2017

Multi-objective search-based approach to estimate issue resolution time

Wisam Al-Zubaidi
University of Wollongong, whaa807@uowmail.edu.au

Hoa K. Dam
University of Wollongong, hoa@uow.edu.au

Aditya K. Ghose
University of Wollongong, aditya@uow.edu.au

Xiaodong Li
Royal Melbourne Institute of Technology, xiaodong.li@rmit.edu.au

Multi-objective search-based approach to estimate issue resolution time

Abstract

Background: Resolving issues is central to modern agile software development where a software is developed and evolved incrementally through series of issue resolutions. An issue could represent a requirement for a new functionality, a report of a software bug or a description of a project task. Aims: Knowing how long an issue will be resolved is thus important to different stakeholders including end-users, bug reporters, bug triagers, developers and managers. This paper aims to propose a multi-objective search-based approach to estimate the time required for resolving an issue. Methods: Using genetic programming (a meta-heuristic optimization method), we iteratively generate candidate estimate models and search for the optimal model in estimating issue resolution time. The search is guided simultaneously by two objectives: maximizing the accuracy of the estimation model while minimizing its complexity. Results: Our evaluation on 8,260 issues from five large open source projects demonstrate that our approach significantly ($p < 0.001$) outperforms both the baselines and state-of-the-art techniques. Conclusions: Evolutionary search-based approaches offer an effective alternative to build estimation models for issue resolution time. Using multiple objectives, one for measuring the accuracy and the other for the complexity, helps produce accurate and simple estimation models.

Keywords

search-based, multi-objective, time, issue, resolution, estimate, approach

Disciplines

Engineering | Science and Technology Studies

Publication Details

Al-Zubaidi, W. H. A., Dam, H. K., Ghose, A. & Li, X. (2017). Multi-objective search-based approach to estimate issue resolution time. PROMISE: Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering (pp. 53-62). New York, United States: ACM.

Multi-objective search-based approach to estimate issue resolution time

Wisam Haitham Abbood Al-Zubaidi
University of Wollongong
Australia
whaa807@uowmail.edu.au

Hoa Khanh Dam
University of Wollongong
Australia
hoa@uow.edu.au

Aditya Ghose
University of Wollongong
Australia
aditya@uow.edu.au

Xiaodong Li
RMIT University
Australia
xiaodong.li@rmit.edu.au

ABSTRACT

Background: Resolving issues is central to modern agile software development where a software is developed and evolved incrementally through series of issue resolutions. An issue could represent a requirement for a new functionality, a report of a software bug or a description of a project task.

Aims: Knowing how long an issue will be resolved is thus important to different stakeholders including end-users, bug reporters, bug triagers, developers and managers. This paper aims to propose a multi-objective search-based approach to estimate the time required for resolving an issue.

Methods: Using genetic programming (a meta-heuristic optimization method), we iteratively generate candidate estimate models and search for the optimal model in estimating issue resolution time. The search is guided simultaneously by two objectives: maximizing the accuracy of the estimation model while minimizing its complexity.

Results: Our evaluation on 8,260 issues from five large open source projects demonstrate that our approach significantly ($p < 0.001$) outperforms both the baselines and state-of-the-art techniques.

Conclusions: Evolutionary search-based approaches offer an effective alternative to build estimation models for issue resolution time. Using multiple objectives, one for measuring the accuracy and the other for the complexity, helps produce accurate and simple estimation models.

1. INTRODUCTION

In software projects, an issue represents description of a bug or a security vulnerability (e.g. bug report issues), or a description of a new functionality (e.g. feature request or user story issues) or enhancements of an existing functionality, or a project task. Most of today's software projects are issue-driven where a project consists of a number of past issues (i.e. issues that have been closed), ongoing issues (i.e. issues that the team are working on), and new issues (i.e. issues that have just been created). Knowing when an issue will be resolved is important for many stakeholders. For example, the end-users may want to know when the new functionality they requested will be implemented. The bug reporters may be interested in learning when a particular bug will be fixed. The project managers may need to estimate the time it will take for completing a project task since these estimates are critical to their plan for costing and timing future releases.

Predicting when a particular issue will be resolved is however difficult. Existing practices in the industry often use the average resolution time of past issues, combined with a certain margin of error, as an estimate for the resolution time of the new issues. However, software issues may be significantly different from one another in their nature and the complexity of resolving them. Hence, the quality of the average resolution time as an estimator is often poor. Other existing practices heavily rely on experts' (e.g. project managers or experienced developers) subjective assessment to arrive at an estimate for the time and effort of resolving an issue. Relying on expert knowledge is however sometimes based on outdated experience and an underlying bias, thus may lead to inaccuracy in estimation. A number of software analytics techniques have recently been proposed to address this problem. These work (e.g. [16, 32]) mine the historical data generated when issues were reported and resolved. They identify features which characterize an issue and also influence on its resolution time. They then build machine learning models, train them using historical issues with known resolution time, and use them for future estimations.

Machine learners have also widely used for estimating the effort required for developing a complete software system (e.g. [17]). Recent approaches (e.g. [10, 28]) have employed a evolutionary, search-based approach to this problem. Largely inspired by *Sarro et. al.*'s work [28] done for effort estimation for the whole project, we employ a multi-objective search-based evolutionary approach to estimate the resolution time of each single issue in the project. Specifically, we leverage a meta-heuristic technique, namely

PROMISE, November 8, 2017, Toronto, Canada
© 2017 ACM. ISBN 978-1-4503-5305-2/17/11…$15.00
DOI: https://doi.org/10.1145/3127005.3127011

genetic programming (GP) [18], to generate a large number of candidate estimation models, and search for the ones that are optimal with respect to a number of objectives. Differing from Sarro *et. al.* [28], we do *not* impose a fixed structure and depth of candidate estimation models.

We explore two objectives guiding our search algorithms. Similarly to *Sarro et. al.* [28], the *first objective* is to minimize the Sum of Absolute Errors, which measures the accuracy of an estimation model in terms of the differences between values (i.e. issue resolution time) estimated by the model and the values actually observed. The pressure of minimizing the estimation errors may however cause the solution model to adhere precisely to noisy data in the training set, which potentially make the model be excessively large and complex (hence, overfitting problems). While accuracy is critical for an estimation model, the Occam's Razor principle of *parsimony* also plays an important role here: the model needs to be expressed in a simple way, easy for software practitioners to interpret [23]. Hence, our *second objective* is to minimize the *complexity* of an estimation model, which can be measured in terms of the size of an expression tree representing the model. This is also another key difference from *Sarro et. al.*'s approach [28]. This second objective also leads to reduced computational costs since it encourages parsimonious (thus, computationally efficient) candidate solutions be generated. We name our approach *Multi-Objective Issue Resolution Time Estimator* (MOIRTE).

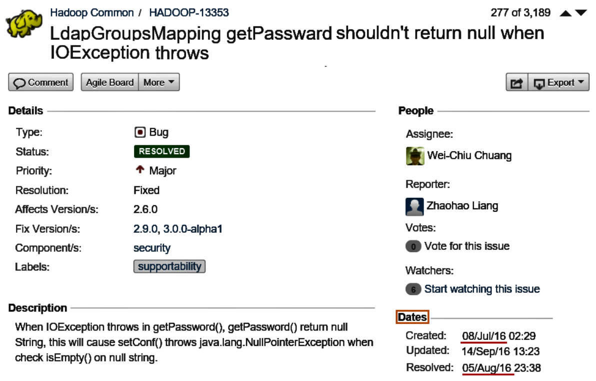
Our search-based approach outperforms the three common baselines (random guessing, and mean and median) and state-of-the-art techniques (linear regression, case-based reasoning, and random forests) in predicting issue resolution time. The evaluation was performed against a dataset of 8,260 issues which we collected from five different projects including four Apache Hadoop projects (Common, HDFS, MapReduce, Yarn) and one Apache Mesos project. Similarly to *Sarro et. al.*'s work [28], we use two standardized measures, Mean Absolute Error and Standardized Accuracy, to evaluate the performance of estimation models, and also use a non-parametric Wilcoxon test [4] and Vargha and Delaney's statistic [31] to demonstrate both the statistical significance and the effect size of the results.

The remainder of this paper is organized as follows. Section 2 formulates the problem of estimating issue resolution time. Section 3 describes our multi-objective approach to solve this problem using evolutionary algorithms. Section 4 reports on the experimental evaluation of our approach. Related work is discussed in Section 5 before we conclude and outline future work in Section 6.

2. ISSUE RESOLUTION TIME

In modern software development settings, software is developed through repeated cycles (iterative) and in smaller parts at a time (incremental), allowing for adaptation to changing requirements at any point during a project's life. A project has a number of iterations, in each of which, the development team resolves a number of *issues*. An issue could be requesting the implementation of a new functionality, fixing a bug or a security vulnerability, or refactoring the code.

Figure 1 shows the report of issue HADOOP-13353 in the Hadoop Common project. This issue report was recorded in the widely-used JIRA project management system. As can be seen from the report, this issue was created on 08 July



Hadoop Common / HADOOP-13353 277 of 3,189

LdapGroupsMapping getPassward shouldn't return null when IOException throws

Details

Type:	Bug
Status:	RESOLVED
Priority:	Major
Resolution:	Fixed
Affects Version/s:	2.6.0
Fix Version/s:	2.9.0, 3.0.0-alpha1
Component/s:	security
Labels:	supportability

People

Assignee: Wei-Chiu Chuang

Reporter: Zhaohao Liang

Votes: 0 Vote for this issue

Watchers: 6 Start watching this issue

Description

When IOException throws in getPassword(), getPassword() return null String, this will cause setConf() throws java.lang.NullPointerException when check isEmpty() on null string.

Dates

Created: 08/Jul/16 02:29
Updated: 14/Sep/16 13:23
Resolved: 05/Aug/16 23:38

Figure 1: An example of a Hadoop Common issue recorded in JIRA. Note that we highlighted the dates when the issue were created and resolved.

2016 and resolved on 05 August 2016. This issue was a bug and its resolution was set to "Fixed", indicating that it was in fact a valid bug and has been fixed.

We would like to estimate how long it will take to resolve an issue using the following information provided with an issue report. These are common information which must be provided at the time an issue is created:

1. *Type*: each issue is assigned with a type (e.g. Bug, Task, Improvement, New feature, etc.) which indicates the nature of the task associated with resolving with the issue. For example, a "bug" issue reports a defect while a "new feature" describes a request for implementing a new functionality.

2. *Priority*: The issue's priority presents the order in which an issue should be attended with respect to other issues. In the projects we studied, there are 5 common types of priority: Blocker, Critical, Major, Minor, and Trivial. Issues with blocker priority should be more concerned than issues with major or minor priority since the former block other issues to be completed.

3. *Reporter*: We use reporter's reputation, a common feature which has been studied in previous work in mining bug reports. The intuition here is that poor issue reports may take longer time to resolve and reporters who frequently write them will accumulate such a reputation. We use the widely-used Hooimeijer's reporter reputation [14] as follows:

$$reputation(D) = \frac{|opened(D) \cap fixed(D)|}{|opened(D)| + 1}$$

The reputation of a reporter D is measured as the ratio of the number of issues that D has opened and fixed to the number of issues that D has opened plus one.

4. *Title and description*: The title and description of an issue explains its nature and thus can be a good feature. We employ a common approach to translate an issue's title and description into the number of word

counts and use this as a feature. In addition, we also use the readability of the issue description as another feature. Readability is a quality indicator for issue reports [14]. We hypothesize that issues that are more difficult to understand will be more difficult to deal with and thus potentially take longer time to resolve. We use the Gunning fog readability metric [22] to measure an issue description's readability score. The lower Gunning fog score is, the easier to understand an issue description.

Note that other information associated with an issue report (e.g. assignee, fix versions, votes, watchers, etc.) can be used. However, some of these information are not mandatory (e.g. fix versions), or do not have any value at the time when an issue is created (e.g. assignee or the number of votes or watchers), which is the time we would like to make a prediction. Some of them are specific to an issue tracking system, while the features that we use here are commonly found in many issue-tracking systems.

3. APPROACH

3.1 Overview

Our approach falls under the search-based software engineering umbrella and is largely inspired by *Sarro et. al.*'s search-based approach [28] to estimate the effort required for completing a whole project. We however focus on the issue-level, i.e. estimating the resolution time for each issue in a project. Figure 2 gives an overview of our approach. We build a training set by collecting completed issues from a given project, and extracting their actual resolution time. The resolution time is the elapsed time when an issue was created and when it was resolved. We design a set of features characterizing an issue (see Section 2) and extract the values of these features at the time when the issue was created. We then iteratively generate candidate estimation models (by using a set of mathematical operators to combine the issue features) and search for the "best" estimation model with respect to the training set. This search process employs evolutionary algorithms which work based on the principle that a population of candidate solutions (also referred to as individuals) to an optimization problem is evolved toward better solutions, following Darwin's evolution theory. Each candidate solution has a number of properties (i.e. chromosomes or genotype) which can be mutated and altered to derive new candidate solutions. In our context of estimating issue resolution time, a candidate solution is an estimation model.

The estimation model found at the end of the search process is used for estimating the resolution time of new issues in the same project (within-project estimation) or in a different project (cross-project estimation). We will now describe the details of our approach.

3.2 Symbolic regression

Estimating issue resolution time can be considered as a regression problem: we need to model the relationship between the time required for resolving an issue and a set of features characterizing the issue. Here, we measure issue resolution time as a *continuous* number of days, and rely on five basic information associated with an issue: type, priority, reporter, title and description (see Section 2) to extract

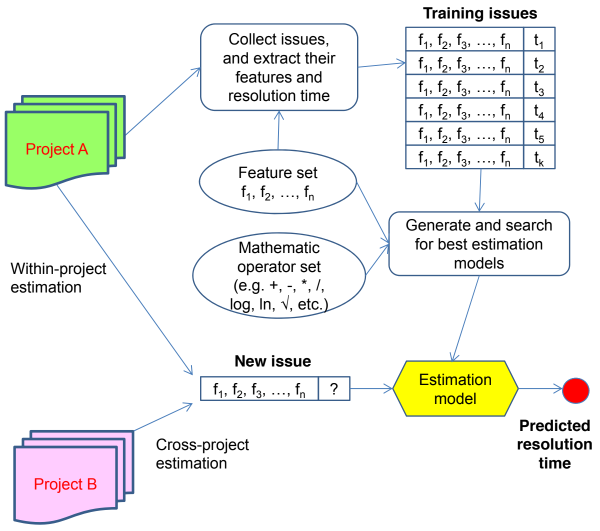

Project A
Collect issues, and extract their features and resolution time
Training issues
f1, f2, f3, …, fn | t1
f1, f2, f3, …, fn | t2
f1, f2, f3, …, fn | t3
f1, f2, f3, …, fn | t4
f1, f2, f3, …, fn | t5
f1, f2, f3, …, fn | tk
Feature set f1, f2, …, fn
Mathematic operator set (e.g. +, -, *, /, log, ln, √, etc.)
Generate and search for best estimation models
Within-project estimation
New issue
f1, f2, f3, …, fn | ?
Estimation model
Predicted resolution time
Cross-project estimation
Project B

Figure 2: An overview of our approach

the features. Since the issue type and priority are categorical features, we need to perform additional steps to ensure the results from a regression model be interpretable. Specifically, we use one hot encoding to transform the issue type in a number of features (corresponding to the number of issue types), each of which represents one type and has a value of either 0 or 1. Since there are 7 issue types (i.e. bug, task, improvement, test, sub-task, new feature, and wish), this results in 7 features. For issue priority, we convert it into an ordinal value from 1 to 5 where 1 represents the least priority and 5 represents the most priority. In total, we have 13 numerical features (7 derived from issue type, 1 from priority, 1 from reporter reputation, 2 from the title and 2 for the description) characterizing an issue.

An estimation model can be viewed as a mathematical expression which combines those features of an issue and a set of mathematical operators to output a scalar value representing the time required for resolving an issue. We use a training dataset of past issues (with known resolution time) to *search* the space of those mathematical expressions to find the estimation model that best fits the training dataset. This model is then used to predict the time required for resolving new issues. This approach is commonly referred to as *symbolic regression* and has been previously used by Sarro *et. al.*'s [28] in estimating effort for the whole project. We adapted this approach from Sarro *et. al.*, but also made several key differences: (i) not imposing a fixed structure nor depth on candidate models; (ii) using a different second objective function to explicitly control the model's complexity; and (iii) using a wider range of mathematical operators.

Since each candidate estimation model is a mathematical expression, we represent it as an expression tree to facilitate the application of genetic operators (described in details later) to derive new candidates. Issue features are encoded as leafs of the tree and mathematical operators as its internal nodes. We thus use genetic programming [18], a meta-heuristic algorithm in the family of evolutionary algorithms, which specifically deals with tree representation. We employ a wide range of thirteen mathematical operators ($+$, $-$, $*$, $/$,

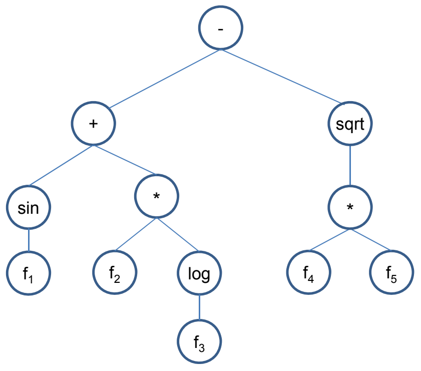


Figure 3: An example of expression tree representing a candidate estimation model. Note that f_i represents a feature of an issue.

exp, log, log_{10}, sin, cos, tan, $power$, $square$, $squareroot$), as opposed to only three operators used in Sarro *et. al.*'s [28]. Figure 3 shows an example of an expression tree representing a estimation model in which the resolution time of an issue is calculated as $(sin(f_1) + (f_2 * log(f_3))) - \sqrt{f4 * f5}$ where f_1, f_2, f_3, f_4 and f_5 are some of the thirteen issue features.

3.3 Fitness functions

The search for the best estimation model is guided by a number of fitness functions, which are used to compare if a candidate model is "better" than another one. We employ two fitness functions: one reflecting the accuracy of an estimation model in predicting issue resolution time and the other representing the model's complexity. The details of these fitness functions are described as below.

3.3.1 Sum of Absolute Errors

We use a training set of past issues (with known resolution time) to evaluate the accuracy of a candidate estimation model. A number of measures have been used to evaluate the predictive performance of an estimation model and can be classified in two groups: relative measures such as Mean of Magnitude of Relative Error or Root Mean Square Error (RMSE), or absolute measures like the Sum of Absolute Error (SAE). Previous work (e.g. [20, 28]) have suggested that the predictive performance of estimation models found by genetic algorithms is affected by the use of those different measures as a fitness function. Specifically, using relative measures as a fitness function has a negative impact on the model accuracy, while absolute measures appear to not have damaging effect [28]. Hence, similarly to [28], we chose to use the Sum of Absolute Error (defined below) as our first fitness function.

$$SAE = \sum_{i=1}^{N} |ActualTime_i - EstimatedTime_i| \qquad (1)$$

where N is the number of issues in the training set, $ActualTime_i$ is the actual resolution time for issue i in the training set, and $EstimatedTime_i$ is the time estimated by a candidate model.

3.3.2 Tree size

Using an accuracy measure as the sole fitness function may result in excessively complex estimation models. The pressure of minimizing the estimation errors may lead to solution models that "overfit" the training data, which thus negatively affects the generalization performance of the models. It also takes more computational resources to store and evaluate complex models during the evolution process. In addition, software practitioners usually find complex estimation models difficult to understand and interpret [23].

The simplest method to control the complexity of estimation models is imposing a fixed limit on the depth or size (i.e. the number of nodes) of expression trees representing those models. Sarro *et. al.* [28] employed this approach by ensuring all the trees in the population having a fixed depth. This approach however suffers from a number of limitations. Determining a good value for the limit is very challenging. A small limit may prevent good solutions from being generated, while a large limit may still result in over-complex solutions. In addition, the process of eliminating non-conformance individuals from the population may create bias and adverse affect [11].

To control the balance between accuracy and complexity, we employ a second fitness function which measures the complexity of a solution estimation model. Since we represent an estimation model as an expression tree, the size of the tree can be used as a complexity indicator. Hence, the second fitness function returns the *size of a solution tree*, i.e. the number of nodes in the tree. For example, there are 12 nodes in the tree in Figure 3, hence its size is 12. The tree size reflects to some extends both the depth and width of a tree.

3.4 Evolutionary search

The search for an estimation model starts with an initial population in which each individual in the population is a candidate estimation model. The initial population is created by randomly generating a number of expression trees (each represents an individual) using the thirteen mathematical operators and thirteen issue features. The fitness values of each individual with respect to each of two fitness functions (see Section 3.3) are computed. The population is then undergone a selection process.

Selected individuals form parents to generate a new generation of individuals through the crossover and mutation operators. These genetic operators act directly on the expression trees to form new valid trees. The mutation operator chooses a node in the expression tree and substitutes the sub-tree at that node by a randomly generated sub-tree. Figure 4 shows an example of how the tree in Figure 3 is mutated. The crossover operators involve two parent trees and generate two offspring trees by exchanging selected branches (see Figure 5). Generated trees which give negative or invalid (e.g. division by zero) estimated time are assigned with a very large (infinite) sum of absolute errors. This would prevent those trees from being selected in the evolution process. This evolution process continues until a fixed number of generations has been reached.

We seek for estimation models that meet both objectives: high accuracy and low complexity. These solutions would belong to a Pareto front of estimation error and tree size (see Figure 6). To find such a Pareto front, we employ a widely-used multi-objective optimization algorithm, the non-dominated sorting genetic algorithm (NSGA-II) [6]. The NSGA-II algorithm works based on the principle of non-

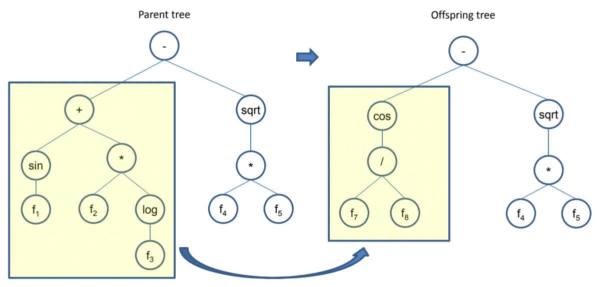


Figure 4: An example of the mutation operator

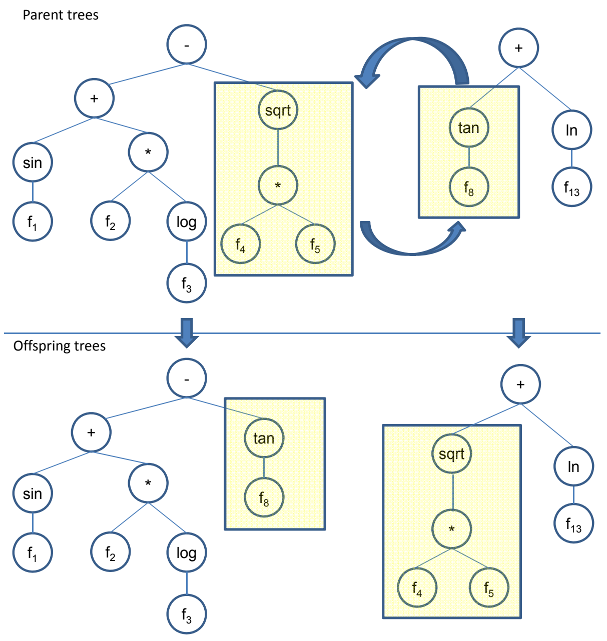


Figure 5: An example of the crossover operator

dominated sorting (Pareto dominance). In multi-objective optimization, an individual is said to *dominate* another individual if the former is better than the latter with respect to at least one objective, and not worse in the remaining objectives. For example, in Figure 6 estimation model E_1 does *not* dominate E_2 since the former is smaller than the latter in tree size but has greater sum of absolute errors. On the other hand, E_1 dominates E_4 since the E_1 is smaller than E_4 in tree size and also has smaller sum of absolute errors.

At each generation, NSGA-II sorts the current population into a number of non-dominated fronts (e.g. fronts 1, 2 and 3 in Figure 6). Each non-dominated front contains individuals which do not dominate each other. Individuals in the first non-dominated front dominate those in the second front, which in turn dominate individuals in the third front, and so on. Individuals in the same non-dominated front are assigned the same rank, which is the index of its front. For example, in Figure 6 estimation models E_1 and E_2 have the same rank 1, while E_4 has rank 2. The crowding distance of each individual is then computed as the sum of the distance between itself and its nearest neighbours on the same front. For example, the crowding distance for estimation model E_1 is the sum of the distances between E_1 and E_2 and between E_1 and E_3 (see Figure 6). NSGA-II [6] uses the rank and the crowding distance to define the crowded-comparison operator $\prec_c$, which is central to the operation of the algorithm.

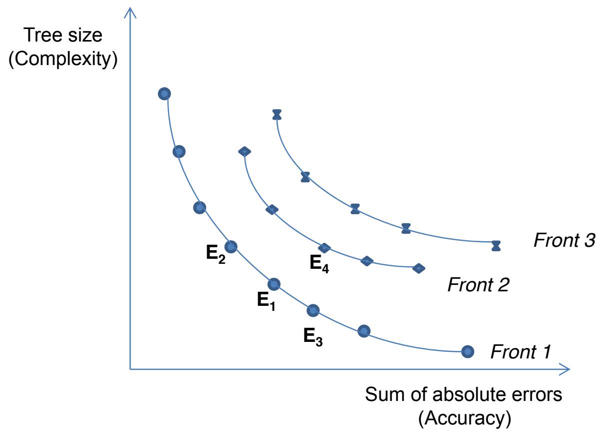


Figure 6: An example of non-dominated fronts

The partial order $\prec_c$ is defined as follows: for two individuals E_i and E_j, $E_i \prec_c E_j$ if and only if: (a) the rank of E_i is less than that of E_j; or (b) their ranks are the same (i.e. they are on the same non-dominated front) and the crowd distance of E_i is greater than that of E_j. The intuition here is that individuals with lesser domination rank are favoured in case when they are on different fronts, and individuals in a less dense region (i.e. higher crowding distance) are preferred in case when they are on the same front. This necessitates the pressure for the population to move towards the Pareto Front and spread along it. NSGA-II uses the crowded-comparison operator $\prec_c$ to guide the selection process for forming the offspring population. The next generation is selected from a combination of the parent and offspring operation. This is to minimize the possibility of losing a high quality solution. In the final generation, NSGA-II returns a set of non-dominated solutions. Choosing which one of these solutions to use is usually a user-specific decision. In our case, we choose to use the solution which has the lowest sum of absolute errors with respect to the training set. For more details on NSGA-II, we refer to the reader to [6].

Following the common practice [15], we used the following parameters. The size of the initial population is set to $10 * V$ where V is the number of features ($V = 13$ in our case, hence 130). The number of generations was set to $1,000 * V$, i.e. 13,000. Crossover probability was set to 0.9, mutation probability was 0.1, and reproduction probability was 0.2. We used tournament selection method. These are common values used in previous studies [15, 28]. We used an implementation of NSGA-II in the MOEA Framework[1].

4. EVALUATION

In this section, we report an empirical evaluation of our approach. We first describe the datasets we have built for

[1]http://moeaframework.org/index.html

our evaluation. We then discuss the experimental settings and performance measures. We then present the evaluation results which answer a range of research questions.

4.1 Datasets

We chose five projects, namely Common, HDFS, MapReduce, Yarn and Mesos from the well-known Apache to build a dataset of issues with known creation and resolved times. Those issues were recorded in the widely-used JIRA tracking system. We used the Representational State Transfer (REST) API provided by JIRA for querying the issues. After that we collected issue reports in JavaScript Object Notation (JSON) format. In terms of Common, HDFS, MapReduce and Yarn, the collected issues have the created date and resolved date up to September 9, 2016 and for Mesos is up to March 24, 2017. From these projects, we extracted a total of 16,858 issues. We excluded issues with a status other than "Resolved" to avoid collecting uncompleted issues and a resolution other than "Fixed" (e.g. "Duplicate" or "Not Fix", or "Invalid") in order to collect only "real" issues. In the end, we included 8,260 issues into our dataset. We have calculated the duration time for each issue by subtracting its resolved time from its created time. The resolution time is measured in days.

Table 1: Descriptive Statistics of our Dataset

Projects	# selected issues	Mean	Median	Mode	Std
Common	1,402	37.19	10	2	62.24
HDFS	2,334	28.66	7	2	55.49
MapReduce	635	45.68	14	2	72.62
Yarn	930	46.82	17	2	69.32
Mesos	2959	52.97	18	1	77.13
Total	8,260				

Table 1 summarizes the descriptive statistics of our dataset in terms of mean, median, mode and standard deviation of resolution time. The median resolution times range from 7 to 18 days in the five projects, while the mean resolution times were from 28 to 52 days. Although the resolution times varied quite substantially (standard deviation in all projects were above 55), most of the issues were resolved within 2 days. We will make the data sets used in this study be publicly available for the research community once the paper has been accepted.

4.2 Experimental settings and measures

For each project, we applied a cross validation process which is a widely employed technique to validate an estimation or prediction model. Specifically, we divided our dataset into 10 folds and applied cross-validation (i.e. used nine folds for training and the remaining one fold for testing) to reduce the estimation instability and bias. We ran each algorithm 30 times and took the median result. Similarly to previous work in effort estimation (e.g. [8, 19, 28, 29]), we employed two widely-used standardised measures, Mean Absolute Error (MAE) and the Standardized Accuracy (SA). They are defined as below.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|ActualTime_i - EstimatedTime_i| \quad (2)$$

where N is the total number of issues used in the test set. Estimation models with lower MAE are better in terms of accuracy.

Standardized Accuracy (SA) measures how good an estimation model is with respect to random guessing:

$$SA = (1 - \frac{MAE}{MAE_{rguess}}) \times 100 \quad (3)$$

where MAE_{rguess} is the MAE averaging a large number of random guesses. Estimation models with larger SA are more useful.

To compare the performance between different estimation models, we followed [28] and tested the statistical significance of the absolute errors produced from those estimation models using the Wilcoxon Signed Rank Test [4]. Wilcoxon test does not require the data be normally distributed, which is suitable to our data. We set the confidence limit at 0.05 (i.e. $p < 0.05$). We also applied the Vargha and Delaney's $\hat{A}_{12}$ non-parametric effect size measure [31]. This non-parametric effect size measure is suitable for testing randomized algorithms in software engineering, especially in the context of effort estimation [1, 28]. The $\hat{A}_{12}$ measures the probability that, estimation model E_i achieves better accuracy (i.e. smaller absolute errors) than estimation model E_j using the following formula [31]:

$$\hat{A}_{12} = (r_1/m - (m+1)/2)/n \quad (4)$$

where r_1 is the rank sum of (test) issues where E_i produces smaller absolute error than E_j does, and m and n are the number of issues tested for E_i and E_j respectively. Note that $\hat{A}_{12} = 0.5$ when the two estimation models perform in an equivalent way, while $\hat{A}_{12} > 0.5$ when E_i perform better than E_j.

4.3 Results

In this section, we present the results of our experimental evaluation[2] in terms of answering the following research questions.

RQ1. *Is the multi-objective search-based approach suitable for estimating issue resolution time?*

To answer this question, we compared our multi-objective search-based approach against three common baselines: *Random Guessing*, and *Mean* and *Median*, which are often used in effort estimation [28]. Random guessing is a naive technique for estimation [28, 29]. It performs random sampling over a set of issues with a known resolving time, choosing randomly one issue from the sample, and uses the resolving time of that issue as the estimate of the target issue. Random guessing does not use any information associated with the target issue. Thus, any useful estimation model should outperform random guessing. Mean and Median estimations are also commonly used as baseline benchmarks for effort estimation. They use the mean or median resolving time of the past issues to estimate the resolving time of the target issue.

As can be seen from Table 2, our approach MOIRTE produced better estimations in terms of MAE and SA than

[2]All the experiments were run on a Microsoft Windows 10 Home PC with an Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz and 8.00 GB RAM.

the Random, Mean and Median did. MOIRTE consistently outperformed all the three baselines in all five projects. Averaging across all projects, MOIRTE achieved an accuracy of 24.17 MAE and 58.45 SA, while the best of the baselines (Median) achieved 38.65 MAE and 33.49 SA.

Table 2: Mean Absolute Error (MAE) and Standard Accuracy (SA) values achieved by our approach MOIRTE, the baselines (Mean and Median Time), and state-of-the-art techniques: Linear Regression (LR), Case-Based Reasoning (CBR) Weiss *et. al.*, and Random Forests (RF). MAE and SA results are also included for the the single objective search with unlimited depth (GP-SAE) – discussed later in RQ3.

Project	Measures	MOIRTE	GP-SAE	Mean	Median	LR	CBR	RF
Common	MAE	**22.35**	30.13	41.52	34.25	40.68	44.63	33.89
	SA	**57.14**	42.23	20.38	33.80	21.99	14.42	35.02
HDFS	MAE	**17.80**	23.36	33.88	27.19	31.25	33.46	26.10
	SA	**58.02**	44.91	20.09	35.88	26.29	21.07	38.44
Mapreduce	MAE	**23.37**	37.52	49.99	42.40	50.46	58.06	39.38
	SA	**63.26**	41.02	21.41	33.34	20.67	8.72	38.10
Yarn	MAE	**23.97**	36.05	49.89	41.53	44.84	49.39	39.18
	SA	**60.01**	39.85	16.76	30.72	25.18	17.59	34.62
MESOS	MAE	**33.35**	42.71	57.08	47.88	54.82	59.35	44.35
	SA	**53.82**	40.86	20.95	33.69	24.08	17.81	38.58

LR:Linear Regression, CBR:Case-Based Reasoning, RF:Random Forest

Table 3: Comparison between our approach MOIRTE and the three baseline techniques using Wilcoxon test and $\hat{A}_{12}$ effect size (in brackets). This comparison is also done for the single-objective genetic programming algorithm with unlimited depth (GP-SAE), which is discussed in RQ3.

Project	Technique	Mean	Median	Random Guessing
Common	MOIRTE	<0.001 [0.76]	<0.001 [0.56]	<0.001 [0.75]
	GP-SAE	<0.001 [0.72]	0.0167 [0.52]	<0.001 [0.72]
HDFS	MOIRTE	<0.001 [0.79]	<0.001 [0.55]	<0.001 [0.78]
	GP-SAE	<0.001 [0.76]	<0.001 [0.52]	<0.001 [0.75]
MapReduce	MOIRTE	<0.001 [0.80]	<0.001 [0.60]	<0.001 [0.72]
	GP-SAE	<0.001 [0.73]	0.4423 [0.52]	<0.001 [0.68]
Yarn	MOIRTE	<0.001 [0.78]	<0.001 [0.59]	<0.001 [0.79]
	GP-SAE	<0.001 [0.75]	0.0033 [0.54]	<0.001 [0.76]
Mesos	MOIRTE	<0.001 [0.76]	<0.001 [0.59]	<0.001 [0.78]
	GP-SAE	<0.001 [0.72]	<0.001 [0.54]	<0.001 [0.73]

Table 3 shows the results of the Wilcoxon test and the corresponding $\hat{A}_{12}$ effect size to measure the statistical significance and effect size (in brackets) of the improved accuracy achieved by MOIRTE over the three baselines. In all cases, our approach MOIRTE significantly outperforms the baselines with $p < 0.001$ and effect sizes greater than 0.5. We also compared the performance of the single-objective genetic programming algorithm using the sum of absolute error as the single objective function (GP-SAE), which will be discussed later in *RQ*3. The results clearly suggest that our approach outperforms the naive benchmarks. The next step is thus assessing if it outperforms the existing techniques in predicting issue resolving time, which leads us to the second research question.

RQ2. *Does the search-based approach provide more accurate estimates than existing techniques used in predicting issue resolution time?*

Table 4: Comparison between our approach MOIRTE and the state-of-the-art techniques (LR, CBR Weiss *et. al.* and RF) using Wilcoxon test and $\hat{A}_{12}$ effect size (in brackets. This comparison is also done for the single-objective genetic programming algorithm with unlimited depth (GP-SAE), which is discussed in RQ3.

Project	Technique	LR	CBR	RF
Common	MOIRTE	<0.001 [0.75]	<0.001 [0.67]	<0.001 [0.72]
	GP-SAE	<0.001 [0.71]	<0.001 [0.62]	<0.001 [0.68]
HDFS	MOIRTE	<0.001 [0.74]	<0.001 [0.64]	<0.001 [0.70]
	GP-SAE	<0.001 [0.71]	<0.001 [0.61]	<0.001 [0.67]
Mapreduce	MORITE	<0.001 [0.79]	<0.001 [0.72]	<0.001 [0.73]
	GP-SAE	<0.001 [0.72]	<0.001 [0.66]	<0.001 [0.67]
Yarn	MORTIE	<0.001 [0.75]	<0.001 [0.70]	<0.001 [0.72]
	GP-SAE	<0.001 [0.70]	<0.001 [0.66]	<0.001 [0.68]
Mesos	MOIRTE	<0.001 [0.74]	<0.001 [0.66]	<0.001 [0.68]
	GP-SAE	<0.001 [0.69]	<0.001 [0.62]	<0.001 [0.62]

To answer this question, we compare our search-based approach against the three existing techniques that have been widely used in effort estimation: Linear Regression (LR), Case-Based Reasoning (CBR) and Random Forests. For case-based reasoning, we used k-nearest-neighbour(kNN) as done in the seminal work of Weiss *et. al.* [32]). Random Forests (RF) is chosen since it is currently the most effective method for effort estimation [17]. RF is an ensemble method which combines the estimates from multiple estimators. RF achieves a significant improvement over the decision tree approach by generating many classification and regression trees. Each tree is built on a random resampling of the data, with a random subset of variables at each node split. Then through averaging, tree predictions can be aggregated. Note that all these three prediction models use the same set of features as in our approach.

We used the implementation of linear regression and kNN provided with Weka [13]. Since it is tedious to find the optimal hyperparameters for these classification or regression algorithms, we automated this process using Weka's MultiSearch, a meta-classifier for tuning hyperparameters of a given base classifier or regressor. Specifically, for linear regression, we focused on tuning two hyperparameters: ridge (ranging from 1e-7 to 10) and selection method (with three methods: none, greedy and M5). For kNN, we used the brute force search algorithm (i.e. LinearNNSearch), Euclidean distance for the distance function, and tuned k number ranging from 1 to 64. We experimented with two implementations of Random Forests: one provided in Weka and the other written in Matlab[3], and found that the Matlab implementation produced better results with our data. Thus we chose this implementation to compare against our approach. We tuned Random Forests from 1 to 500 trees. All tuning was done using training data.

The MAE and SE results (see Table 2) shows that Random Forests is the best performer amongst the three existing techniques. However, when comparing against our approach, Random Forests consistently produced higher MAE and lower SA than MOIRTE in all five projects. The improvement brought by our approach over Random Forests was from 24% to 40% in MAE, with an average improvement of 34% in the five projects we studied. The Wilcoxon

[3]https://github.com/ami-GS/randomforest-matlab

test (see Table 4) also confirms this: the improvement of MOIRTE over LR, CBR, and RF is significant ($p < 0.001$) with the effect size greater than 0.7 in most cases. These results suggest that our multi-objective search-based approach offers an alternative and effective technique to issue resolution time estimation. The improvement may be due to its capability of capturing the nonlinear relationship between issue features and resolution time. Also, our approach does not carry any human biases nor affected by unknown domain-specific knowledge by not imposing any prior model structure and size.

RQ3. *Does the multi-objective approach produce more accurate estimates than the single-objective approach in predicting issue resolution time, and at the same time produce solutions of lower complexity?*

This question aims to investigate whether using the tree size as the second objective offers significant benefits in terms of both accuracy and complexity. To do so, we also implemented the traditional single-objective genetic programming algorithm using the sum of absolute error as the objective function. We name this alternative approach as GP-SAE. We experimented with two variants of this algorithm, one with unlimited depth tree and the other with a limited depth tree of 10. The former represents a method with no control on the complexity of the solutions, while the latter represents a technique with a constant limit on the tree size.

Table 5: Comparison between our multi and single objective (MOIRTE vs. GP-SAE) using Wilcoxon test and $\hat{A}_{12}$ effect size (in brackets). The second row reports the average tree size (i.e. the number of nodes) of a solution estimation model – the first number produced by MOIRTE while the second number produced by GP-SAE.

	GP-SAE (unlimited depth)				
	Common	**HDFS**	**Mapreduce**	**Yarn**	**Mesos**
MOIRTE	<0.001 [0.55]	<0.001 [0.54]	<0.001 [0.61]	<0.001 [0.57]	<0.001 [0.58]
Tree size	13 vs. 415	14 vs. 498	18 vs. 328	17 vs. 306	24 vs. 386
	GP-SAE (depth = 10)				
	Common	**HDFS**	**Mapreduce**	**Yarn**	**Mesos**
MOIRTE	<0.001 [0.52]	<0.001 [0.53]	<0.001 [0.59]	<0.001 [0.56]	<0.001 [0.56]
Tree size	13 vs. 38	14 vs. 46	18 vs. 42	17 vs. 30	24 vs. 81

Results from Tables 2, 3 and 4 show that the single objective approach with unlimited depth tree (GP-SAE) even outperforms all the baselines and state-of-the-art techniques. However, using tree size as the second objective has brought significant improvement in estimation accuracy. Across the five projects we studied, MOIRTE achieved from 22% – 37% improvement over GP-SAE in MAE (see Tables 2). The Wilcoxon test also confirmed that the improvement brought by using our multi-objective approach is significant ($p < 0.001$) in all cases with effect size from 0.54 to 0.61 (see the first row in Table 5).

Our approach of using tree size as the second objective is effective not only in improving the accuracy of the estimation model but also in the reducing its complexity. As can be seen in Table 5, the average tree sizes of the solution estimation models produced by MOIRTE were significantly less than those produced GP-SAE (e.g. 13 nodes versus 415 nodes for the Hadoop Common project). The approach of setting a fixed depth limit (depth = 10) is also not as effective as the multi-objective approach. Although using this approach reduced the tree size of the solution estimation models, it is still inferior to the multi-objective approach in terms of accuracy (see the bottom part of Table 5). These results clearly demonstrate the benefit of using our multi-objective approach in terms of producing both accurate and simple estimation models.

RQ4. *Is the proposed approach suitable for cross-project estimation?*

Estimating issue resolution time in new projects is often difficult due to lack of training data. One common technique to address this problem is training a model using data from a (source) project and applying it to the new (target) project. We employed this technique and performed 20 cross-project estimation experiments.

Table 6 reports the MAE produced by our approach in cross-project settings. For example, when we used the issues from Hadoop Common for training to obtain an estimation model and then applied this model for the issues in Hadoop HDFS, our approach achieved 19.31 MAE. In the within-project setting, i.e. the training was done using Hadoop HDFS, our approach achieved 17.80 MAE (see Table 2). The decreased performance in this case was relatively small (only 7%), which is also observed in the other cases. These results suggest that our approach can be used for cross-project estimation with a small sacrifice in accuracy. We also observe that estimations done cross the four projects in Hadoop were more accurate than those performed cross one of the four projects in Hadoop and the Apache Mesos project. This may be due to the fact that the four Hadoop projects share many commonalities than those in a totally different project like Mesos.

Table 6: MAE produced by our approach MOIRTE in cross-project settings, trained using a project in the row and tested on a project in the column.

Project	**Common**	**HDFS**	**Mapreduce**	**Yarn**	**Mesos**
Common	▨	19.31	27.54	34.20	42.08
HDFS	28.30	▨	30.23	33.94	42.09
Mapreduce	29.45	21.86	▨	32.72	40.47
Yarn	27.21	22.90	27.56	▨	39.90
Mesos	30.10	26.87	35.63	35.15	▨

4.4 Threats to validity

To mitigate threats to construct validity, we used real world data from issues reported in large open source projects. We collected the common issue features and the actual time took to resolve issues. In terms of the conclusion validity, we carefully selected unbiased error measures and applied a number of statistical tests to verify our assumptions [2]. Our study was performed on five datasets of different sizes. Furthermore, we carefully followed recent best practices in evaluating effort estimation models (e.g. [25]) to miminize conclusion instability. Another threat is related to the random initialization of the first generation. Therefore, a single run of an experimental study may deliver results that can be affected either by the favorable initial random selection or the bad randomly selected point [5]. To avoid this problem, we have conducted a multiple run (30 runs in this study) and chose the median result.

To overcome the external validity threat, we have considered a large number of issues from five different projects.

The size and the complexity of these issues are also significantly diverse. By this way, different contexts can be characterised by some specific projects and human factors (e.g. team structure and communication, time and other constraints, and so on). However, we cannot claim that our datasets are representative of all kind of software projects, and that our results can generalize to all software projects, especially those in commercial settings.

5. RELATED WORK

Predicting issue resolution time could be considered as a form of software effort estimation. Research in software effort estimation dates back several decades and they can generally be divided into model-based and expert-based methods [24]. Model-based approaches leverages data from old projects to make predictions about new projects. Expert-based methods rely on human expertise to make such judgements. Most of the existing work (e.g. [10, 17, 28]) in effort estimation focus on waterfall-like software development. These approaches estimate the effort required for developing a complete software system, relying on a set of features manually designed for characterizing a software project. This contrasts with our work since we estimate a single issue in the project at a time.

There is an emerging interest in predicting the fixing time of a bug, which was initiated by the work of Weiss *et. al.* [32]. These work (e.g.[7, 30, 33, 34] use machine learning techniques (e.g. kNN in [32] or Random Forests in [3, 16, 21]) to build their prediction models. For example, the work in [32] estimates the fixing time of a bug by finding the previous bugs that have similar description to the given bug (using text similar techniques) and using the known time of fixing those previous bugs. Using decision trees and other machine learning techniques, the work in [27] predict the lifetime of Eclipse bugs based on several primitive features of a bug such as severity, component, and number of comments. The work in [21] explored a different set of issue features including location, reporter and description. The time when the prediction is made also affect the predictive performance as shown in the study in [12]. They tested the predictive models with initial bug report data as well as those with post-submission information and found that inclusion of post-submission bug report data of up to one month can further improve prediction models. Most of those techniques used classifiers which do not deal with continuous response variables, they need to discretize the fix-time into categories, e.g. within 1 month, 1 year and more than 1 year as in [21]. This is one of the key difference to our work since we are able to predict the exact resolution time. In addition, our work offers an alternative in which we propose a search-based evolutionary approach to the problem.

Our work falls in the area of search-based software engineering where substantial work has been done and a range of them focused on software effort estimation. Most of the recent work in the context of effort prediction can be found in the review paper of Ferrucci et. al. [10]. While most of existing work in this space use single-objective search, a few of them (e.g. [26, 28]) has recently proposed multi-objective search approach to effort estimation. For example, a recent study done by Sarro *et. al.* [28] employed NSGA-II with two objectives sum of absolute error and confidence interval. Our work is largely inspired by Sarro *et. al.*'s work in the use of multi-objective search, sum of absolute error as an objective function, and standardized performance measures such as MAE, SA, Wilcoxon and effect size tests. There are however several key differences from our work and Sarro *et. al.*'s work. First, we built effort (time) estimation models for a single software issue rather than for the whole project (as done by Sarro *et. al.*). This makes our work more relevant and applicable to the modern agile software development settings where the focus is at the issue level. Second, we used the tree size as the second objective function to simultaneously manage the parsimonious and accuracy of generated estimation models. Hence, our approach does not impose any fixed structure or depth on the candidate models, which is different from Sarro *et. al.*'s approach. In addition, while Sarro *et. al.* used only three mathematical operators, we used a wider range of thirteen mathematical operators and thus accommodate a larger search space.

6. CONCLUSIONS AND FUTURE WORK

We have proposed a multi-objective search-based approach to estimate issue resolution time. Our approach leverages evolutionary algorithms to find robust estimation models. The search is guided simultaneously by two contrasting objectives: maximizing the accuracy of an estimation model and minimizing the complexity of the estimation model. A comprehensive evaluation done in five large open source projects with 8,260 issues demonstrates that our approach significantly (i.e. with $p < 0.001$) outperforms not only the three common naive baselines but also three state-of-the-art techniques. Our results also demonstrate the benefit of using the complexity measure as the second fitness function since this approach produced more accurate but less complex estimation model than the single-objective approach did. Results from our cross-project experiments also suggest that our approach is also applicable for cross-project estimation.

Future work would involve validating these results with some additional projects, especially those in the commercial settings. We also plan to investigate the use of other objectives such as confidence interval (as used in previous work [28]) and other complexity measures (e.g. order of nonlinearity). To compare the Pareto front of solutions returned by those approaches against those by our approach, we plan to use the evaluation measures recently proposed by Ferrucci et al.[9] such as contributions, hypervolume, and generational distance. We will also explore the use of other multi-objective evolutionary algorithms as part of our future work.

7. REFERENCES

[1] A. Arcuri and L. Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 1–10. IEEE, 2011.

[2] A. Arcuri and L. Briand. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(3):219–250, 2014.

[3] M. Choetkiertikul, H. K. Dam, T. Tran, and A. Ghose. Characterization and prediction of issue-related risks in software projects. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR)*, pages 280–291. IEEE, 2015.

[4] J. Cohen. Statistical power analysis for the behavioral sciences. 1988, hillsdale, nj: L. *Lawrence Earlbaum Associates*, 2.

[5] M. de Oliveira Barros and A. C. Dias-Neto. 0006/2011-threats to validity in search-based software engineering empirical studies. *RelaTe-DIA*, 5(1), 2011.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[7] A. Dehghan, K. Blincoe, and D. Damian. A hybrid model for task completion effort estimation. In *Proceedings of the 2nd International Workshop on Software Analytics*, pages 22–28. ACM, 2016.

[8] J. J. Dolado, D. Rodriguez, M. Harman, W. B. Langdon, and F. Sarro. Evaluation of estimation models using the minimum interval of equivalence. *Applied Soft Computing*, pages 1–12, 2016.

[9] F. Ferrucci, M. Harman, J. Ren, and F. Sarro. Not going to take this anymore: Multi-objective overtime planning for software engineering projects. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 462–471, Piscataway, NJ, USA, 2013. IEEE Press.

[10] F. Ferrucci, M. Harman, and F. Sarro. Search-based software project management. In *Software Project Management in a Changing World*, pages 373–399. Springer, 2014.

[11] C. Gathercole and P. Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In *Proceedings of the 1st Annual Conference on Genetic Programming*, pages 291–296, Cambridge, MA, USA, 1996. MIT Press.

[12] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, pages 52–56. ACM, 2010.

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[14] P. Hooimeijer and W. Weimer. Modeling bug report quality. In *Proceedings of the 22 IEEE/ACM international conference on Automated software engineering (ASE)*, pages 34 – 44. ACM Press, nov 2007.

[15] S.-J. Huang and N.-H. Chiu. Optimization of analogy weights by genetic algorithm for software effort estimation. *Information and software technology*, 48(11):1034–1045, 2006.

[16] R. Kikas, M. Dumas, and D. Pfahl. Using dynamic and contextual features to predict issue lifetime in github projects. In *Proceedings of the 13th International Workshop on Mining Software Repositories*, pages 291–302. ACM, 2016.

[17] E. Kocaguneli, T. Menzies, and J. W. Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering*, 38(6):1403–1416, 2012.

[18] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[19] W. B. Langdon, J. Dolado, F. Sarro, and M. Harman. Exact mean absolute error of baseline predictor, marp0. *Information and Software Technology*, 73:16–18, 2016.

[20] C. Lokan. What should you optimize when building an estimation model? In *11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 1–10. IEEE, 2005.

[21] L. Marks, Y. Zou, and A. E. Hassan. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, page 11. ACM, 2011.

[22] D. R. McCallum and J. L. Peterson. Computer-based readability indexes. In *Proceedings of the ACM'82 Conference*, pages 44–48. ACM, 1982.

[23] T. Menzies. Occam's razor and simple software project management. In G. Ruhe and C. Wohlin, editors, *Software Project Management in a Changing World*, pages 447–472. Springer, 2014.

[24] T. Menzies, Z. Chen, J. Hihn, and K. Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, 32(11):883–895, 2006.

[25] T. Menzies and M. Shepperd. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering*, 17(1):1–17, 2012.

[26] L. L. Minku and X. Yao. Software effort estimation as a multiobjective learning problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(4):35, 2013.

[27] L. D. Panjer. Predicting eclipse bug lifetimes. In *Proceedings of the Fourth International Workshop on mining software repositories*, page 29. IEEE Computer Society, 2007.

[28] F. Sarro, A. Petrozziello, and M. Harman. Multi-objective software effort estimation. In *Proceedings of the 38th International Conference on Software Engineering*, pages 619–630. ACM, 2016.

[29] M. Shepperd and S. MacDonell. Evaluating prediction systems in software project estimation. *Information and Software Technology*, 54(8):820–827, 2012.

[30] Y. Tian, D. Lo, and C. Sun. Drone: Predicting priority of reported bugs by multi-factor analysis. In *ICSM*, pages 200–209, 2013.

[31] A. Vargha and H. D. Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[32] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 1. IEEE Computer Society, 2007.

[33] X. Xia, D. Lo, E. Shihab, X. Wang, and B. Zhou. Automatic, high accuracy prediction of reopened bugs. *Automated Software Engineering*, 22(1):75–109, 2015.

[34] X. Xia, D. Lo, X. Wang, X. Yang, S. Li, and J. Sun. A comparative study of supervised learning algorithms for re-opened bug prediction. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 331–334. IEEE, 2013.