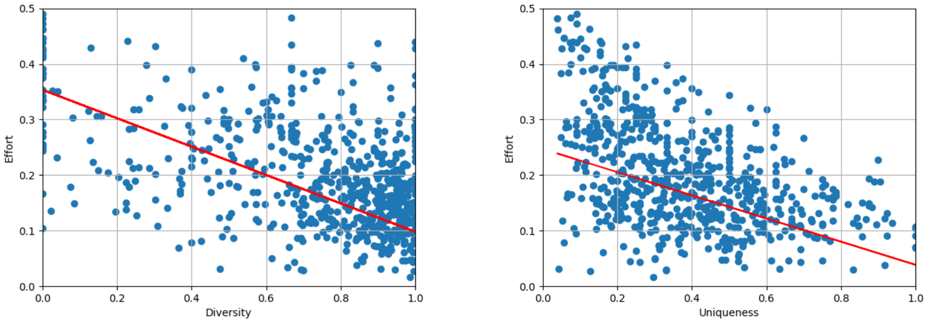
To investigate the relations between these metrics in more detail, we display the correlation values per project in Table 4.1. For three projects we can say with 95% confidence that density is correlated with effort. However, the Pearson correlation for Commons Compress is negative while the Pearson correlation values for Commons Math and Commons Codec are positive. Hence, the results show no strong evidence that density is correlated with effort. Regarding diversity and uniqueness, we observe in Table 4.1 that both metrics are negatively correlated to effort, and that the correlation values in 4 out of 5 projects are statistically significant. Regarding DDU, for two projects the results show statistical significance that DDU is negatively correlated to effort. However, for three projects there is no evidence that DDU is correlated to effort. Therefore, there is no strong evidence that DDU is negatively correlated to effort.

In the second experiment, we generate two test suites for a given class: a test suite with 50% of the test cases enabled, and a test suite with 100% of the test cases enabled. This naturally results in two test suites with two different DDU values. For



(a) DDU, $r = -0.046$, $p = 0.224$.

(b) Density, $r = 0.183$, $p < 0.001$.



(c) Diversity, $r = -0.326$, $p < 0.001$.

(d) Uniqueness, $r = -0.120$, $p < 0.001$.

Figure 4.3: Scatterplot of density, diversity, uniqueness, and DDU against effort.

Table 4.1: Correlation between density, diversity, uniqueness, DDU, and effort.

Subject	Size	Pearson correlation / Correlation p-value			
		Density	Diversity	Uniqueness	DDU
Commons Codec	34	0.63 **5.880×10^{-5}**	-0.33 0.057	-0.65 **3.713×10^{-5}**	-0.23 0.197
Commons Compress	104	-0.22 **0.027**	-0.45 **1.348×10^{-6}**	-0.40 **2.083×10^{-5}**	-0.37 **1.121×10^{-4}**
Commons Math	420	0.20 **4.982×10^{-5}**	-0.36 **1.782×10^{-14}**	-0.19 **7.553×10^{-5}**	-0.03 0.572
Guice	94	0.01 0.935	-0.31 **0.002**	-0.29 **0.005**	-0.22 **0.031**
Jsoup	37	0.29 0.085	-0.37 **0.024**	0.16 0.337	0.20 0.229

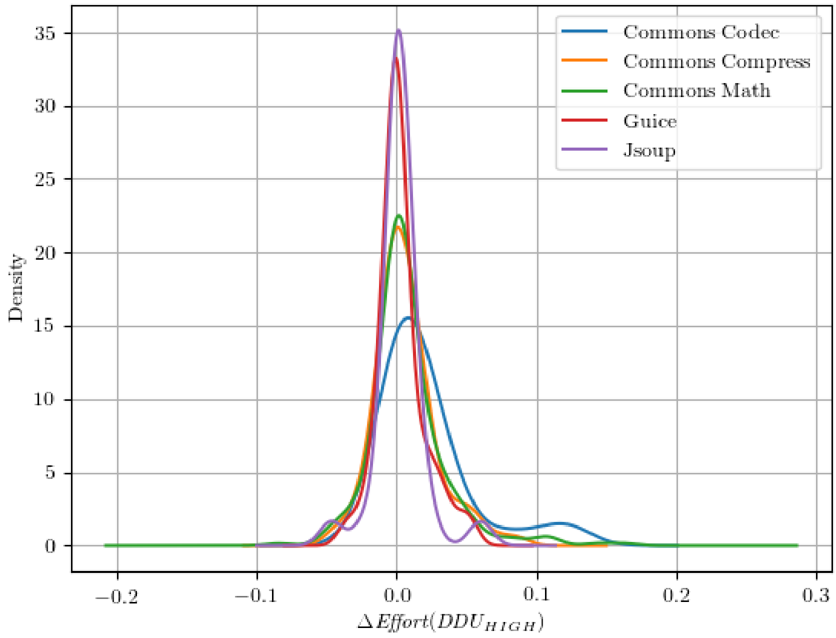


Figure 4.4

each class, we compare the effort of a test suite with a lower DDU value with a test suite with a higher DDU value. Identical approach is used for density, diversity, and uniqueness. Note that we exclude classes where the two test suites do not result in a metric difference. The results of this experiment are shown in Figure 4.4, and the median effort and statistical test results for each subject are shown in Table 4.2, Table 4.3, Table 4.4, and Table 4.5. We observe that $Effort(DDU_{HIGH})$ is statistically different in two cases out of five. Hence, there is no strong evidence that improving the DDU for a class' test suite will improve diagnosability.

Revisiting the second research question:

RQ2: What is the relation between density, diversity, uniqueness, and DDU and diagnosability?

A: In the first experiment, two out of the five projects show that DDU is negatively correlated with diagnosability. In the second experiment, two out of the five projects show that improving the DDU value of a class' test suite will improve the diagnosability. Therefore, based on these two experiments, there is no strong evidence that DDU is correlated to diagnosability. There is also no strong evidence that density, diversity, and uniqueness are correlated to diagnosability.

Table 4.2: Metrics and statistical tests related to density. Note that t is computed using the t-test and T is computed using the Wilcoxon signed-rank test.

Subject	Size	Effort / Shapiro-Wilk / Statistical test	
		$\mathit{Effort}(\mathit{density}_{LOW})$	$\mathit{Effort}(\mathit{density}_{HIGH})$
Commons Codec	29	0.1617	0.1722
		$W = 0.9633$, $\textit{p-value} = 0.3968$	$W = 0.9559$, $\textit{p-value} = 0.2609$
		$t = 0.4314$, $\textit{p-value} = 0.6678$	
Commons Compress	74	0.1539	0.1539
		$W = 0.9587$, $\textit{p-value} = 0.0165$	$W = 0.9582$, $\textit{p-value} = 0.0155$
		$T = 1340.0$, $\textit{p-value} = 0.9539$	
Commons Math	251	0.1765	0.1720
		$W = 0.9696$, $\textit{p-value} = 3.5022 \times 10^{-5}$	$W = 0.9682$, $\textit{p-value} = 2.2473 \times 10^{-5}$
		$T = 13967.0$, $\textit{p-value} = 0.2734$	
Guice	75	0.1408	0.1464
		$W = 0.9711$, $\textit{p-value} = 0.0843$	$W = 0.9749$, $\textit{p-value} = 0.1430$
		$t = -0.0742$, $\textit{p-value} = 0.9408$	
Jsoup	29	0.1078	0.1002
		$W = 0.9178$, $\textit{p-value} = 0.0268$	$W = 0.9208$, $\textit{p-value} = 0.0320$
		$T = 176.0$, $\textit{p-value} = 0.3695$	

Table 4.3: Metrics and statistical tests related to diversity. Note that t is computed using the t-test and T is computed using the Wilcoxon signed-rank test.

Subject	Size	Effort / Shapiro-Wilk / Statistical test	
		$\mathit{Effort}(\mathit{diversity}_{LOW})$	$\mathit{Effort}(\mathit{diversity}_{HIGH})$
Commons Codec	24	0.1791	0.1881
		$W = 0.9678$, $\textit{p-value} = 0.6154$	$W = 0.9685$, $\textit{p-value} = 0.6318$
		$t = -0.2070$, $\textit{p-value} = 0.8368$	
Commons Compress	73	0.1584	0.1584
		$W = 0.9546$, $\textit{p-value} = 0.0103$	$W = 0.9597$, $\textit{p-value} = 0.0199$
		$T = 1040.0$, $\textit{p-value} = 0.1241$	
Commons Math	256	0.1767	0.1724
		$W = 0.9693$, $\textit{p-value} = 2.6065 \times 10^{-5}$	$W = 0.9646$, $\textit{p-value} = 5.9871 \times 10^{-6}$
		$T = 13463.0$, $\textit{p-value} = 0.0649$	
Guice	75	0.1410	0.1464
		$W = 0.9712$, $\textit{p-value} = 0.0856$	$W = 0.9753$, $\textit{p-value} = 0.1512$
		$t = -0.0574$, $\textit{p-value} = 0.9542$	
Jsoup	29	0.1128	0.0998
		$W = 0.9211$, $\textit{p-value} = 0.0326$	$W = 0.9158$, $\textit{p-value} = 0.0239$
		$T = 192.0$, $\textit{p-value} = 0.5813$	

Table 4.4: Metrics and statistical tests related to uniqueness. Note that t is computed using the t-test and T is computed using the Wilcoxon signed-rank test.

Subject	Size	Effort / Shapiro-Wilk / Statistical test	
		$Effort(uniqueness_{LOW})$	$Effort(uniqueness_{HIGH})$
Commons Codec	26	0.1758	0.1622
		$W = 0.9676$, $p\text{-}value = 0.5624$	$W = 0.9654$, $p\text{-}value = 0.5104$
		$t = -0.9103$, $p\text{-}value = 0.3669$	
Commons Compress	55	0.1463	0.1463
		$W = 0.9378$, $p\text{-}value = 0.0069$	$W = 0.9445$, $p\text{-}value = 0.0132$
		$T = 481.0$, $p\text{-}value = \mathbf{0.0154}$	
Commons Math	187	0.1662	0.1564
		$W = 0.9621$, $p\text{-}value = 6.2737 \times 10^{-5}$	$W = 0.9636$, $p\text{-}value = 9.1669 \times 10^{-5}$
		$T = 5077.0$, $p\text{-}value = \mathbf{8.5858 \times 10^{-7}}$	
Guice	54	0.1348	0.1406
		$W = 0.9559$, $p\text{-}value = 0.0455$	$W = 0.9607$, $p\text{-}value = 0.0747$
		$T = 459.0$, $p\text{-}value = 0.0558$	
Jsoup	27	0.0979	0.0998
		$W = 0.8962$, $p\text{-}value = 0.0110$	$W = 0.9337$, $p\text{-}value = 0.0853$
		$T = 83.0$, $p\text{-}value = \mathbf{0.0108}$	

Table 4.5: Metrics and statistical tests related to DDU. Note that t is computed using the t-test and T is computed using the Wilcoxon signed-rank test.

Subject	Size	Effort / Shapiro-Wilk / Statistical test	
		$Effort(DDU_{LOW})$	$Effort(DDU_{HIGH})$
Commons Codec	29	0.1722	0.1454
		$W = 0.9593$, $p\text{-}value = 0.3174$	$W = 0.9655$, $p\text{-}value = 0.4472$
		$t = -0.8282$, $p\text{-}value = 0.4110$	
Commons Compress	74	0.1583	0.1583
		$W = 0.9568$, $p\text{-}value = 0.0129$	$W = 0.9617$, $p\text{-}value = 0.0247$
		$T = 974.0$, $p\text{-}value = \mathbf{0.0384}$	
Commons Math	262	0.1742	0.1717
		$W = 0.9699$, $p\text{-}value = 2.5558 \times 10^{-5}$	$W = 0.9619$, $p\text{-}value = 2.1199 \times 10^{-6}$
		$T = 11308.0$, $p\text{-}value = \mathbf{2.1258 \times 10^{-5}}$	
Guice	75	0.1403	0.1464
		$W = 0.9722$, $p\text{-}value = 0.0982$	$W = 0.9744$, $p\text{-}value = 0.1333$
		$t = -0.3054$, $p\text{-}value = 0.7604$	
Jsoup	29	0.1078	0.1002
		$W = 0.9237$, $p\text{-}value = 0.0379$	$W = 0.9153$, $p\text{-}value = 0.0233$
		$T = 155.0$, $p\text{-}value = 0.1765$	

Chapter 5

DDU vs. Error Detection

In this chapter, we discuss

Chapter 6

Conclusion

Bibliography

[1] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. Spectrum-based multiple fault localization. In *Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on*, pages 88–99. IEEE, 2009.

[2] Boris Beizer. *Software testing techniques*. Dreamtech Press, 2003.

[3] Tom Britton, Lisa Jeng, Graham Carver, Paul Cheak, and Tomer Katzenellenbogen. Reversible debugging software. *Judge Bus. School, Univ. Cambridge, Cambridge, UK, Tech. Rep*, 2013.

[4] Chris Parnin and Alessandro Orso. Are automated debugging techniques actually helping programmers? In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, pages 199–209. ACM, 2011.

[5] Alexandre Perez, Rui Abreu, and Arie van Deursen. A test-suite diagnosability metric for spectrum-based fault localization approaches. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 654–664, 2017.

[6] Thomas Reps, Thomas Ball, Manuvir Das, and James Larus. The use of program profiling for software maintenance with applications to the year 2000 problem. In *Software EngineeringESEC/FSE'97*, pages 432–449. Springer, 1997.

[7] John Robbins. *Debugging Applications for Microsoft .NET and Microsoft Windows*. Microsoft Press, 2003.

[8] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, Aug 2016.